Linux Kernel Hackers' Guide

Due to the fact that nearly every post to this site recently has been either by rude cracker-wannabes asking how to break into other people's systems or a request for basic technical support, posting to the KHG has been disabled, probably permanently. For now, you can read old posts, but you cannot send replies. In any case, there are now far better resources available.

Go get the real thing!

Alessandro Rubini wrote *Linux Device Drivers*, which is what the KHG could have been (maybe) but isn't. If you have a question and can't find the answer here, go get a copy of *Linux Device Drivers* and read it--chances are that when you are done, you will not need to ask a question here.

Run, don't walk to get a copy of this book.

The Linux Kernel

Go read The Linux Kernel if you want an introduction to the Linux kernel that is better than the KHG. It is a great complement to *Linux Device Drivers*. Read it.

Table of Contents

Tour of the Linux Kernel
: This is a somewhat incomplete tour of the Linux Kernel, based on Linux 1.0.9 and the 1.1.*x* development series. Most of it is still relevant.

Device Drivers
: The most common Linux kernel programming task is writing a new device driver. The great majority of the code in the kernel is new device drivers; between 1.2.13 and 2.0 the size of the source code more than doubled, and most of that was from adding device drivers.

Filesystems
: Adding a filesystem to Linux doesn't have to involve magic...

Linux Memory Management
: A few outdated documents, and one completely new one by David Miller on the Linux cache flush architecture.

How System Calls Work on Linux/i86
: Although this was written while Linux 0.99.2 was current, it still applies. A few filenames may need updating. **find** is your friend--just respond with the changes and they will be added.

Other Sources of Information
: The KHG is just one collection of information about the Linux kernel. There are others!

Membership and Subscription

At the bottom of the page, you will notice two hyperlinks (among several others): **Subscribe** and **Members**. Using the KHG to its fullest involves these two hyperlinks, even though you are not required to be a member to read these pages and post responses.

Membership

HyperNews membership is site-wide. That is, you only need to sign up and become a member once for the entire KHG. It doesn't take much to be a member. Each member is identified by a unique name, which can either be a nickname or an email address. We suggest using your email address; that way it will be unique and easy to remember. On the other hand, you may want to choose a nickname if you expect to be changing your email address at any time.

We also want your real name, email address, and home page (if you have one). You can give us your phone and address if you want. You will be asked to choose a password. You can change any of these items at any time by clicking on the Membership hyperlink again.

Subscription

Subscribing to a page puts you on a mailing list to be sent notification of any new responses to the page to which you are subscribed. You subscribe separately to each page in which you are interested by clicking the **Subscription** link on the page to which you want to subscribe. You are also subscribed, by default, to pages that you write.

When you subscribe to a page, you subscribe to that page **and** all of its responses.

Contributing

Please respond to these pages if you have something to add. Think of posting a response rather like posting to an email list, except that an editor might occasionally come along to clean things up and/or put them in the main documents' bodies. So if you would post it to an email list in a similar discussion, it is probably appropriate to post here.

In order to make reading these pages a pleasure for everyone, any incomprehensible, unrelated, outdated, abusive, or other completely unnecessary post may be removed by an administrator. So if you have a message that would be inappropriate on a mailing list, it's probably also inappropriate here.

The administrators have the final say on what's appropriate. We don't expect this to become an issue...

About the new KHG

The *Linux Kernel Hackers' Guide* has changed quite a bit since its original conception four years ago. I struggled along with the help of many other hackers to produce a document that lived primarily on paper, and was intended to document the kernel in much the same way that a program's user guide is intended to document the program for users.

It was less successful than most user guides, for a number of reasons:

- I was working on it part time, and was otherwise busy.
- The Linux kernel is a moving target.
- I am not personally capable of documenting the entire Linux kernel.
- I became far too concerned with making the typesetting pretty, getting bogged down in details *and* making the document typographically noisy at the same time.

I floundered around, trying to be helpful, and made at least one right decision: most of the people who needed to read the old KHG needed to write device drivers, and the most fully-developed part of the KHG was the device driver section.

There is a clear need for further development of the KHG, and it's clear that my making it a monolithic document stood in the way of progress. The KHG is now a series of more or less independent web pages, with places for readers to leave comments and corrections that can be incorporated in the document at the maintainer's leisure--and are available to readers before they are incorporated.

The KHG is now completely web-based. There will be no official paper version. You need kernel source code nearby to read the KHG anyway, and I want to shift the emphasis from officially documenting the Linux kernel to being a learning resource about the Linux kernel--one that may well be useful to other people who want to document one part or another of the Linux kernel more fully, as well as to people who just want to hack the kernel.

Enjoy!

Messages

349. Loading shared objects - How? *by Wesley Terpstra*
342. How can I see the current kernel configuration? *by Melwin*
    1. My mouse no work in X windows *by alfonso santana*
340. The crash(1M) command in Linux? *by Dmitry*
338. Where can I gen detailed info on VM86 *by Sebastien Plante*
335. How to print floating point numbers from the kernel? *by pkunisetty@hotmail.com*
333. PS/2 Mouse Operating in Remote Mode *by Andrei Racz*
331. basic module *by vano0023@tc.umn.edu*
329. How to check if the user is local? *by jb@nicol.ml.org*
328. Ldt & Privileges *by Ganesh*
326. skb queues *by Rahul Singh*
323. Page locking (for DMA) and process termination? *by Espen Skoglund*
322. SMP code *by 97yadavm@scar.utoronto.ca*
319. Porting GC: Difficulties with pthreads *by Talin*
314. Linux for "Besta - 88"? *by Dmitry*
    1. MVME147 Linux *by Edward Tulupnikov*
313. /proc/locks *by Marco Morandini*
310. syscall *by ppappu@lrc.di.epfl.ch*
308. How to run a bigger kernel ? *by Kyung D. Ryu*
300. Linux Terminal Device Driver *by Nils Appeldoorn*
    1. Terminal DD *by Doug McNash*
297. DMA to user allocated buffer ? *by Chris Read*
    1. allocator-example in A.Rubini's book *by Thomas Sefzick*
293. Patching problems *by Maryam*
    1. Untitled *by welch@mcmail.com*
290. Ethernet Collision *by jerome bonnet*
    1. Ethernet collisions *by Juha Laine*
289. Segmentation in Linux *by Andrew Sampson*
288. How can the kernel copy directly data from one process to another process? *by Jürgen Zeller*
    1. Use the /Proc file system *by marty@twsu.campus.mci.net*
286. Remapping Memory Buffer using vmalloc/vma_nopage *by Brian W. Taylor*

    1. Fixed.... strncpy to blame *by Brian W. Taylor*
283. Does memory area assigned by "vmalloc()" get swapped to disk? *by Saurabh Desai*
    1. Lock the pages in memory *by balaji@ittc.ukans.edu*
    -> How about assigning a fixed size array...does it get swapped too? *by saurabh desai*
282. Creative Lab's DVD Encore *by Brandon*
274. TCP sliding window *by Olivier*
273. Packets and default route versus direct route *by Steve Resnick*
269. IPv6 description - QoS Implementation - 2 IP Queues *by wehrle*
    2. See the kernel IPv4 implementation documentation *by Juha Laine*
268. writing to user file directly from kernel space, How can it be done? *by Johan*
267. how can i increase the number of processes running? *by ElmerFudd*
261. How do I change the amount of time a process is allowed before it is pre-empted? *by Escher@dn101aw.cse.eng.auburn.edu*
260. Network device stops after a while *by Andrew Ordin*
    1. Untitled *by Andrew*
259. Does MMAP work with Redhat 4.2? *by Guy*
    1. Yes, it works just fine. *by Michael K. Johnson*
        3. What about mprotect? *by Sengan Baring-Gould*
        2. It Works! Thanks! *by Guy*
256. multitasking *by Dennis J Perkins*
    1. Answer *by David Welch*
    -> multitasking *by Dennis J Perkins*
    -> answer *by David Welch*
247. linux on sparc *by darrin hodges*
241. How to call a function in user space from inside the kernel ? *by Ronald Tonn*
    1. How to call a user routine from kernel mode *by David Welch*
240. Can I map kernel (device driver) memory into user space ? *by Ronald Tonn*
237. driver x_open,x_release work, x_ioctl,x_write don't *by Carl Schwartz*
    1. Depmod Unresolved symbols? *by Carl Schwartz*
235. How to sleep for x jiffies? *by Trent Piepho*
    1. Use add_timer/del_timer (in kernel/sched.c) *by Amos Shapira*
234. Adding code to the Linux Kernel *by Patrick*
    1. /dev/random *by Simon Green*
231. MSG_WAITALL flag *by Leonard Mosescu*
230. possible bug in ipc/msg.c *by Michael Adda*
225. scheduler Question *by Arne Spetzler*
    1. Untitled *by Ovsov*
    -> thanks *by arne spetzler*
221. File Descriptor Passing? *by The Llamatron*
220. Linux SMP Scheduling *by Angela*
    2. Finding definitions in the source *by Felix Rauch*

1. Re: Linux SMP Scheduling *by Franky*

217. Difference between ELF and kernel file *by Thomas Prokosch*

216. How kernel communicates with outside when it's started? *by Xuan Cao*

   1. Printing to the kernel log *by Thomas Prokosch*

213. The way from a kernel hackers' idea to an "official" kernel? *by Roger Schreiter*

   1. linux-kernel@vger.rutgers.edu *by Michael K. Johnson*

212. Curious about sleep_on_interruptible() in ancient kernels. *by Colin Howell*

208. Server crashes using 2.0.32 and SMP *by Steve Resnick*

   1. Debugging server crash *by Balaji Srinivasan*

   -> More Information *by Steve Resnick*

   -> it should not have happenned... *by Balaji Srinivasan*

207. Signals ID definitions *by Franky*

206. the segment D000 is not visible *by martinv2@ctima.uma.es*

205. ICMP - Supressing Messages in 2.1.82 Kernel *by Brent Johnson*

   1. Change /etc/syslog.conf *by Balaji Srinivasan*

203. Modem bits *by Franky*

   1. Untitled *by Kostya*

200. I need some way to measure the time a process spend in READY QUEUE *by Leandro Gelasi*

197. How to make sockets work whilst my process is in kernel mode? *by Mikhail Kourinny*

193. Realtime Problem *by Uwe Gaethke*

   1. SCHED_FIFO scheduling *by Balaji Srinivasan*

190. inodes *by Ovsov*

186. Difference between SOCK_RAW SOCK_PACKET *by Chris Leung*

   1. SOCK_PACKET *by Eddie Leung*

185. Need additional termcap entries for TERM=linux *by Karl Bullock*

184. Question on Umount or sys_umount *by teddy*

183. Passing file descriptors to the kernel *by Pradeep Gore*

   1. A way to "transform" a file descriptor into a struct file* in a user process *by Lorenzo Cavallaro*

181. Dead Man Timer *by Jody Winston*

179. raw sockets *by lightman*

178. a kernel-hacking newbie *by Bradley Lawrence*

   2. A place to start.

   1. Modems in general *by Ian Carr-de Avelon*

176. How to write CD-ROM Driver ? Any Source Code ? *by Madhura Upadhya*

174. Measuring the scheduler overhead *by Jasleen Kaur*

173. Where can I find the tcpdump or snoop in linux? *by wangc@taurus*

   1. man which *by trajek@j00nix.org*

172. Timers don't work?? *by Joshua Liew*

   1. Timers Work... *by Balaji Srinivasan*

171. problem of Linux's bridge code *by wangc@taurus*

170. Documention on writing kernel modules *by Erik Nygren*
168. How to display a clock on my console? *by keco*
167. Difference between SCO and Linux drivers. *by M COTE*
165. Changing the scheduler from round robin to shortest job first for kernel 2.0 and up *by royal_and_mary_harrell@msn.com*
    2. Improving the Scheduer *by Lee Ingram*
        1. Improving the Scheduler : use QNX-like *by Leandro Gelasi*
    1. Re: Changing the sched. from round robin to shortest job first for kernel 2.0 and up. *by Pirasenna V.T.*
164. meanings of file->private_data *by ncuandre@ms14.hinet.net*
162. /dev/signalprocess *by flatmax*
161. how to track VM page access sequence? *by shawn*
160. Whats the difference between dev_tint(dev) and mark_bh(NET_BH)? *by Jaspreet Singh*
159. PCI *by mullerc@iname.com*
    1. RE: PCI *by Armin A. Arbinger*
158. Can I make syscall from inside a kernel module? *by Shawn Chang*
    3. Re: Can I make syscall from inside a kernel module? *by Massoud Asgharifard*
        1. Make a syscall despite of wrong fs!! *by Mikhail Kourinny*
    2. code snip to make a sys_* call from a module *by Pradeep Gore*
    1. Dont use system calls within kernel...(esp sys_mlock) *by Balaji Srinivasan*
157. Untitled *by Steve Durst*
154. RAW Sockets (Art)
153. use phy mem *by WYB*
151. HyperNews for RH Linux ? *by Eigil Krogh Sorensen*
    1. Not really needed *by Cameron*
150. about raw ethernet frame: how to do it ? *by crbild@smc.it*
149. process table *by Blaz Novak*
148. Stream drivers *by Nick Egorov*
    3. Streams drivers
    1. Stream in Solaris *by cai.yu@rdc.etc.ericsson.se*
143. Xircom External Ethernet driver anywhere? *by mike head*
140. interruptible_sleep_on() too slow! *by Bill Blackwell*
    1. wrong functions *by Michael K. Johnson*
139. creating a kernel relocatable module *by Simon Kittle*
138. Up to date serial console patches *by Simon Green*
136. Kernel-Level Support for Checkpointing on Linux? *by Argenis R. Fernandez*
    1. Working on it. *by Jan Rychter*
135. Problem creating a new system call *by sauru*
    3. How did the file /arch/i386/kernel/entry.S do its job *by Wang Ju*
    2. system call returns "Bad Address". Why? *by sauru*
        1. Re:return values *by C.H.Gopinath*

2. Re:return values *by Sameer Shah*
1. possible reason for segmentation fault
1. Creating a new sytem call: solution *by C.H.Gopinath*
2. problem with system call slot 167 *by Todd Medlock*
1. Kernel Debuggers for Linux *by sauru*
133. Resetting interface counters *by Keith Dart*
130. writing/accessing modules *by Jones MB*
1. Use a device driver and read()/write()/ioctl() *by Michael K. Johnson*
-> getting to the kernel's memory *by Jones MB*
-> use buffers! *by Rubens*
124. Help with CPU scheduler! *by Lerris*
1. Response to "Help with CPU scheduler!" *by Jeremy Impson*
-> Response to "Help with CPU scheduler!" (Redux) *by Jeremy Impson*
117. calling interupts from linux *by John J. Binder*
1. You can't *by Michael K. Johnson*
-> Calling BIOS interrupts from Linux kernel *by Ian Collier*
-> Possible, but takes work *by Michael K. Johnson*
-> VBE video driver *by Ian Collier*
-> VM86 mode at which abstraction level? *by Michael K. Johnson*
116. DVD-ROM and Linux? (sorry if it's off topic...) *by Joel Hardy*
3. DVD-ROM and linux *by Yuqing_Deng@brown.edu*
2. Response to DVD and Mpeg in Linux *by Mike Corrieri*
1. DVD Encryption *by Mark Treiber*
-> Untitled *by Tim*
-> DVD?
115. Kernel Makefile Configuration: how? *by Simon Green*
2. How to add a driver to the kernel ? *by jacek Radajewski*
1. See include/linux/autoconf.h *by Balaji Srinivasan*
113. Multiprocessor Linux *by Davis Terrell*
1. Building an SMP kernel *by Michael K. Johnson*
-> SMP and module versions *by linux@catlimited.com*
111. Improving event timers? *by bodomo@hotmail.com*
109. measuring time to load a virtual mem page from disk *by kandr*
94. using cli/sti() and save_flags/restore_flags() *by george*
93. Protected Mode *by ac*
2. 'Developers manual' from Intel(download)... *by Mats Odman*
1. Advanced 80386 Programming Techniques *by Michael K. Johnson*
92. DMA buffer sizes *by nomrom@hotmail.com*
2. DMA limits *by Albert Cahalan <acahalan at cs.uml.edu>*
1. Not page size, page order *by Michael K. Johnson*
91. Problem Getting the Kernel small enough *by afish@tdcdesigncorps.com*

    2. Check it's the right file, zImage not vmlinux *by Cameron*
    1. Usually easy, but.... *by Ian Carr-de Avelon*
89. How to create /proc/sys variables? *by Orlando Cantieni*
88. Linux for NeXT black? *by Dale Amon*
87. vremap() in kernel modules? *by Liam Wickins*
86. giveing compatiblity to win95 for ext2 partitions (for programmers forced to deal with both) *by pharos*
    2. Well, What's the status of the Windows / Dos driver for Ext2? *by Brock Lynn*
    1. Working on it! *by ibaird*
        1. revision *by ibarid*
        -> Untitled *by Olaf*
84. setsockopt() error when triying to use ipfwadm for masquerading *by omaq@encomix.es*
    1. Re: masquerading *by Charles Barrasso*
83. reset the irq 0 timer after APM suspend *by Dong Chen*
    1. Re: fixed, patch for kernel 2.0.30 *by Dong Chen*
77. Source Code in C for make Linux partitions. *by Limbert Sanabria*
    1. Untitled *by lolley*
76. How can I "cheat" and change the IP address (src,dest) in the sent socket? *by Rami*
    5. Transparent Proxy *by Zygo Blaxell*
    4. Untitled *by qwzhang@public2.bta.net.cn*
    3. Untitled *by navin97@hotmail.com*
        2. Changing your IP address is easy, but... *by Zygo Blaxell*
        1. You have to know a bit of C (if u wanna learn) ;) *by Lorenzo Cavallaro*
    2. Untitled
    1. Do it in the kernel *by Michael K. Johnson*
74. Where is the source file for accept() *by kaixu@hocpa.ho.lucent.com*
    1. Here, in /usr/src/linux/net/socket.c *by wumin@netchina.co.cn*
72. How can I use RAW SOCKETS in UNIX? *by Rami*
    1. Re: Raw sockets *by genie@risq.belcaf.minsk.by*
69. the KHG in spanish? *by Jorge Alvarado Revatta*
    2. Si tenga preguntas, quisa yo pueda ayudarte. *by KernelJock*
        3. Tengo una pregunta *by riderghost@rocketmail.com*
        2. Español *by LL2*
    1. No esta aqui! Pero... *by Michael K. Johnson*
67. How to get a Memory snapshot ? *by Manuel Porras Brand*
    1. Why not to get a memory snapshot? *by Jukka Santala*
    -> Why you would want to get a memory snapshot *by Dave M.*
66. resources hard limits *by castejon@kbw350.chem.yale.edu*
    1. Setting resource limits *by Jukka Santala*
65. How to invalidate a chache page *by Gerhard Uttenthaler*
    1. Read the rest of the KHG! *by Michael K. Johnson*

64. Where are the tunable parameters of the kernel? *by demiguel@robot3.cps.unizar.es*
    1. Kernel tunable parameters *by Jukka Santala*
62. How can my device driver access data structures in user space? *by Stephan Theil*
    1. Forced Cast data type *by Wang Ju*
61. Problem in doing RAW SOCKET Programming *by anjali sharma*
    1. Problem with ICMP echo-request/reply *by Raghavendra Bhat*
59. Tunable Kernel Parameters? *by dennyf@bmn.net*
    2. Increasing number of files in system *by Simon Cooper*
        1. Increasing number of open files parameter *by Simon Cooper*
    1. sysctl in Linux *by Jukka Santala*
        1. Setting and getting kernel vars *by kbrown@csuhayward.edu*
58. ELF matters *by Carlos Munoz*
    1. Information about ELF Internals *by Pat Ekman*
57. Droping Packets *by Charles Barrasso*
    1. [Selectively] Droping Packets *by Jose R. cordones*
56. The /proc/profile *by Charles Barrasso*
    1. readprofile systool *by Jukka Santala*
55. Can you block or ignore ICMP packets? *by HackOps@nutnet.com*
    4. ICMP send rate limit / ignoring *by Jukka Santala*
        1. Omission in earlier rate-limit... *by Jukka Santala*
        -> Patch worked... *by Jukka Santala*
    3. Using ipfwadm *by Charles Barrasso*
        1. ipfwadm configuration utility *by Sonny Parlin*
    1. Icmp.c and kernal ping replies *by Don Thomas*
52. encaps documentation *by Kuang-chun Cheng*
51. Mounting Caldrea OpenDOS formatted fs's *by Trey Childs*
49. finding the address that caused a SIGSEGV. *by Ben Shelef*
47. sti() called too late. *by Erik Thiele*
    1. sti() called too late. *by Gadi Oxman*
38. Module Development Info? *by Mark S. Mathews*
    1. Needed here too *by ajay*
    2. Help needed here too! *by ajay*
35. Need quicker timer than 100 ms in kernel-module *by Erik Thiele*
    1. 10 ms timer patch *by Reinhold J. Gerharz*
        2. please send me 10 ms timer patch *by Tolga Ayav*
        1. Please send me the patch *by Jin Hwang*
            1. UTIME: Microsecond Resolution Timers *by BalajiSrinivasan*
34. Need help with finding the linked list of unacked sk_buffs in TCP *by Vijay Gupta*
31. Partition Type *by Suman Ball*
30. New document on exception handling *by Michael K. Johnson*
29. How to make paralelism in to the kernel? *by Delian Dlechev*

27. readv/writev & other sock funcs *by Dave Wreski*
25. I'd like to see the scheduler chapter *by Tim Bird*
    1. Untitled *by Vijay Gupta*
    3. Go ahead! *by Michael K. Johnson*
21. Unable to access KHG, port 8080 giving problem. *by Srihari Nelakuditi*
    1. Get a proxy *by Michael K. Johnson*
20. proc fs docs? *by David Woodruff*
    1. Examples code as documentation *by Jeremy Impson*
18. What is SOCK_RAW and how do I use it? *by arkane*
    1. What raw sockets are for. *by Cameron MacKinnon*
15. Linux kernel debugging *by yylai@hk.net*
    2. GDB for Linux *by David Grothe*
        2. Another kernel debugging tool *by David Hinds*
            2. Kernel debugging with breakpoints *by Keith Owens*
            -> Need help for debugging *by C.H.Gopinath*
        1. gdb debugging of kernel now available *by David Grothe*
    1. Device debugging *by alombard©iiic.ethz.ch*
9. Realtime mods anyone? *by bill duncan*
    7. Summary of Linux Real-Time Status *by Markus Kuhn*
    6. Hard real-time now available *by Michael K. Johnson*
        2. Shortcomings of RT-Linux *by Balaji Srinivasan*
        1. Firm Realtime available *by Balaji Srinivasan*
    5. found some hacks ?!? *by Mayk Langer*
        2. I want to know how to hack Red Hat Linux Release 5.0 *by Kevin*
    4. POSIX.4 scheduler *by Peter Monta*
        1. cli()/sti() latency, hard numbers *by Ingo Molnar*
    2. Realtime is already done(!) *by Kai Harrekilde-Petersen*
    1. 100 ms real time should be easy *by jeff millar*
        1. Real-Time Applications with Linux POSIX.4 Scheduling *by P. Woolley*
7. Why can't we incorporate new changes in linux kernel in KHG ? *by Praveen Kumar Dwivedi*
    1. You can! *by Michael K. Johnson*
3. Kernel source code *by Gabor J.Toth*
    1. The sounds of silence... *by Gabor J.Toth*
        1. Breaking the silence :) *by Kyle Ferrio*
            1. Scribbling in the margins *by Michael K. Johnson*
        2. It requires thought... *by Michael K. Johnson*
    2. Kernel source is already browsable online *by Axel Boldt*
2. Need easy way to download whole KHG
    5. KHG being mirrored nightly for download! *by Michael K. Johnson*
    2. postscript version of these documents? *by Michael Stiller*
        1. Sure! *by Michael K. Johnson*

- -> Not so Sure! *by jeff millar*
- -> Enough already! *by Michael K. Johnson*

1. Mirror packages are available, but that's not really enough *by Michael K. Johnson*
    4. Mirror whole KHG package, off line reading and Post to this site *by Kim In-Sung*
        2. Untitled *by Jim Van Zandt*
        1. That works. (using it now). Two tips: *by Richard Braakman*
            2. Appears to be a bug in getwww, though... *by Michael K. Johnson*
            - -> Sucking up to the wrong site... ;) *by Jukka Santala*

1. Help make the new KHG a success *by Michael K. Johnson*

Tour of the Linux kernel source

By Alessandro Rubini, rubini@pop.systemy.it

This chapter tries to explain the Linux source code in an orderly manner, trying to help the reader to achieve a good understanding of how the source code is laid out and how the most relevant unix features are implemented. The target is to help the experienced C programmer who is not accustomed to Linux in getting familiar with the overall Linux design. That's why the chosen entry point for the kernel tour is the kernel own entry point: system boot.

A good understanding of C language is required to understand this material, as well as some familiarity with both Unix concepts and the PC architecture. However, no C code will appear in this chapter, but rather pointers to the actual code. The finest issues of kernel design are explained in other chapters of this guide, while this chapter tends to remain an informal overview.

Any pathname for files referenced in this chapter is referred to the main source-tree directory, usually /usr/src/linux.

NEW Most of the information reported here is taken from the source code of Linux release 1.0. Nonetheless, references to later versions are provided at times. Any paragraph within the tour with the NEW image in front of it is meant to underline changes the kernel has undergone after the 1.0 release. If no such paragraph is present, then no changes occurred up to release 1.0.9-1.1.76.

MORE Sometimes a paragraph like this occurs in the text. It is a pointer to the right sources to get more information on the subject just covered. Needless to say, *the source* is the primary source.

Booting the system

When the PC is powered up, the 80x86 processor finds itself in real mode and executes the code at address 0xFFFF0, which corresponds to a ROM-BIOS address. The PC BIOS performs some tests on the system and initializes the interrupt vector at physical address 0. After that it loads the first sector of a bootable device to 0x7C00, and jumps to it. The device is usually the floppy or the hard drive. The preceding description is quite a simplified one, but it's all that's needed to understand the kernel initial workings.

The very first part of the Linux kernel is written in 8086 assembly language (boot/bootsect.S). When run, it moves itself to absolute address 0x90000, loads the next 2 kBytes of code from the boot device to address 0x90200, and the rest of the kernel to address 0x10000. The message ``Loading..." is displayed during system load. Control is then passed to the code in boot/Setup.S, another real-mode assembly source.

The setup portion identifies some features of the host system and the type of vga board. If requested to, it asks the user to choose the video mode for the console. It then moves the whole system from address 0x10000 to address 0x1000, enters protected mode and jumps to the rest of the system (at 0x1000).

The next step is kernel decompression. The code at 0x1000 comes from zBoot/head.S which initializes registers and invokes `decompress_kernel()`, which in turn is made up of zBoot/inflate.c, zBoot/unzip.c and zBoot/misc.c. The decompressed data goes to address 0x100000 (1 Meg), and this is the main reason why Linux can't run with less than 2 megs ram. [It's been done in 1 MB with uncompressed kernels; see Memory Savers--ED]

Encapsulation of the kernel in a gzip file is accomplished by Makefile and utilities in the zBoot directory. They are interesting files to look at.

Kernel release 1.1.75 moved the boot and zBoot directories down to arch/i386/boot. This change is meant to allow true kernel builds for different architectures. Nonetheless, I'll stick to i386-specific information.

Decompressed code is executed at address 0x1010000 **[Maybe I've lost track of physical addresses, here, as I don't know very well gas source code]**, where all the 32-bit setup is accomplished: IDT, GDT and LDT are loaded, the processor and coprocessor are identified, and paging is setup; eventually, the routine `start_kernel` is invoked. The source for the above operations is in boot/head.S. It is probably the trickiest code in the whole kernel.

Note that if an error occurs during any of the preceding steps, the computer will lockup. The OS can't deal with errors when it isn't yet fully operative.

`start_kernel()` resides in init/main.c, and never returns. Anything from now on is coded in C language, left aside interrupt management and system call enter/leave (well, most of the macros embed assembly code, too).

Spinning the wheel

After dealing with all the tricky questions, `start_kernel()` initializes all the parts of the kernel, specifically:

- Sets the memory bounds and calls `paging_init()`.
- Initializes the traps, IRQ channels and scheduling.
- Parses the command line.
- If requested to, allocates a profiling buffer.
- Initializes all the device drivers and disk buffering, as well as other minor parts.
- Calibrates the delay loop (computes the ``BogoMips" number).
- Checks if interrupt 16 works with the coprocessor.

Finally, the kernel is ready to `move_to_user_mode()`, in order to fork the `init` process, whose code is in the same source file. Process number 0 then, the so-called idle task, keeps running in an infinite idle loop.

The `init` process tries to execute /etc/init, or /bin/init, or /sbin/init.

If none of them succeeds, code is provided to execute ``/bin/sh /etc/rc" and fork a root shell on the first terminal. This code dates back to Linux 0.01, when the OS was made by the kernel alone, and no `login` process was available.

After `exec()`ing the init program from one of the standard places (let's assume we have one of them), the kernel has no direct control on the program flow. Its role, from now on is to provide processes with system calls, as well as servicing asynchronous events (such as hardware interrupts). Multitasking has been setup, and it is now init which manages multiuser access by `fork()`ing system daemons and login processes.

Being the kernel in charge of providing services, the tour will proceed by looking at those services (the ``system calls"), as well as by providing general ideas about the underlying data structures and code organization.

How the kernel sees a process

From the kernel point of view, a process is an entry in the process table. Nothing more.

The process table, then, is one of the most important data structures within the system, together with the memory-management tables and the buffer cache. The individual item in the process table is the `task_struct` structure, quite a huge one, defined in include/linux/sched.h. Within the `task_struct` both low-level and high-level information is kept--ranging from the copy of some hardware registers to the inode of the working directory for the process.

The process table is both an array and a double-linked list, as well as a tree. The physical implementation is a static array of pointers, whose length is `NR_TASKS`, a constant defined in include/linux/tasks.h, and each structure resides in a reserved memory page. The list structure is achieved through the pointers `next_task` and `prev_task`, while the tree structure is quite complex and will not be described here. You may wish to change `NR_TASKS` from the default vaue of 128, but be sure to have proper dependency files to force recompilation of all the source files involved.

After booting is over, the kernel is always working on behalf of one of the processes, and the global variable `current`, a pointer to a `task_struct` item, is used to record the running one. `current` is only changed by the scheduler, in kernel/sched.c. When, however, all procecces must be looked at, the macro `for_each_task` is used. It is conderably faster than a sequential scan of the array, when the system is lightly loaded.

A process is always running in either ``user mode" or ``kernel mode". The main body of a user program is executed in user mode and system calls are executed in kernel mode. The stack used by the process in the two execution modes is different--a conventional stack segment is used for user mode, while a fixed-size stack (one page, owned by the process) is used in kernel mode. The kernel stack page is never swapped out, because it must be available whenever a system call is entered.

System calls, within the kernel, exist as C language functions, their `official' name being prefixed by `` `sys_' ``. A system call named, for example, *`burnout`* invokes the kernel function `sys_`*`burnout`*`()`.

The system call mechanism is described in chapter 3 of this guide. Looking at `for_each_task` and `SET_LINKS`, in include/linux/sched.h can help understanding the list and tree structures in the process table.

Creating and destroying processes

A unix system creates a process though the `fork()` system call, and process termination is performed either by `exit()` or by receiving a signal. The Linux implementation for them resides in kernel/fork.c and kernel/exit.c.

Forking is easy, and fork.c is short and ready understandable. Its main task is filling the data structure for the new process. Relevant steps, apart from filling fields, are:

- getting a free page to hold the `task_struct`
- finding an empty process slot (`find_empty_process()`)
- getting another free page for the `kernel_stack_page`
- copying the father's LDT to the child
- duplicating `mmap` information of the father

`sys_fork()` also manages file descriptors and inodes.

The 1.0 kernel offers some vestigial support to threading, and the `fork()` system call shows some hints to that. Kernel threads is work-in-progress outside the mainstream kernel.

Exiting from a process is trickier, because the parent process must be notified about any child who exits. Moreover, a process can exit by being `kill()`ed by another process (these are Unix features). The file exit.c is therefore the home of `sys_kill()` and the vairious flavours of `sys_wait()`, in addition to `sys_exit()`.

The code belonging to exit.c is not described here--it is not that interesting. It deals with a lot of details in order to leave the system in a consistent state. The POSIX standard, then, is quite demanding about signals, and it must be dealt with.

Executing programs

After `fork()`ing, two copies of the same program are running. One of them usually `exec()`s another program. The `exec()` system call must locate the binary image of the executable file, load and run it. The word `load' doesn't necessarily mean ``copy in memory the binary image'', as Linux supports demand loading.

The Linux implementation of `exec()` supports different binary formats. This is accomplished through the `linux_binfmt` structure, which embeds two pointers to functions--one to load the executable and the other to load the library, each binary format representing both the executable and the library. Loading of shared libraries is implemented in the same source file as `exec()` is, but let's stick to `exec()` itself.

The Unix systems provide the programmer with six flavours of the `exec()` function. All but one of them can be implemented as library functions, and theLinux kernel implements `sys_execve()` alone. It performs quite a simple task: loading the head of the executable, and trying to execute it. If the first two bytes are ``#!'', then the first line is parsed and an interpreter is invoked, otherwise the registered binary formats are sequentially tried.

The native Linux format is supported directly within fs/exec.c, and the relevant functions are `load_aout_binary` and `load_aout_library`. As for the binaries, the function loading an ``a.out'' executable ends up either in `mmap()`ing the disk file, or in calling `read_exec()`. The former way uses the Linux demand loading mechanism to fault-in program pages when they're accessed, while the latter way is used when memory mapping is not supported by the host filesystem (for example the ``msdos'' filesystem).

NEW Late 1.1 kernels embed a revised msdos filesystem, which supports `mmap()`. Moreover, the `struct linux_binfmt` is a linked list rather than an array, to allow loading a new binary format as a kernel module. Finally, the structure itself has been extended to access format-related core-dump routines.

Accessing filesystems

It is well known that the filesystem is the most basic resource in a Unix system, so basic and ubiquitous that it needs a more handy name--I'll stick to the standard practice of calling it simply ``fs''.

I'll assume the reader already knows the basic Unix fs ideas--access permissions, inodes, the superblock, **mount**ing and **umount**ing. Those concepts are well explained by smarter authors than me within the standard Unix literature, so I won't duplicate their efforts and I'll stick to Linux specific issues.

While the first Unices used to support a single fs type, whose structure was widespread in the whole kernel, today's practice is to use a standardized interface between the kernel and the fs, in order to ease data interchange across architectures. Linux itself provides a standardized layer to pass information between the kernel and each fs module. This interface layer is called VFS, for ``virtual filesystem''.

Filesystem code is therefore split into two layers: the upper layer is concerned with the management of kernel tables and data structures, while the lower layer is made up of the set of fs-dependent functions, and is invoked through the VFS data structures. All the fs-independent material resides in the fs/*.c files. They address the following issues:

- Managing the buffer chache (buffer.c);
- Responding to the `fcntl()` and `ioctl()` system calls (fcntl.c and ioctl.c);
- Mapping pipes and fifos on inodes and buffers (fifo.c, pipe.c);
- Managing file- and inode-tables (file_table.c, inode.c);
- Locking and unlocking files and records (locks.c);
- Mapping names to inodes (namei.c, open.c);
- Implementing the tricky `select()` function (select.c);
- Providing information (stat.c);
- mounting and umounting filesystems (super.c);
- `exec()`ing executables and dumping cores (exec.c);
- Loading the various binary formats (bin_fmt*.c, as outlined above).

The VFS interface, then, consists of a set of relatively high-level operations which are invoked from the fs-independent code and are actually performed by each filesystem type. The most relevant structures are `inode_operations` and `file_operations`, though they're not alone: other structures exist as well. All of them are defined within include/linux/fs.h.

The kernel entry point to the actual file system is the structure `file_system_type`. An array of `file_system_type`s is embodied within fs/filesystems.c and it is referenced whenever a mount is issued. The function `read_super` for the relevant fs type is then in charge of filling a `struct super_block` item, which in turn embeds a `struct super_operations` and a `struct` *type*`_sb_info`. The former provides pointers to generic fs operations for the current fs-type, the latter embeds specific information for the fs-type.

NEW The array of filesystem types has been turned in a linked list, to allow loading new fs types as kernel modules. The function (`un-`)`register_filesystem` is coded within fs/super.c.

Quick Anatomy of a Filesystem Type

The role of a filesystem type is to perform the low-level tasks used to map the relatively high level VFS operations on the physical media (disks, network or whatever). The VFS interface is flexible enough to allow support for both conventional Unix filesystems and exotic situations such as the msdos and umsdos types.

Each fs-type is made up of the following items, in addition to its own directory:

- An entry in the `file_systems[]` array (fs/filesystems.c);
- The superblock include file (include/linux/*type*_fs_sb.h);
- The inode include file (include/linux/*type*_fs_i.h);
- The generic own include file (include/linux/*type*_fs.h});
- Two `#include` lines within include/linux/fs.h, as well as the entries in `struct super_block` and `struct inode`.

The own directory for the fs type contains all the real code, responsible of inode and data management.

MORE The chapter about procfs in this guide uncovers all the details about low-level code and VFS interface for that fs type. Source code in fs/procfs is quite understandable after reading the chapter.

We'll now look at the internal workings of the VFS mechanism, and the minix filesystem source is used as a working example. I chose the minix type because it is small but complete; moreover, any other fs type in Linux derives from the minix one. The ext2 type, the de-facto standard in recent Linux installations, is much more complex than that and its exploration is left as an exercise for the smart reader.

When a minix-fs is mounted, `minix_read_super` fills the `super_block` structure with data read from the mounted device. The `s_op` field of the structure will then hold a pointer to `minix_sops`, which is used by the generic filesystem code to dispatch superblock operations.

Chaining the newly mounted fs in the global system tree relies on the following data items (assuming `sb` is the `super_block` structure and `dir_i` points to the inode for the mount point):

- `sb->s_mounted` points to the root-dir inode of the mounted filesystem (`MINIX_ROOT_INO`);
- `dir_i->i_mount` holds `sb->s_mounted`;
- `sb->s_covered` holds `dir_i`

Umounting will eventually be performed by `do_umount`, which in turn invokes `minix_put_super`.

Whenever a file is accessed, `minix_read_inode` comes into play; it fills the system-wide `inode` structure with fields coming form `minix_inode`. The `inode->i_op` field is filled according to `inode->i_mode` and it is responsible for any further operation on the file. The source for the minix functions just described are to be found in fs/minix/inode.c.

The `inode_operations` structure is used to dispatch inode operations (you guessed it) to the fs-type specific kernel functions; the first entry in the structure is a pointer to a `file_operations` item, which is the data-management equivalent of `i_op`. The minix fs-type allows three instances of inode-operation sets (for direcotries, for files and for symbolic links) and two instances of file-operation sets (symlinks don't need one).

Directory operations (`minix_readdir` alone) are to be found in fs/minix/dir.c; file operations (read and write) appear within fs/minix/file.c and symlink operations (reading and following the link) in fs/minix/symlink.c.

The rest of the minix directory implements the following tasks:

- bitmap.c manages allocation and freeing of inodes and blocks (the ext2 fs, otherwise, has two different source files);
- fsynk.c is responsible for the `fsync()` system calls--it manages direct, indirect and double indirect blocks (I assume you know about them, it's common Unix knowledge);
- namei.c embeds all the name-related inode operations, such as creating and destroying nodes, renaming and linking;
- truncate.c performs truncation of files.

The console driver

Being the main I/O device on most Linux boxes, the console driver deserves some attention. The source code related to the console, as well as the other character drivers, is to be found in drivers/char, and we'll use this very directory as our referenece point when naming files.

Console initialization is performed by the function `tty_init()`, in tty_io.c. This function is only concerned in getting major device numbers and calling the init function for each device set. `con_init()`, then is the one related to the console, and resides in console.c.

NEW Initialization of the console has changed quite a lot during 1.1 evolution. `console_init()` has been detatched from `tty_init()`, and is called directly by ../../main.c. The virtual consoles are now dynamically allocated, and quite a good deal of code has changed. So, I'll skip the details of initialization, allocation and such.

How file operations are dispatched to the console

This paragraph is quite low-level, and can be happily skipped over.

Needless to say, a Unix device is accessed though the filesystem. This paragraph details all steps from the device file to the actual console functions. Moreover, the following information is extracted from the 1.1.73 source code, and it may be slightly different from the 1.0 source.

When a device inode is opened, the function `chrdev_open()` (or `blkdev_open()`, but we'll stich to character devices) in ../../fs/devices.c gets executed. This function is reached by means of the structure `def_chr_fops`, which in turn is referenced by `chrdev_inode_operations`, used by all the filesystem types (see the previous section about filesystems).

`chrdev_open` takes care of specifying the device operations by substituting the device specific `file_operations` table in the current `filp` and calls the specific `open()`. Device specific tables are kept in the array `chrdevs[]`, indexed by the majour device number, and filled by the same ../../fs/devices.c.

If the device is a tty one (aren't we aiming at the console?), we come to the tty drivers, whose functions are in `tty_io.c`, indexed by `tty_fops`. Thus, `tty_open()` calls `init_dev()`, which allocates any data structure needed by the device, based on the minor device number.

The minor number is also used to retrieve the actual driver for the device, which has been registered through `tty_register_driver()`. The driver, then, is still another structure used to dispatch computation, just like `file_ops`; it is concerned with writing and controlling the device. The last data structure used in managing a tty is the line discipline, described later. The line discipline for the console (and any other tty device) is set by `initialize_tty_struct()`, invoked by `init_dev`.

Everything we touched in this paragraph is device-independent. The only console-specific particular is that console.c, has registered its own driver during `con_init()`. The line discipline, on the contrary, in independent of the device.

MORE The `tty_driver` structure is fully explained within `<linux/tty_driver.h>`.

NEW The above information has been extracted from 1.1.73 source code. It isn't unlikely for your kernel to be somewhat different (``This information is subject to change without notice'').

Writing to the console

When a console device is written to, the function `con_write` gets invoked. This function manages all the control characters and escape sequences used to provide applications with complete screen management. The escape sequences implemented are those of the vt102 terminal; This means that your environment should say `TERM=vt102` when you are telnetting to a non-Linux host; the best choice for local activities, however, is `TERM=console` because the Linux console offers a superset of vt102 functionality.

`con_write()`, thus, is mostly made up of nested switch statements, used to handle a finite state automaton interpreting escape sequences one character at a time. When in normal mode, the character being printed is written directly to the video memory, using the current `attr`-ibute. Within console.c, all the fields of `struct vc` are made accessible through macros, so any reference to (for example) `attr`, does actually refer to the field in the structure `vc_cons[currcons]`, as long as `currcons` is the number of the console being referred to.

NEW Actually, `vc_cons` in newer kernels is no longer an array of structures , it now is an array of pointers whose contents are `kmalloc()`ed. The use of macros greatly simplified changing the approach, because much of the code didn't need to be rewritten.

Actual mapping and unmapping of the console memory to screen is performed by the functions `set_scrmem()` (which copies data from the console buffer to video memory) and `get_scrmem` (which copies back data to the console buffer). The private buffer of the current console is physically mapped on the actual video RAM, in order to minimize the number of data transfers. This means that `get-` and `set-_scrmem()` are `static` to console.c and are called only during a console switch.

Reading the console

Reading the console is accomplished through the line-discipline. The default (and unique) line discipline in Linux is called `tty_ldisc_N_TTY`. The line discipline is what ``disciplines input through a line''. It is another function table (we're used to the approach, aren't we?), which is concerned with reading the device. With the help of `termios` flags, the line discipline is what controls input from the tty: raw, cbreak and cooked mode; `select()`; `ioctl()` and so on.

The read function in the line discipline is called `read_chan()`, which reads the tty buffer independently of whence it came from. The reason is that character arrival through a tty is managed by asynchronous hardware interrupts.

MORE The line discipline `N_TTY` is to be found in the same tty_io.c, though later kernels use a different n_tty.c source file.

The lowest level of console input is part of keyboard management, and thus it is handled within keyboard.c, in the function `keyboard_interrupt()`.

Keyboard management

Keyboard management is quite a nightmare. It is confined to the file keyboard.c, which is full of hexadecimal numbers to represent the various keycodes appearing in keyboards of different manifacturers.

I won't dig in keyboard.c, because no relevant information is there to the kernel hacker.

MORE For those readers who are really interested in the Linux keyboard, the best approach to keyboard.c is from the last line upward. Lowest level details occur mainly in the first half of the file.

Switching the current console

The current console is switched through invocation of the function `change_console()`, which resides in tty_io.c and is invoked by both keyboard.c and vt.c (the former switches console in response to keypresses, the latter when a program requests it by invoking an `ioctl()` call).

The actual switching process is performed in two steps, and the function `complete_change_console()` takes care of the second part of it. Splitting the switch is meant to complete the task after a possible handshake with the process controlling the tty we're leaving. If the console is not under process control, `change_console()` calls `complete_change_console()` by itself. Process intervertion is needed to successfully switch from a graphic console to a text one and viceversa, and the X server (for example) is the controlling process of its own graphic console.

The selection mechanism

``selection'' is the cut and paste facility for the Linux text consoles. The mechanism is mainly handled by a user-level process, which can be instantiated by either selection or gpm. The user-level program uses `ioctl()` on the console to tell the kernel to highlight a region of the screen. The selected text, then, is copied to a selection buffer. The buffer is a static entity in console.c. Pasting text is accomplished by `manually' pushing characters in the tty input queue. The whole selection mechanism is protected by `#ifdef` so users can disable it during kernel configuration to save a few kilobytes of ram.

Selection is a very-low-level facility, and its workings are hidden from any other kernel activity. This means that most `#ifdef`'s simply deals with removing the highlight before the screen is modified in any way.

Newer kernels feature improved code for selection, and the mouse pointer can be highlighted independently of the selected text (1.1.32 and later). Moreover, from 1.1.73 onward a dynamic buffer is used for selected text rather than a static one, making the kernel 4kB smaller.

`ioctl()`ling the device

The `ioctl()` system call is the entry point for user processes to control the behaviour of device files. Ioctl management is spawned by ../../fs/ioctl.c, where the real `sys_ioctl()` resides. The standard `ioctl` requests are performed right there, other file-related requests are processed by `file_ioctl()` (same source file), while any other request is dispatches to the device-specific `ioctl()` function.

The `ioctl` material for console devices resides in vt.c, because the console driver dispatches ioctl requests to `vt_ioctl()`.

The information above refer to 1.1.7x. The 1.0 kernel doesn't have the ``driver'' table, and `vt_ioctl()` is pointed to directly by the `file_operations()` table.

Ioctl material is quite confused, indeed. Some requests are related to the device, and some are related to the line discipline. I'll try to summarize things for the 1.0 and the 1.1.7x kernels. Anything happened in between.

The 1.1.7x series features the following approach: tty_ioctl.c implements only line discipline requests (namely `n_tty_ioctl()`, which is the only n_tty function outside of n_tty.c), while the `file_operations` field points to `tty_ioctl()` in tty_io.c. If the request number is not resolved by `tty_ioctl()`, it is passed along to `tty->driver.ioctl` or, if it fails, to `tty->ldisc.ioctl`. Driver-related stuff for the console it to be found in vt.c, while line discipline material is in tty_ioctl.c.

In the 1.0 kernel, `tty_ioctl()` is in tty_ioctl.c and is pointed to by generic tty `file_operations`. Unresolved requests are passed along to the specific ioctl function or to the line-discipline code, in a way similar to 1.1.7x.

Note that in both cases, the `TIOCLINUX` request is in the device-independent code. This implies that the console selection can be set by `ioctl`ling any tty (`set_selection()` always operates on the foreground console), and this is a security hole. It is also a good reason to switch to a newer kernel, where the problem is fixed by only allowing the superuser to handle the selection.

A variety of requests can be issued to the console device, and the best way to know about them is to browse the source file vt.c.

Messages

8. access a file from module *by proy018@avellano.usal.es*
7. Which head.S? *by Johnie Stafford*
    1. Untitled *by benschop@eb.ele.tue.nl*
6. STREAMS and Linux *by Venkatesha Murthy G.*
    1. Re: STREAMS and LINUX *by Vineet Sharma*
5. Do you still need to run update ? *by Chris Ebenezer*
4. Do you still need to run bdflush? *by Steve Dunham*
    1. Already answered... *by Michael K. Johnson*
3. Kernel Configuration and Makefile Structure *by Steffen Moeller*
    1. Editing services available... *by Michael K. Johnson*
    2. Kernel configuration *by Venkatesha Murthy G.*
2. Re: Kernel threads *by Paul Gortmaker*
    1. More on usage of kernel threads. *by David S. Miller*
1. kernel startup code *by Alan Cox*
    1. Untitled *by Karapetyants Vladimir Vladimirovitch*

access a file from module

Forum: Tour of the Linux kernel source
Date: Thu, 08 May 1997 12:06:47 GMT
From: <proy018@avellano.usal.es>

I need to access a file from a module

Which head.S?

Forum: Tour of the Linux kernel source

Keywords: head.S
Date: Sat, 20 Jul 1996 00:57:09 GMT

From: Johnie Stafford <jms@pobox.com>

```
In the "Tour of the Linux kernel source" section there is
reference to boot/head.S. I did a find on the source, this
is a list of the "head.S"'s in the source:

./arch/i386/boot/compressed/head.S
./arch/i386/kernel/head.S
./arch/alpha/boot/head.S
./arch/alpha/kernel/head.S
./arch/sparc/kernel/head.S
./arch/mips/kernel/head.S
./arch/ppc/kernel/head.S
./arch/ppc/boot/compressed/head.S
./arch/m68k/kernel/head.S

Obviously, there is a different one for each architecture.
But which version for the i386 architecture is being
refered to here, and what's the difference?

        Johnie
```

Messages

1. Untitled *by benschop@eb.ele.tue.nl*

Untitled

Forum: Tour of the Linux kernel source
Re: Which head.S? (Johnie Stafford)
Keywords: head.S
Date: Tue, 23 Jul 1996 07:38:08 GMT
From: <benschop@eb.ele.tue.nl>

The file arch/i386/kernel/head.S is linked with the uncompressed kernel. If the kernel is not compressed this is the only head.S used. In a compressed kernel, all 32 bit objects from the kernel, including the above mentioned head.o are compressed and the compressed data is lumped together in the file piggy.o. Now the file arch/i386/boot/compressed/head.S comes into play. This and the decompressor and piggy.o form a new 32-bit object.

STREAMS and Linux

Forum: Tour of the Linux kernel source
Keywords: STREAMS devices drivers
Date: Mon, 15 Jul 1996 12:01:50 GMT
From: Venkatesha Murthy G. <gvmt@csa.iisc.ernet.in>

Hi all,

Correct me if i am wrong, but Linux doesn't have any STREAMS devices or drivers as of now. But as Ritchie's paper explains, they are flexible and can find use in a lot of places where piplelined processing is involved - net drivers for instance. Anything being done/planned in that direction?

Venkatesha Murthy (gvmt@csa.iisc.ernet.in)

Messages

1. Re: STREAMS and LINUX *by Vineet Sharma*

Re: STREAMS and LINUX

Forum: Tour of the Linux kernel source
Re: STREAMS and Linux (Venkatesha Murthy G.)
Keywords: STREAMS devices drivers
Date: Thu, 10 Apr 1997 15:49:04 GMT
From: Vineet Sharma <vsharma@hss.hns.com>

```
Go to ftp.gcom.com/pub/linux/src/ and pick up the Lis-2.0.25.tar.gz package.
```

Do you still need to run update ?

Forum: Tour of the Linux kernel source
Date: Tue, 25 Jun 1996 13:59:41 GMT
From: Chris Ebenezer <ncfee@wmin.ac.uk>

The docs on this daemon state that it is one of a pair of two daemons - bdflush/update - which manage disk buffers. In the latest (2.0.x) kernels starting up update does not have the effect of also starting up bdflush. So is update still needed ?

Do you still need to run bdflush?

Forum: Tour of the Linux kernel source
Date: Mon, 27 May 1996 18:55:44 GMT
From: Steve Dunham <dunham@gdl.msu.edu>

The recent 1.3.x kernels add a kernel thread named (kflushd) What does this do? Does it replace the functionality of the user program 'bdflush'?

Messages

1. Already answered... *by Michael K. Johnson*

Already answered...

Forum: Tour of the Linux kernel source
Re: Do you still need to run bdflush? (Steve Dunham)
Keywords: kflushd, searching
Date: Mon, 27 May 1996 19:42:00 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

It looks like I'll eventually have to add search capability to the KHG. It will be a while before I have time, though.

Your questions is already answered in a response elsewhere in this document; see Kernel threads, posted by Paul Gortmaker.

Re: Kernel threads

Forum: Tour of the Linux kernel source
Keywords: kernel threads
Date: Wed, 22 May 1996 16:51:58 GMT
From: Paul Gortmaker <gpg109@rsphy1.anu.edu.au>

The above mentions that v1.0 has "early" support for threads, which can use a bit of an update. They are fully functional and in use in late v1.3.x kernels. For example the internal bdflush daemon used to be started by a non-returning syscall in all the v1.2.x kernels, but as of around v1.3.4x or so, I made it into an internal thread, and dispensed with the reliance on the user space syscall to launch the thing. This is now what is seen as "kflushd" or process #2 on all recent kernels. Since then, other threads such as "kflushd" and multiple "nfsiod" processes have taken advantage of the same functionality.

Paul.

Messages

1. More on usage of kernel threads. *by David S. Miller*

More on usage of kernel threads.

Forum: Tour of the Linux kernel source
Re: Re: Kernel threads (Paul Gortmaker)
Keywords: kernel threads asynchronous faults
Date: Fri, 24 May 1996 05:33:03 GMT
From: David S. Miller <davem@caip.rutgers.edu>

```
        As an another addendum the AP+ multicomputer port
(actually it is a part of the generic Sparc kernel sources)
uses kernel threads to solve the problem of servicing a
true fault from interrupt space.  The kernel thread is called
asyncd(), the AP+ multicomputer takes interrupts when one
cell on the machine does a dma access to another cell and the
page is not present or otherwise needs to be faulted in or
whatever.  The interrupt handler adds this fault to a queue
of faults to service and wakes up the async daemon which runs
with real time priority much like the other linux kernel
daemons.  Poof, solution to the classic interrupt context
limitation problem. ;-)
```

Later, David S. Miller (davem@caip.rutgers.edu)

Kernel Configuration and Makefile Structure

Forum: Tour of the Linux kernel source
Keywords: configuration makefile
Date: Wed, 22 May 1996 17:34:39 GMT
From: Steffen Moeller <steffen@di.unito.it>

I'm missing a description of the Makefile mechanism and the principle of the configuration. Or is this too trivial for a Hacker's Guide? I do not think so since

- it's a nice introduction,
- all hackers have to understand it and
- it's a good place to put hyperlinks to the real stuff in this guide.

If there's some positive feedback I'd like to start on this myself, but I'd need some help - at least for the language.

Steffen

Messages

1. Editing services available... *by Michael K. Johnson*
2. Kernel configuration *by Venkatesha Murthy G.*

Editing services available...

Forum: Tour of the Linux kernel source
Re: Kernel Configuration and Makefile Structure (Steffen Moeller)
Keywords: configuration makefile
Date: Thu, 23 May 1996 17:00:42 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

This is certainly not too trivial a topic for the KHG. If you are willing to tackle it, feel free. If someone else wants to work on it, that's fine too.

If by "...but I'd need some help - at least for the language" you mean that you would like someone to edit your piece, you can send it to me for editing. If I feel that it needs more work before being added, I'll send it back for revision, hopefully with helpful comments... :-)

Kernel configuration

Forum: Tour of the Linux kernel source
Re: Kernel Configuration and Makefile Structure (Steffen Moeller)
Keywords: configuration
Date: Thu, 11 Jul 1996 12:30:00 GMT
From: Venkatesha Murthy G. <gvmt@csa.iisc.ernet.in>

I really haven't *understood* kernel configutarion but i can tell you what i do when i want to add a config option. I first edit arch/i386/config.in and add a line that looks like

bool 'whatever explanation' CONFIG_WHATEVER default

this is supposed to mean that CONFIG_WHATEVER is a boolean taking values y or n. When you 'make config' you'll get something like

'whatever explanation (CONFIG_WHATEVER) [default]'

and you type in y or n. Now this automagically #defines CONFIG_WHATEVER in <linux/autoconf.h>. Code that is specefic to the configuration can now be enclosed in #ifdef CONFIG_WHATEVER ... #endif so it will be compiled in only when configured. If you want any more explanation than can be given on one line, you can have a set of 'comment ...." lines before the 'bool ....' line and that will be displayed for you during configuration.

I don't know if you'll find it useful but still .....

Venkatesha Murthy (gvmt@csa.iisc.ernet.in)

kernel startup code

Forum: Tour of the Linux kernel source
Keywords: SMP start_kernel()
Date: Fri, 17 May 1996 10:48:00 GMT
From: Alan Cox <*unknown*>

The intel startup code and start_kernel() is partly used for SMP startup as the intel MP design starts the secondary CPU's in real mode. In addition to make it more fun you can only pass one piece of information - the address (page boundary) that the processor is made to boot at. The SMP kernel writes a trampoline routine at the base of a page it allocates for the stack of each CPU. The secondary processors (or AP's as Intel calls them for Application Processors) load their SS:SP based on the code segment enter protected mode and jump into the 32bit kernel startup.

The kernel startup for the SMP kernel in start_kernel() calls a few startup routines for the architecture and then waits for the boot processor to complete initialisation. At this point it starts running an idle thread and is schedulable.

Messages

1. Untitled *by Karapetyants Vladimir Vladimirovitch*

Broken URL: http://www.redhat.com:8080/HyperNews/get/tour/tour/1/1.html

Try: http://www.redhat.com:8080/HyperNews/get/tour/tour/1.html

Device Drivers

If you choose to write a device driver, you must take everything written here as a guide, and no more. I cannot guarantee that this chapter will be free of errors, and I cannot guarantee that you will not damage your computer, even if you follow these instructions exactly. It is highly unlikely that you will damage it, but I cannot guarantee against it. There is only one ``infallible'' direction I can give you: **Back up!** Back up before you test your new device driver, or you may regret it later.

Other sources of information

Quite a few other references are also available on the topic of writing Linux device drivers by now. I put up some (slightly outdated by now, but still worth reading, I think) notes for a talk I gave in May 1995 entitled Writing Linux Device Drivers, which is specifically oriented at character devices implemented as kernel runtime-loadable modules.

Linux Journal has had a long-running series of articles called **Kernel Korner** which, despite the wacky name, has had quite a bit of useful information on it. Some of the articles from that column may be available on the web; most of them are available for purchase as back issues. One particularly useful series of articles, which focussed in far more detail than my 30 minute talk on the subject of kernel runtime-loadable modules, was in issues 23, 24, 25, 26, and 28. They were written by Alessandro Rubini and Georg v. Zezschwitz. Issue 29 is slated (as of this writing) to have an article on writing network device drivers, written by Alan Cox. Issues 9, 10, and 11 have a series that I wrote on block device drivers.

Messages

22. DMA to user space *by Marcel Boosten* NEW
21. How a device driver can driver his device *by Kim yeonseop* NEW
    1. Untitled NEW
20. memcpy error? *by Edgar Vonk*
19. Unable to handle kernel paging request - error *by Edgar Vonk*
17. _syscallX() Macros *by Tom Howley*
16. MediaMagic Sound Card DSP-16. How to run in Linux. *by Robert Hinson*
15. What does mark_bh() do? *by Erik Petersen*
    1. Untitled *by Praveen Dwivedi*
14. 3D Acceleration *by jamesbat@innotts.co.uk*
13. Device Drivers: /dev/radio... *by Matthew Kirkwood*
12. Does anybody know why kernel wakes my driver up without apparant reasons? *by David van Leeuwen*
11. Getting a DMA buffer aligned with 64k boundaries *by Juan de La Figuera Bayon*
10. Hardware Interface I/O Access *by Terry Moore*
    1. You are somewhat confused... *by Michael K. Johnson*
9. Is Anybody know something about SIS 496 IDE chipset? *by Alexander*
7. Vertical Retrace Interrupt - I need to use it *by Brynn Rogers*
    1. Your choice... *by Michael K. Johnson*
6. help working with skb structures *by arkane*
5. Interrupt Sharing ? *by Frieder Löffler*

   1. Interrupt sharing-possible *by Vladimir Myslik*
   -> Interrupt sharing - How to do with Network Drivers? *by Frieder Löffler*
   -> Interrupt sharing 101 *by Christophe Beauregard*
4. Device Driver notification of "Linux going down" *by Stan Troeh*
   1. Through application which has opened the device *by Michael K. Johnson*
   2. Device Driver notification of "Linux going down" *by Marko Kohtala*
3. Is waitv honored? *by Michael K. Johnson*
2. PCI Driver *by Flavia Donno*
   1. There is linux-2.0/drivers/pci/pci.c *by Hasdi*
1. Re: Network Device Drivers *by Paul Gortmaker*
   1. Re: Network Device Drivers *by Neal Tucker*
      1. network driver info *by Neal Tucker*
      -> Network Driver Desprately Needed *by Paul Atkinson*
   2. Transmit function *by Joerg Schorr*
      1. Re: Transmit function *by Paul Gortmaker*
      -> Skbuff *by Joerg Schorr*

What is a Device Driver?

Making hardware work is tedious. To write to a hard disk, for example, requires that you write magic numbers in magic places, wait for the hard drive to say that it is ready to receive data, and then feed it the data it wants, very carefully. To write to a floppy disk is even harder, and requires that the program supervise the floppy disk drive almost constantly while it is running.

Instead of putting code in each application you write to control each device, you share the code between applications. To make sure that that code is not compromised, you protect it from users and normal programs that use it. If you do it right, you will be able to add and remove devices from your system without changing your applications at all. Furthermore, you need to be able to load your program into memory and run it, which the operating system also does. So an operating system is essentially a priviledged, general, sharable library of low-level hardware and memory and process control functions and routines.

All versions of Unix have an abstract way of reading and writing devices. By making the devices act as much as possible like regular files, the same calls (`read()`, `write()`, etc.) can be used for devices and files. Within the kernel, there are a set of functions, registered with the filesystem, which are called to handle requests to do I/O on ``device special files,'' which are those which represent devices. (See `mknod(1,2)` for an explanation of how to make these files.)

All devices controlled by the same device driver are given the same **major number**, and of those with the same major number, different devices are distinguished by different **minor numbers**. (This is not strictly true, but it is close enough. If you understand where it is not true, you don't need to read this section, and if you don't but want to learn, read the code for the tty devices, which uses up 2 major numbers, and may use a third and possibly fourth by the time you read this. Also, the ``misc'' major device supports many minor devices that only need a few minor numbers; we'll get to that later.)

This chapter explains how to write any type of Linux device driver that you might need to, including character, block, SCSI, and network drivers. It explains what functions you need to write, how to initialize your drivers and obtain memory for them efficiently, and what function are built in to Linux to make your job easier.

Creating device drivers for Linux is easier than you might think. It merely involves writing a few functions and registering them with the Virtual Filesystem Switch (VFS), so that when the proper device special files are accessed, the VFS can call your functions.

However, a word of warning is due here: Writing a device driver **is** writing a part of the Linux kernel. This means that your driver runs with kernel permissions, and can do anything it wants to: write to any memory, reformat your hard drive, damage your monitor or video card, or even break your dishes, if your dishwasher is controlled by your computer. Be careful.

Also, your driver will run in kernel mode, and the Linux kernel, like most Unix kernels, is non-pre-emptible. This means that if you driver takes a long time to work without giving other programs a chance to work, your computer will appear to ``freeze'' when your driver is running. Normal user-mode pre-emptive scheduling does not apply to your driver.

Messages

1. Question ? *by Rose Merone*
-> Not yet... *by Michael K. Johnson*

Question ?

Forum: What is a Device Driver?
Date: Mon, 24 Mar 1997 08:39:09 GMT
From: Rose Merone <*unknown*>

D'ya have a book that covers all about device driver management in Linux ?

Messages

1. Not yet... *by Michael K. Johnson*

Not yet...

Forum: What is a Device Driver?
Re: Question ? (Rose Merone)
Date: Mon, 21 Apr 1997 14:00:19 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Alessandro Rubini is writing a book about writing device drivers for O'Reilly. See http://www.ora.com/catalog/linuxdrive/ and http://www.ora.com/catalog/linuxdrive/desc.html

User-space device drivers

It is not always necessary to write a device driver for a device, especially in applications where no two applications will compete for the device. The most useful example of this is a memory-mapped device, but you can also do this with devices in I/O space (devices accessed with `inb()` and `outb()`, etc.). If your process is running as superuser (root), you can use the `mmap()` call to map some of your process memory to actual memory locations, by `mmap()`'ing a section of /dev/mem. When you have done this mapping, it is pretty easy to write and read from real memory addresses just as you would read and write any variables.

If your driver needs to respond to interrupts, then you really need to be working in kernel space, and need to write a real device driver, as there is no good way at this time to deliver interrupts to user processes. Although the DOSEMU project has created something called the SIG (Silly Interrupt Generator) which allows interrupts to be posted to user processes (I believe through the use of signals), the SIG is not particularly fast, and should be thought of as a last resort for things like DOSEMU.

An interrupt is an asyncronous notification posted by the hardware to alert the device driver of some condition. You have likely dealt with `IRQ's when setting up your hardware; an IRQ is an ``Interrupt ReQuest line,'' which is triggered when the device wants to talk to the driver. This may be because it has data to give to the drive, or because it is now ready to receive data, or because of some other ``exceptional condition'' that the driver needs to know about. It is similar to user-level processes receiving a signal, so similar that the same `sigaction` structure is used in the kernel to deal with interrupts as is used in user-level programs to deal with signals. Where the user-level has its signals delivered to it by the kernel, the kernel has interrupt delivered to it by hardware.

If your driver must be accessible to multiple processes at once, and/or manage contention for a resource, then you also need to write a real device driver at the kernel level, and a user-space device driver will not be sufficient or even possible.

Example: `vgalib`

A good example of a user-space driver is the `vgalib` library. The standard `read()` and `write()` calls are really inadequate for writing a really fast graphics driver, and so instead there is a library which acts conceptually like a device driver, but runs in user space. Any processes which use it **must** run setuid root, because it uses the `ioperm()` system call. It is possible for a process that is not setuid root to write to /dev/mem if you have a group `mem` or `kmem` which is allowed write permission to /dev/mem and the process is properly setgid, but only a process running as root can execute the `ioperm()` call.

There are several I/O ports associated with VGA graphics. **vgalib** creates symbolic names for this with `#define` statements, and then issues the `ioperm()` call like this to make it possible for the process to read and write directly from and to those ports:

```
    if (ioperm(CRT_IC, 1, 1)) {
        printf("VGAlib: can't get I/O permissions \n");
        exit (-1);
    }
    ioperm(CRT_IM,  1, 1);
    ioperm(ATT_IW, 1, 1);
[...]
```

It only needs to do error checking once, because the only reason for the `ioperm()` call to fail is that it is not being called by the superuser, and this status is not going to change.

⚠After making this call, the process is allowed to use `inb` and `outb` machine instructions, but only on the specified ports. These instructions can be accessed without writing directly in assembly by including , but will only work if you compile **with optimization on,** by giving the `-O?` to gcc. Read `<linux/asm.h>` for details.

After arranging for port I/O, `vgalib` arranges for writing directly to kernel memory with the following code:

```
    /* open /dev/mem */
    if ((mem_fd = open("/dev/mem", O_RDWR) ) < 0) {
        printf("VGAlib: can't open /dev/mem \n");
        exit (-1);
    }

    /* mmap graphics memory */
    if ((graph_mem = malloc(GRAPH_SIZE + (PAGE_SIZE-1))) == NULL) {
        printf("VGAlib: allocation error \n");
        exit (-1);
    }
    if ((unsigned long)graph_mem % PAGE_SIZE)
        graph_mem += PAGE_SIZE - ((unsigned long)graph_mem % PAGE_SIZE);
    graph_mem = (unsigned char *)mmap(
        (caddr_t)graph_mem,
        GRAPH_SIZE,
        PROT_READ|PROT_WRITE,
        MAP_SHARED|MAP_FIXED,
        mem_fd,
        GRAPH_BASE
    );
    if ((long)graph_mem < 0) {
        printf("VGAlib: mmap error \n");
        exit (-1);
    }
```

It first opens /dev/mem, then allocates memory enough so that the mapping can be done on a page (4 KB) boundary, and then attempts the map. `GRAPH_SIZE` is the size of VGA memory, and `GRAPH_BASE` is the first address of VGA memory in /dev/mem. Then by writing to the address that is returned by `mmap()`, the process is actually writing to screen memory.

Example: mouse conversion

If you want a driver that acts a bit more like a kernel-level driver, but does not live in kernel space, you can also make a fifo, or named pipe. This usually lives in the /dev/ directory (although it doesn't need to) and acts substantially like a device once set up. However, fifo's are one-directional only--they have one reader and one writer.

For instance, it used to be that if you had a PS/2-style mouse, and wanted to run XFree86, you had to create a fifo called /dev/mouse, and run a program called mconv which read PS/2 mouse ``droppings'' from /dev/psaux, and wrote the equivalent microsoft-style ``droppings'' to /dev/mouse. Then XFree86 would read the ``droppings'' from /dev/mouse, and it would be as if there were a microsoft mouse connected to /dev/mouse. Even though XFree86 is now able to read PS/2 style ``droppings'', the concepts in this example still stand. (If you have a better example, I'd be glad to see it.)

The evil instruction

Don't use the `cli()` instruction. It's possible to use it as root to disable interrupts, and one particular program used to used to use it--the **clock** program. However, this kills SMP machines. If you need to use `cli()`, you need a kernel-space driver, and a user-space driver will only cause grief as more and more Linux users use SMP machines.

Messages

1. What is SMP?
-> SMP: Two Definitions? *by Reinhold J. Gerharz*
-> Only one definition for Linux... *by Michael K. Johnson*

What is SMP?

Forum: User-space device drivers
Keywords: SMP
Date: Mon, 16 Dec 1996 00:22:27 GMT
From: <unknown>

```
It might not be appropriate to ask, but it'd be real nice to
know what SMP means.  I never saw cli() instruction do any
harm to any Linux machine I've met.
```

Messages

1. SMP: Two Definitions? *by Reinhold J. Gerharz*
-> Only one definition for Linux... *by Michael K. Johnson*

SMP: Two Definitions?

Forum: User-space device drivers
Re: What is SMP?
Keywords: SMP
Date: Thu, 09 Jan 1997 03:18:21 GMT
From: Reinhold J. Gerharz <rgerharz@erols.com>

I thought SMP meant "symetric multi-processing," a technology where two or more processors share equal access to memory, device I/O, and interrupts. Ideally one would expect a 100 percent improvement in processing performance for each additional processor, but in reality only 80-90 percent is achieved.

However, I have discovered that to some people, SMP means "shared-memory multi-processing." This technology allows multiple processors to run user programs, but one processor reserves interrupt and I/O handling for itself. This is traditionally called "asymetric multi-processing," and I have tentatively concluded that only "marketing types" would use this terminology to confuse potential customers.

Messages

1. Only one definition for Linux... *by Michael K. Johnson*

Only one definition for Linux...

Forum: User-space device drivers
Re: What is SMP?
Re: SMP: Two Definitions? (Reinhold J. Gerharz)
Keywords: SMP
Date: Mon, 13 Jan 1997 14:26:44 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

In the Linux world, SMP really does mean symmetric multi-processing. Currently, there's a lock around the whole kernel so that only one CPU can be in kernel mode at once, but all the CPUs can run in kernel mode at different times.

As you add more CPU's to an SMP system, the amount of extra performance you get out of each additional CPU decreases, until at some point it actually decreases performance to add another CPU. Most systems simply don't support enough CPUs to get a negative marginal performance gain, so that usually isn't an issue.

Also, because Linux uses a single lock, the current kernels degrade more quickly as you add more CPUs than a multiple-lock system would for I/O-bound tasks. CPU-bound tasks, on the other hand, work very well with a single lock around the kernel.

Device Driver Basics

We will assume that you decide that you do not wish to write a user-space device, and would rather implement your device in the kernel. You will probably be writing writing two files, a `.c` file and a `.h` file, and possibly modifying other files as well, as will be described below. We will refer to your files as foo.c and foo.h, and your driver will be the `foo` driver.

Namespace

One of the first things you will need to do, before writing any code, is to name your device. This name should be a short (probably two or three character) string. For instance, the parallel device is the ``` ``lp ```" device, the floppies are the ``` ``fd ```" devices, and SCSI disks are the ``` ``sd ```" devices. As you write your driver, you will give your functions names prefixed with your chosen string to avoid any namespace confusion. We will call your prefix `foo,` and give your functions names like `foo_read(), foo_write(),` etc.

Allocating memory

Memory allocation in the kernel is a little different from memory allocation in normal user-level programs. Instead of having a `malloc()` capable of delivering almost unlimited amounts of memory, there is a `kmalloc()` function that is a bit different:

- Memory is provided in pieces whose size is a power of 2, except that pieces larger than 128 bytes are allocated in blocks whose size is a power of 2 minus some small amount for overhead. You can request any odd size, but memory will not be used any more efficiently if you request a 31-byte piece than it will if you request a 32 byte piece. Also, there is a limit to the amount of memory that can be allocated, which is currently 131056 bytes.
- `kmalloc()` takes a second argument, the priority. This is used as an argument to the `get_free_page()` function, where it is used to determine when to return. The usual priority is `GFP_KERNEL`. If it may be called from within an interrupt, use `GFP_ATOMIC` and be truly prepared for it to fail (don't panic). This is because if you specify `GFP_KERNEL`, `kmalloc()` may sleep, which cannot be done on an interrupt. The other option is `GFP_BUFFER`, which is used only when the kernel is allocating buffer space, and never in device drivers.

To free memory allocated with `kmalloc()`, use one of two functions: `kfree()` or `kfree_s()`. These differ from `free()` in a few ways as well:

- `kfree()` is a macro which calls `kfree_s()` and acts like the standard `free()` outside the kernel.
- If you know what size object you are freeing, you can speed things up by calling `kfree_s()` directly. It takes two arguments: the first is the pointer that you are freeing, as in the single argument to `kfree()`, and the second is the size of the object being freed.

See Supporting Functions for more information on `kmalloc()`, `kfree()`, and other useful functions.

Be gentle when you use kmalloc. Use only what you have to. Remember that kernel memory is unswappable, and thus allocating extra memory in the kernel is a far worse thing to do in the kernel than in a user-level program. Take only what you need, and free it when you are done, unless you are going to use it right away again.

Character vs. block devices

There are two main types of devices under all Unix systems, character and block devices. Character devices are those for which no buffering is performed, and block devices are those which are accessed through a cache. Block devices must be random access, but character devices are not required to be, though some are. Filesystems can only be mounted if they are on block devices.

Character devices are read from and written to with two function: `foo_read()` and `foo_write()`. The `read()` and `write()` calls do not return until the operation is complete. By contrast, block devices do not even implement the `read()` and `write()` functions, and instead have a function which has historically been called the ``strategy routine.'' Reads and writes are done through the buffer cache mechanism by the generic functions `bread()`, `breada()`, and `bwrite()`. These functions go through the buffer cache, and so may or may not actually call the strategy routine, depending on whether or not the block requested is in the buffer cache (for reads) or on whether or not the buffer cache is full (for writes). A request may be asyncronous: `breada()` can request the strategy routine to schedule reads that have not been asked for, and to do it asyncronously, in the background, in the hopes that they will be needed later.

The sources for character devices are kept in drivers/char/, and the sources for block devices are kept in drivers/block/. They have similar interfaces, and are very much alike, except for reading and writing. Because of the difference in reading and writing, initialization is different, as block devices have to register a strategy routine, which is registered in a different way than the `foo_read()` and `foo_write()` routines of a character device driver. Specifics are dealt with in Character Device Initialization and Block Device Initialization.

Interrupts vs. Polling

Hardware is slow. That is, in the time it takes to get information from your average device, the CPU could be off doing something far more useful than waiting for a busy but slow device. So to keep from having to **busy-wait** all the time, **interrupts** are provided which can interrupt whatever is happening so that the operating system can do some task and return to what it was doing without losing information. In an ideal world, all devices would probably work by using interrupts. However, on a PC or clone, there are only a few interrupts available for use by your peripherals, so some drivers have to poll the hardware: ask the hardware if it is ready to transfer data yet. This unfortunately wastes time, but it sometimes needs to be done.

Some hardware (like memory-mapped displays) is as fast as the rest of the machine, and does not generate output asyncronously, so an interrupt-driven driver would be rather silly, even if interrupts were provided.

In Linux, many of the drivers are interrupt-driven, but some are not, and at least one can be either, and can be switched back and forth at runtime. For instance, the `lp` device (the parallel port driver) normally polls the printer to see if the printer is ready to accept output, and if the printer stays in a not ready phase for too long, the driver will sleep for a while, and try again later. This improves system performance. However, if you have a parallel card that supplies an interrupt, the driver will utilize that, which will usually make performance even better.

There are some important programming differences between interrupt-driven drivers and polling drivers. To understand this difference, you have to understand a little bit of how system calls work under Unix. The kernel is not a separate task under Unix. Rather, it is as if each process has a copy of the kernel. When a process executes a system call, it does not transfer control to another process, but rather, the process changes execution modes, and is said to be ``in kernel mode.'' In this mode, it executes kernel code which is trusted to be safe.

In kernel mode, the process can still access the user-space memory that it was previously executing in, which is done through a set of macros: `get_fs_*()` and `memcpy_fromfs()` read user-space memory, and `put_fs_*()` and `memcpy_tofs()` write to user-space memory. Because the process is still running, but in a different mode, there is no question of where in memory to put the data, or where to get it from. However, when an interrupt occurs, any process might currently be running, so these macros cannot be used--if they are, they will either write over random memory space of the running process or cause the kernel to panic.

Instead, when scheduling the interrupt, a driver must also provide temporary space in which to put the information, and then sleep. When the interrupt-driven part of the driver has filled up that temporary space, it wakes up the process, which copies the information from that temporary space into the process' user space and returns. In a block device driver, this temporary space is automatically provided by the buffer cache mechanism, but in a character device driver, the driver is responsible for allocating it itself.

The sleep-wakeup mechanism

[Begin by giving a general description of how sleeping is used and what it does. This should mention things like all processes sleeping on an event are woken at once, and then they contend for the event again, etc...]

Perhaps the best way to try to understand the Linux sleep-wakeup mechanism is to read the source for the `__sleep_on()` function, used to implement both the `sleep_on()` and `interruptible_sleep_on()` calls.

```
static inline void __sleep_on(struct wait_queue **p, int state)
{
    unsigned long flags;
    struct wait_queue wait = { current, NULL };

    if (!p)
        return;
    if (current == task[0])
        panic("task[0] trying to sleep");
    current->state = state;
    add_wait_queue(p, &wait);
    save_flags(flags);
    sti();
    schedule();
    remove_wait_queue(p, &wait);
    restore_flags(flags);
}
```

A `wait_queue` is a circular list of pointers to task structures, defined in `<linux/wait.h>` to be

```
struct wait_queue {
    struct task_struct * task;
    struct wait_queue * next;
};
```

`state` is either `TASK_INTERRUPTIBLE` or `TASK_UNINTERUPTIBLE`, depending on whether or not the sleep should be interruptable by such things as system calls. In general, the sleep should be interruptible if the device is a slow one; one which can block indefinitely, including terminals and network devices or pseudodevices.

`add_wait_queue()` turns off interrupts, if they were enabled, and adds the new `struct wait_queue` declared at the beginning of the function to the list `p`. It then recovers the original interrupt state (enabled or disabled), and returns.

`save_flags()` is a macro which saves the process flags in its argument. This is done to preserve the previous state of the interrupt enable flag. This way, the `restore_flags()` later can restore the interrupt state, whether it was enabled or disabled. `sti()` then allows interrupts to occur, and `schedule()` finds a new process to run, and switches to it. Schedule will not choose this process to run again until the state is changed to `TASK_RUNNING` by `wake_up()` called on the same wait queue, `p`, or conceivably by something else.

The process then removes itself from the `wait_queue`, restores the orginal interrupt condition with `restore_flags()`, and returns.

Whenever contention for a resource might occur, there needs to be a pointer to a `wait_queue` associated with that resource. Then, whenever contention does occur, each process that finds itself locked out of access to the resource sleeps on that resource's `wait_queue`. When any process is finished using a resource for which there is a `wait_queue`, it should wake up and processes that might be sleeping on that `wait_queue`, probably by calling `wake_up()`, or possibly `wake_up_interruptible()`.

If you don't understand why a process might want to sleep, or want more details on when and how to structure this sleeping, I urge you to buy one of the operating systems textbooks listed in the Annotated Bibliography and look up **mutual exclusion** and **deadlock**.

More advanced sleeping

If the `sleep_on()`/`wake_up()` mechanism in Linux does not satisfy your device driver needs, you can code your own versions of `sleep_on()` and `wake_up()` that fit your needs. For an example of this, look at the serial device driver (drivers/char/serial.c) in function `block_til_ready()`, where quite a bit has to be done between the `add_wait_queue()` and the `schedule()`.

The VFS

The Virtual Filesystem Switch, or **VFS**, is the mechanism which allows Linux to mount many different filesystems at the same time. In the first versions of Linux, all filesystem access went straight into routines which understood the `minix` filesystem. To make it possible for other filesystems to be written, filesystem calls had to pass through a layer of indirection which would switch the call to the routine for the correct filesystem. This was done by some generic code which can handle generic cases and a structure of pointers to functions which handle specific cases. One structure is of interest to the device driver writer; the `file_operations` structure.

From /usr/include/linux/fs.h:

```
struct file_operations {
    int  (*lseek)   (struct inode *, struct file *, off_t, int);
    int  (*read)    (struct inode *, struct file *, char *, int);
    int  (*write)   (struct inode *, struct file *, char *, int);
    int  (*readdir) (struct inode *, struct file *, struct dirent *, int count);
    int  (*select)  (struct inode *, struct file *, int, select_table *);
    int  (*ioctl)   (struct inode *, struct file *, unsigned int, unsigned int);
    int  (*mmap)    (struct inode *, struct file *, unsigned long, size_t, int,
unsigned long);
    int  (*open)    (struct inode *, struct file *);
    void (*release) (struct inode *, struct file *);
};
```

Essentially, this structure constitutes a parital list of the functions that you may have to write to create your driver.

This section details the actions and requirements of the functions in the `file_operations` structure. It documents all the arguments that these functions take. **[It should also detail all the defaults, and cover more carefully the possible return values.]**

The `lseek()` function

This function is called when the system call `lseek()` is called on the device special file representing your device. An understanding of what the system call `lseek()` does should be sufficient to explain this function, which moves to the desired offset. It takes these four arguments:

`struct inode * inode`
: Pointer to the inode structure for this device.

`struct file * file`
: Pointer to the file structure for this device.

`off_t offset`
: Offset **from origin** to move to.

`int origin`
: 0 = take the offset from absolute offset 0 (the beginning).
1 = take the offset from the current position.
2 = take the offset from the end.

`lseek()` returns `-errno` on error, or the absolute position (>= 0) after the lseek.

If there is no `lseek()`, the kernel will take the default action, which is to modify the `file->f_pos` element. For an `origin` of 2, the default action is to return `-EINVAL` if `file->f_inode` is NULL, otherwise it sets `file->f_pos` to `file->f_inode->i_size` + `offset`. Because of this, if `lseek()` should return an error for your device, you must write an `lseek()` function which returns that error.

The `read()` and `write()` functions

The read and write functions read and write a character string to the device. If there is no `read()` or `write()` function in the `file_operations` structure registered with the kernel, and the device is a character device, `read()` or `write()` system calls, respectively, will return `-EINVAL`. If the device is a block device, these functions should not be implemented, as the VFS will route requests through the buffer cache, which will call your strategy routine. The `read` and `write` functions take these arguments:

`struct inode * inode`
: This is a pointer to the inode of the device special file which was accessed. From this, you can do several things, based on the `struct inode` declaration about 100 lines into /usr/include/linux/fs.h. For instance, you can find the minor number of the file by this construction: `unsigned int minor = MINOR(inode->i_rdev);` The definition of the `MINOR` macro is in , as are many other useful definitions. Read fs.h and a few device drivers for more details, and see Supporting Functions for a short description. `inode->i_mode` can be used to find the mode of the file, and there are macros available for this, as well.

`struct file * file`
: Pointer to file structure for this device.

`char * buf`
: This is a buffer of characters to read or write. It is located in *user-space* memory, and therefore must be accessed using the `get_fs*(), put_fs*(),` and `memcpy*fs()` macros detailed in Supporting Functions. User-space memory is inaccessible during an interrupt, so if your driver is interrupt driven, you will have to copy the contents of your buffer into a queue.

`int count`
: This is a count of characters in `buf` to be read or written. It is the size of `buf`, and is how you know that you have reached the end of `buf`, as `buf` is not guaranteed to be null-terminated.

The `readdir()` function

This function is another artifact of `file_operations` being used for implementing filesystems as well as device drivers. Do not implement it. The kernel will return `-ENOTDIR` if the system call `readdir()` is called on your device special file.

The `select()` function

The `select()` function is generally most useful with character devices. It is usually used to multiplex reads without polling--the application calls the `select()` system call, giving it a list of file descriptors to watch, and the kernel reports back to the program on which file descriptor has woken it up. It is also used as a timer. However, the `select()` function in your device driver is not directly called by the system call `select()`, and so the `file_operations` `select()` only needs to do a few things. Its arguments are:

`struct inode * inode`
: Pointer to the inode structure for this device.

`struct file * file`
: Pointer to the file structure for this device.

`int sel_type`
: The select type to perform:

`SEL_IN`	read
`SEL_OUT`	write
`SEL_EX`	exception

`select_table * wait`
: If `wait` is not NULL and there is no error condition caused by the select, `select()` should put the process to sleep, and arrange to be woken up when the device becomes ready, usually through an interrupt. If `wait` is NULL, then the driver should quickly see if the device is ready, and return even if it is not. The `select_wait()` function does this already.

If the calling program wants to wait until one of the devices upon which it is selecting becomes available for the operation it is interested in, the process will have to be put to sleep until one of those operations becomes available. This does **not** require use of a `sleep_on*()` function, however. Instead the `select_wait()` function is used. (See Supporting Functions for the definition of the `select_wait()` function). The sleep state that `select_wait()` will cause is the same as that of `sleep_on_interruptible()`, and, in fact, `wake_up_interruptible()` is used to wake up the process.

However, `select_wait()` will not make the process go to sleep right away. It returns directly, and the `select()` function you wrote should then return. The process isn't put to sleep until the system call `sys_select()`, which originall called your `select()` function, uses the information given to it by the `select_wait()` function to put the process to sleep. `select_wait()` adds the process to the wait queue, but `do_select()` (called from `sys_select()`) actually puts the process to sleep by changing the process state to `TASK_INTERRUPTIBLE` and calling `schedule()`.

The first argument to `select_wait()` is the same `wait_queue` that should be used for a `sleep_on()`, and the second is the `select_table` that was passed to your `select()` function.

After having explained all this in excruciating detail, here are two rules to follow:

1. Call `select_wait()` if the device is not ready, and return 0.
2. Return 1 if the device is ready.

If you provide a `select()` function, do not provide timeouts by setting `current->timeout`, as the `select()` mechanism uses `current->timeout`, and the two methods cannot co-exist, as there is only one `timeout` for each process. Instead, consider using a timer to provide timeouts. See the description of the `add_timer()` function in Supporting Functions for details.

The `ioctl()` function

The `ioctl()` function processes ioctl calls. The structure of your `ioctl()` function will be: first error checking, then one giant (possibly nested) switch statement to handle all possible ioctls. The ioctl number is passed as `cmd`, and the argument to the ioctl is passed as `arg`. It is good to have an understanding of how `ioctls` ought to work before making them up. If you are not sure about your ioctls, do not feel ashamed to ask someone knowledgeable about it, for a few reasons: you may not even need an ioctl for your purpose, and if you do need an ioctl, there may be a better way to do it than what you have thought of. Since ioctls are the least regular part of the device interface, it takes perhaps the most work to get this part right. Take the time and energy you need to get it right.

The first thing you need to do is look in Documentation/ioctl-number.txt, read it, and pick an unused number. Then go from there.

`struct inode * inode`
: Pointer to the inode structure for this device.

`struct file * file`
: Pointer to the file structure for this device.

`unsigned int cmd`
: This is the ioctl command. It is generally used as the switch variable for a case statement.

`unsigned int arg`
: This is the argument to the command. This is user defined. Since this is the same size as a `(void *)`, this can be used as a pointer to user space, accessed through the fs register as usual.

Returns:
: `-errno` on error
: Every other return is user-defined.

If the `ioctl()` slot in the `file_operations` structure is not filled in, the VFS will return `-EINVAL`. However, in all cases, if `cmd` is one of `FIOCLEX`, `FIONCLEX`, `FIONBIO`, or `FIOASYNC`, default processing will be done:

`FIOCLEX` (0x5451)
: Sets the close-on-exec bit.

`FIONCLEX` (0x5450)
: Clears the close-on-exec bit.

`FIONBIO` (0x5421)
: If `arg` is non-zero, set `O_NONBLOCK`, otherwise clear `O_NONBLOCK`.

`FIOASYNC` (0x5452)
: If `arg` is non-zero, set `O_SYNC`, otherwise clear `O_SYNC`. `O_SYNC` is not yet implemented, but it is documented here and parsed in the kernel for completeness.

Note that you have to avoid these four numbers when creating your own ioctls, since if they conflict, the VFS ioctl code will interpret them as being one of these four, and act appropriately, causing a very hard-to-track-down bug.

The `mmap()` function

`struct inode * inode`
: Pointer to inode structure for device.

`struct file * file`
: Pointer to file structure for device.

`unsigned long addr`
: Beginning of address in main memory to `mmap()` into.

`size_t len`
: Length of memory to `mmap()`.

`int prot`
: One of:

`PROT_READ`	region can be read.

`PROT_WRITE`	region can be written.
`PROT_EXEC`	region can be executed.
`PROT_NONE`	region cannot be accessed.

`unsigned long off`
: Offset in the file to `mmap()` from. This address in the file will be mapped to address `addr`.

The `open()` and `release()` functions

`struct inode * inode`
: Pointer to inode structure for device.

`struct file * file`
: Pointer to file structure for device.

`open()` is called when a device special files is opened. It is the policy mechanism responsible for ensuring consistency. If only one process is allowed to open the device at once, `open()` should lock the device, using whatever locking mechanism is appropriate, usually setting a bit in some state variable to mark it as busy. If a process already is using the device (if the busy bit is already set) then `open()` should return `-EBUSY`. If more than one process may open the device, this function is responsible to set up any necessary queues that would not be set up in `write()`. If no such device exists, `open()` should return `-ENODEV` to indicate this. Return 0 on success.

`release()` is called only when the process closes its last open file descriptor on the files. **[I am not sure this is true; it might be called on every close.]** If devices have been marked as busy, `release()` should unset the busy bits if appropriate. If you need to clean up `kmalloc()`'ed queues or reset devices to preserve their sanity, this is the place to do it. If no `release()` function is defined, none is called.

The `init()` function

This function is not actually included in the `file_operations` structure, but you are required to implement it, because it is this function that registers the `file_operations` structure with the VFS in the first place--without this function, the VFS could not route any requests to the driver. This function is called when the kernel first boots and is configuring itself. The init function then detects all devices. You will have to call your `init()` function from the correct place: for a character device, this is `chr_dev_init()` in drivers/char/mem.c.

While the `init()` function runs, it registers your driver by calling the proper registration function. For character devices, this is `register_chrdev()`. (See Supporting Functions for more information on the registration functions.) `register_chrdev()` takes three arguments: the major device number (an int), the ``name" of the device (a string), and the address of the *`device`*`_fops file_operations` structure.

When this is done, and a character or block special file is accessed, the VFS filesystem switch automagically routes the call, whatever it is, to the proper function, if a function exists. If the function does not exist, the VFS routines take some default action.

The `init()` function usually displays some information about the driver, and usually reports all hardware found. All reporting is done via the `printk()` function.

Messages

1. using XX_select() for device without interrupts *by Elwood Downey*
2. found reason for select() problem
3. Why do VFS functions get both structs inode and file? *by Reinhold J. Gerharz*

Supporting Functions

Here is a list of many of the most common supporting functions available to the device driver writer. If you find other supporting functions that are useful, please point them out to me. I know this is not a complete list, but I hope it is a helpful one.

`add_request()`

```
static void add_request(struct blk_dev_struct *dev, struct request *
req)
```

This is a static function in ll_rw_block.c, and cannot be called by other code. However, an understanding of this function, as well as an understanding of `ll_rw_block()`, may help you understand the strategy routine.

If the device that the request is for has an empty request queue, the request is put on the queue and the strategy routine is called. Otherwise, the proper place in the queue is chosen and the request is inserted in the queue, maintaining proper order by insertion sort.

Proper order (the elevator algorithm) is defined as:

1. Reads come before writes.
2. Lower minor numbers come before higher minor numbers.
3. Lower block numbers come before higher block numbers.

The elevator algorithm is implemented by the macro `IN_ORDER()`, which is defined in drivers/block/blk.h **[This may have changed somewhat recently, but it shouldn't matter to the driver writer anyway...]**

Defined in: drivers/block/ll_rw_block.c
See also: `make_request()`, `ll_rw_block()`.

`add_timer()`

```
void add_timer(struct timer_list * timer)
#include <linux/timer.h>
```

Installs the timer structures in the list `timer` in the timer list.

The `timer_list` structure is defined by:

```
struct timer_list {
        struct timer_list *next;
        struct timer_list *prev;
        unsigned long expires;
        unsigned long data;
        void (*function)(unsigned long);
};
```

In order to call `add_timer()`, you need to allocate a `timer_list` structure, and then call `init_timer()`, passing it a pointer to your `timer_list`. It will nullify the `next` and `prev` elements, which is the correct initialization. If necessary, you can allocate multiple `timer_list` structures, and link them into a list. Do make sure that you properly initialize all the unused pointers to `NULL`, or the timer code may get very confused.

For each struct in your list, you set three variables:

`expires`
: The number of jiffies (100ths of a second in Linux/86; thousandths or so in Linux/Alpha) after which to time out.

`function`
: Kernel-space function to run after timeout has occured.

`data`
: Passed as the argument to `function` when `function` is called.

Having created this list, you give a pointer to the first (usually the only) element of the list as the argument to `add_timer()`. Having passed that pointer, keep a copy of the pointer handy, because you will need to use it to modify the elements of the list (to set a new timeout when you need a function called again, to change the function to be called, or to change the data that is passed to the function) and to delete the timer, if necessary.

Note: This is *not* process-specific. Therefore, if you want to wake a certain process at a timeout, you will have to use the sleep and wake primitives. The functions that you install through this mechanism will run in the same context that interrupt handlers run in.

Defined in: kernel/sched.c
See also: `timer_table` in include/linux/timer.h, `init_timer()`, `del_timer()`.

cli()

```
#define cli() __asm__ __volatile__ ("cli"::)
#include <asm/system.h>
```

Prevents interrupts from being acknowledged. `cli` stands for ``CLear Interrupt enable".

See also: `sti()`

del_timer

```
void del_timer(struct timer_list * timer)
#include <linux/timer.h>
```

Deletes the timer structures in the list `timer` in the timer list.

The timer list that you delete must be the address of a timer list you have earlier installed with `add_timer()`. Once you have called `del_timer()` to delete the timer from the kernel timer list, you may deallocate the memory used in the `timer_list` structures, as it is no longer referenced by the kernel timer list.

Defined in: kernel/sched.c
See also: `timer_table` in include/linux/timer.h, `init_timer()`, `add_timer()`.

end_request()

```
static void end_request(int uptodate)
#include "blk.h"
```

Called when a request has been satisfied or aborted. Takes one argument:

`uptodate`
  If not equal to 0, means that the request has been satisfied.
  If equal to 0, means that the request has not been satisfied.

If the request was satisfied (`uptodate != 0`), `end_request()` maintains the request list, unlocks the buffer, and may arrange for the scheduler to be run at the next convenient time (`need_resched = 1`; this is implicit in `wake_up()`, and is not explicitly part of `end_request()`), before waking up all processes sleeping on the `wait_for_request` event, which is slept on in `make_request()`, `ll_rw_page()`, and `ll_rw_swap_file()`.

Note: This function is a static function, defined in drivers/block/blk.h for every non-SCSI device that includes blk.h. (SCSI devices do this differently; the high-level SCSI code itself provides this functionality to the low-level device-specific SCSI device drivers.) It includes several defines dependent on static device information, such as the device number. This is marginally faster than a more generic normal C function.

Defined in: kernel/blk_drv/blk.h
See also: `ll_rw_block()`, `add_request()`, `make_request()`.

free_irq()

```
void free_irq(unsigned int irq)
#include <linux/sched.h>
```

Frees an irq previously aquired with `request_irq()` or `irqaction()`. Takes one argument:

`irq`
: interrupt level to free.

Defined in: kernel/irq.c
See also: `request_irq()`, `irqaction()`.

get_user()

```
#define get_user(ptr)
((__typeof__(*(ptr)))__get_user((ptr),sizeof(*(ptr))))
#include <asm/segment.h>
```

Allows a driver to access data in user space, which is in a different segment than the kernel. Derives the type of the argument and the return type automatically. **This means that you have to use types correctly. Shoddy typing will simply fail to work.**

⚠**Note:** these functions may cause implicit I/O, if the memory being accessed has been swapped out, and therefore pre-emption may occur at this point. Do not include these functions in critical sections of your code even if the critical sections are protected by `cli()`/`sti()` pairs, because that implicit I/O will violate the integrity of your `cli()`/`sti()` pair. If you need to get at user-space memory, copy it to kernel-space memory *before* you enter your critical section.

These functions take one argument:

`addr`
: Address to get data from.

Returns:
: Data at that offset in user space.

Defined in: include/asm/segment.h
See also: `memcpy_*fs()`, `put_user()`, `cli()`, `sti()`.

inb(), inb_p()

```
inline unsigned int inb(unsigned short port)
inline unsigned int inb_p(unsigned short port)
#include <asm/io.h>
```

Reads a byte from a port. `inb()` goes as fast as it can, while `inb_p()` pauses before returning.

Some devices are happier if you don't read from them as fast as possible. Both functions take one argument:

`port`
: Port to read byte from.

Returns:
: The byte is returned in the low byte of the 32-bit integer, and the 3 high bytes are unused, and may be garbage.

Defined in: include/asm/io.h
See also: `outb()`, `outb_p()`.

init_timer()

Inline function for initializing `timer_list` structures for use with `add_timer()`.

Defined in: include/linux/timer.h
See also: `add_timer()`.

irqaction()

```
int irqaction(unsigned int irq, struct sigaction *new)
#include <linux/sched.h>
```

Hardware interrupts are really a lot like signals. Therefore, it makes sense to be able to register an interrupt like a signal. The `sa_restorer()` field of the `struct sigaction` is not used, but otherwise it is the same. The int argument to the `sa.handler()` function may mean different things, depending on whether or not the IRQ is installed with the `SA_INTERRUPT` flag. If it is not installed with the `SA_INTERRUPT` flag, then the argument passed to the handler is a pointer to a register structure, and if it is installed with the `SA_INTERRUPT` flag, then the argument passed is the number of the IRQ. For an example of handler set to use the `SA_INTERRUPT` flag, look at how `rs_interrupt()` is installed in drivers/char/serial.c

The `SA_INTERRUPT` flag is used to determine whether or not the interrupt should be a ``fast'' interrupt. Normally, upon return from the interrupt, `need_resched`, a global flag, is checked. If it is set (!= 0), then `schedule()` is run, which may schedule another process to run. They are also run with all other interrupts still enabled. However, by setting the `sigaction` structure member `sa_flags` to `SA_INTERRUPT`, ``fast'' interrupts are chosen, which leave out some processing, and very specifically do not call `schedule()`.

`irqaction()` takes two arguments:

`irq`
: The number of the IRQ the driver wishes to acquire.

`new`

A pointer to a sigaction struct.

Returns:

-EBUSY if the interrupt has already been acquired,
-EINVAL if sa.handler() is NULL,
0 on success.

Defined in: kernel/irq.c
See also: request_irq(), free_irq()

IS_*(inode)

```
IS_RDONLY(inode) ((inode)->i_flags & MS_RDONLY)
IS_NOSUID(inode) ((inode)->i_flags & MS_NOSUID)
IS_NODEV(inode) ((inode)->i_flags & MS_NODEV)
IS_NOEXEC(inode) ((inode)->i_flags & MS_NOEXEC)
IS_SYNC(inode) ((inode)->i_flags & MS_SYNC)
#include <linux/fs.h>
```

These five test to see if the inode is on a filesystem mounted the corresponding flag.

kfree*()

```
#define kfree(x) kfree_s((x), 0)
void kfree_s(void * obj, int size)
#include <linux/malloc.h>
```

Free memory previously allocated with kmalloc(). There are two possible arguments:

obj
: Pointer to kernel memory to free.

size
: To speed this up, if you know the size, use kfree_s() and provide the correct size. This way, the kernel memory allocator knows which bucket cache the object belongs to, and doesn't have to search all of the buckets. (For more details on this terminology, read mm/kmalloc.c.)

[kfree_s() may be obsolete now.]

Defined in: mm/kmalloc.c, include/linux/malloc.h
See also: kmalloc().

kmalloc()

```
void * kmalloc(unsigned int len, int priority)
#include <linux/kernel.h>
```

`kmalloc()` used to be limited to 4096 bytes. It is now limited to 131056 bytes ((32*4096)-16) on Linux/Intel, and twice that on platforms such as Alpha with 8Kb pages. Buckets, which used to be all exact powers of 2, are now a power of 2 minus some small number, except for numbers less than or equal to 128. For more details, see the implementation in mm/kmalloc.c.

`kmalloc()` takes two arguments:

`len`
: Length of memory to allocate. If the maximum is exceeded, kmalloc will log an error message of ``` ``kmalloc of too large a block (%d bytes)." ``` and return `NULL`.

`priority`
: `GFP_KERNEL` or `GFP_ATOMIC`. If `GFP_KERNEL` is chosen, `kmalloc()` may sleep, allowing pre-emption to occur. This is the normal way of calling `kmalloc()`. However, there are cases where it is better to return immediately if no pages are available, without attempting to sleep to find one. One of the places in which this is true is in the swapping code, because it could cause race conditions, and another in the networking code, where things can happen at much faster speed that things could be handled by swapping to disk to make space for giving the networking code more memory. The most important reason for using `GFP_ATOMIC` is if it is being called from an interrupt, when you cannot sleep, and cannot receive other interrupts.

Returns:
: `NULL` on failure.
: Pointer to allocated memory on success.

Defined in: mm/kmalloc.c
See also: `kfree()`

`ll_rw_block()`

```
void ll_rw_block(int rw, int nr, struct buffer_head *bh[])
#include <linux/fs.h>
```

No device driver will ever call this code: it is called only through the buffer cache. However, an understanding of this function may help you understand the function of the strategy routine.

After sanity checking, if there are no pending requests on the device's request queue, `ll_rw_block()` ``plugs" the queue so that the requests don't go out until all the requests are in the queue, sorted by the elevator algorithm. `make_request()` is then called for each request. If the queue had to be plugged, then the strategy routine for that device is not active, and it is called, **with interrupts disabled. It is the responsibility of the strategy routine to re-enable interrupts.**

Defined in: devices/block/ll_rw_block.c
See also: `make_request()`, `add_request()`.

`MAJOR()`

```
#define MAJOR(a) (((unsigned)(a))>>8)
#include <linux/fs.h>
```

This takes a 16 bit device number and gives the associated major number by shifting off the minor number.

See also: `MINOR()`.

make_request()

```
static void make_request(int major, int rw, struct buffer_head *bh)
```

This is a static function in ll_rw_block.c, and cannot be called by other code. However, an understanding of this function, as well as an understanding of `ll_rw_block()`, may help you understand the strategy routine.

`make_request()` first checks to see if the request is readahead or writeahead and the buffer is locked. If so, it simply ignores the request and returns. Otherwise, it locks the buffer and, except for SCSI devices, checks to make sure that write requests don't fill the queue, as read requests should take precedence.

If no spaces are available in the queue, and the request is neither readahead nor writeahead, `make_request()` sleeps on the event `wait_for_request`, and tries again when woken. When a space in the queue is found, the request information is filled in and `add_request()` is called to actually add the request to the queue. **Defined in:** devices/block/ll_rw_block.c
See also: `add_request()`, `ll_rw_block()`.

MINOR()

```
#define MINOR(a) ((a)&0xff)
#include <linux/fs.h>
```

This takes a 16 bit device number and gives the associated minor number by masking off the major number.

See also: `MAJOR()`.

memcpy_*fs()

```
inline void memcpy_tofs(void * to, const void * from, unsigned long
n)
inline void memcpy_fromfs(void * to, const void * from, unsigned
long n)
#include <asm/segment.h>
```

Copies memory between user space and kernel space in chunks larger than one byte, word, or long. Be very careful to get the order of the arguments right!

⚠**Note:** these functions may cause implicit I/O, if the memory being accessed has been swapped out, and therefore pre-emption may occur at this point. Do not include these functions in critical sections of your code, even if the critical sections are protected by `cli()`/`sti()` pairs, because implicit I/O will violate the `cli()` protection. If you need to get at user-space memory, copy it to kernel-space memory *before* you enter your critical section.

These functions take three arguments:

`to`
: Address to copy data to.

`from`
: Address to copy data from.

`n`
: Number of bytes to copy.

Defined in: include/asm/segment.h
See also: `get_user()`, `put_user()`, `cli()`, `sti()`.

`outb(), outb_p()`

```
inline void outb(char value, unsigned short port)
inline void outb_p(char value, unsigned short port)
#include <asm/io.h>
```

Writes a byte to a port. `outb()` goes as fast as it can, while `outb_p()` pauses before returning. Some devices are happier if you don't write to them as fast as possible. Both functions take two arguments:

`value`
: The byte to write.

`port`
: Port to write byte to.

Defined in: include/asm/io.h
See also: `inb()`, `inb_p()`.

`printk()`

```
int printk(const char* fmt, ...)
#include <linux/kernel.h>
```

`printk()` is a version of `printf()` for the kernel, with some restrictions. It cannot handle floats, and has a few other limitations, which are documented in kernel/vsprintf.c. It takes a variable number of arguments:

`fmt`
: Format string, `printf()` style.

`...`
: The rest of the arguments, `printf()` style.

Returns:
: Number of bytes written.

⚠**Note:** `printk()` may cause implicit I/O, if the memory being accessed has been swapped out, and therefore pre-emption may occur at this point. Also, `printk()` will set the interrupt enable flag, so **never use it in code protected by `cli()`.** Because it causes I/O, it is not safe to use in protected code anyway, even it if didn't set the interrupt enable flag.

Defined in: kernel/printk.c.

`put_user()`

```
#define put_user(x,ptr) __put_user((unsigned
long)(x),(ptr),sizeof(*(ptr)))
#include <asm/segment.h>
```

Allows a driver to write data in user space, which is in a different segment than the kernel. Derives the type of the arguments and the storage size automatically. **This means that you have to use types correctly. Shoddy typing will simply fail to work.**

⚠**Note:** these functions may cause implicit I/O, if the memory being accessed has been swapped out, and therefore pre-emption may occur at this point. Do not include these functions in critical sections of your code even if the critical sections are protected by `cli()`/`sti()` pairs, because that implicit I/O will violate the integrity of your `cli()`/`sti()` pair. If you need to get at user-space memory, copy it to kernel-space memory *before* you enter your critical section.

These functions take two arguments:

`val`
: Value to write

`addr`
: Address to write data to.

Defined in: asm/segment.h
See also: `memcpy_*fs()`, `get_user()`, `cli()`, `sti()`.

register_*dev()

```
int register_chrdev(unsigned int major, const char *name, struct
file_operations *fops)
int register_blkdev(unsigned int major, const char *name, struct
file_operations *fops)
#include <linux/fs.h>
#include <linux/errno.h>
```

Registers a device with the kernel, letting the kernel check to make sure that no other driver has already grabbed the same major number. Takes three arguments:

`major`
: Major number of device being registered.

`name`
: Unique string identifying driver. Used in the output for the /proc/devices file.

`fops`
: Pointer to a `file_operations` structure for that device. This must **not** be `NULL`, or the kernel will panic later.

Returns:
: `-EINVAL` if major is >= `MAX_CHRDEV` or `MAX_BLKDEV` (defined in), for character or block devices, respectively.
: `-EBUSY` if major device number has already been allocated.
: 0 on success.

Defined in: fs/devices.c
See also: `unregister_*dev()`

request_irq()

```
int request_irq(unsigned int irq, void (*handler)(int), unsigned
long flags, const char *device)
#include <linux/sched.h>
#include <linux/errno.h>
```

Request an IRQ from the kernel, and install an IRQ interrupt handler if successful. Takes four arguments:

`irq`
: The IRQ being requested.

`handler`
: The handler to be called when the IRQ occurs. The argument to the handler function will be the number of the IRQ that it was invoked to handle.

`flags`
: Set to `SA_INTERRUPT` to request a ``fast'' interrupt or 0 to request a normal, ``slow'' one.

`device`
: A string containing the name of the device driver, *device*.

Returns:
: `-EINVAL` if `irq` > 15 or `handler` = `NULL`.
: `-EBUSY` if `irq` is already allocated.
: 0 on success.

If you need more functionality in your interrupt handling, use the `irqaction()` function. This uses most of the capabilities of the `sigaction` structure to provide interrupt services similar to to the signal services provided by `sigaction()` to user-level programs.

Defined in: kernel/irq.c
See also: `free_irq()`, `irqaction()`.

`select_wait()`

```
inline void select_wait(struct wait_queue **wait_address,
select_table *p)
#include <linux/sched.h>
```

Add a process to the proper `select_wait` queue. This function takes two arguments:

`wait_address`
: Address of a `wait_queue` pointer to add to the circular list of waits.

`p` p is `NULL`, `select_wait` does nothing, otherwise the current process is put to sleep. This should be the `select_table *wait` variable that was passed to your `select()` function.

Defined in: linux/sched.h
See also: `*sleep_on(), wake_up*()`

`*sleep_on()`

```
void sleep_on(struct wait_queue ** p)
void interruptible_sleep_on(struct wait_queue ** p)
#include <linux/sched.h>
```

Sleep on an event, putting a `wait_queue` entry in the list so that the process can be woken on that event. `sleep_on()` goes into an uninteruptible sleep: The only way the process can run is to be woken by `wake_up()`. `interruptible_sleep_on()` goes into an interruptible sleep that can be woken by signals and process timeouts will cause the process to wake up. A call to `wake_up_interruptible()` is necessary to wake up the process and allow it to continue running where it left off. Both take one argument:

`p`
: Pointer to a proper `wait_queue` structure that records the information needed to wake the

process.

Defined in: kernel/sched.c
See also: `select_wait()`, `wake_up*()`.

sti()

```
#define sti() __asm__ __volatile__ ("sti"::)
#include <asm/system.h>
```

Allows interrupts to be acknowledged. `sti` stands for ``SeT Interrupt enable''.

Defined in: asm/system.h
See also: `cli()`.

sys_get*()

```
int sys_getpid(void)
int sys_getuid(void)
int sys_getgid(void)
int sys_geteuid(void)
int sys_getegid(void)
int sys_getppid(void)
int sys_getpgrp(void)
```

These system calls may be used to get the information described in the table below, or the information can be extracted directly from the process table, like this:
`foo = current->pid;`

`pid`	Process ID
`uid`	User ID
`gid`	Group ID
`euid`	Effective user ID
`egid`	Effective group ID
`ppid`	Process ID of process' parent process
`pgid`	Group ID of process' parent process

The system calls should not be used because they are slower *and* take more space. Because of this, they are no longer exported as symbols throughout the whole kernel.

Defined in: kernel/sched.c

unregister_*dev()

```
int unregister_chrdev(unsigned int major, const char *name)
int unregister_blkdev(unsigned int major, const char *name)
#include <linux/fs.h>
#include <linux/errno.h>
```

Removes the registration for a device device with the kernel, letting the kernel give the major number to some other device. Takes two arguments:

`major`
: Major number of device being registered. Must be the same number given to `register_*dev()`.

`name`
: Unique string identifying driver. Must be the same number given to `register_*dev()`.

Returns:
: `-EINVAL` if major is >= `MAX_CHRDEV` or `MAX_BLKDEV` (defined in `<linux/fs.h>`), for character or block devices, respectively, or if there have not been file operations registered for major device `major`, or if `name` is not the same name that the device was registered with.
: 0 on success.

Defined in: fs/devices.c
See also: `register_*dev()`

wake_up*()

```
void wake_up(struct wait_queue ** p)
void wake_up_interruptible(struct wait_queue ** p)
#include <linux/sched.h>
```

Wakes up a process that has been put to sleep by the matching `*sleep_on()` function. `wake_up()` can be used to wake up tasks in a queue where the tasks may be in a `TASK_INTERRUPTIBLE` or `TASK_UNINTERRUPTIBLE` state, while `wake_up_interruptible()` will only wake up tasks in a `TASK_INTERRUPTIBLE` state, and will be insignificantly faster than `wake_up()` on queues that have only interruptible tasks. These take one argument:

`q`
: Pointer to the `wait_queue` structure of the process to be woken.

Note that `wake_up()` does not switch tasks, it only makes processes that are woken up runnable, so that the next time `schedule()` is called, they will be candidates to run.

Defined in: kernel/sched.c
See also: `select_wait()`, `*sleep_on()`

Messages

14. down/up() - semaphores; set/clear/test_bit() *by Erez Strauss*
13. Bug in printk description! *by Theodore Ts'o*
12. File access within a device driver? *by Paul Osborn*
11. man pages for reguest_region() and release_region() (?) *by mharrison@i-55.com*
10. Can register_*dev() assign an unused major number? *by rgerharz@erols.com*
    1. Register_*dev() can assign an unused major number. *by Reinhold J. Gerharz*
9. memcpy_*fs(): which way is "fs"? *by Reinhold J. Gerharz*
    1. memcpy_tofs() and memcpy_fromfs() *by David Hinds*
8. init_wait_queue() *by Michael K. Johnson*
7. request_irq(...,void *dev_id) *by Robert Wilhelm*
    1. dev_id seems to be for IRQ sharing *by Steven Hunyady*
6. udelay should be mentioned *by Klaus Lindemann*
5. vprintk would be nice... *by Robert Baruch*
    1. RE: vprintk would be nice...
4. add_timer function errata? *by Tim Ferguson*
    1. add_timer function errata *by Tom Bjorkholm*
3. Very short waits *by Kenn Humborg*
2. Add the kill_xxx() family to Supporting functions? *by Burkhard Kohl*
1. Allocating large amount of memory *by Michael K. Johnson*
    1. bigphysarea for Linux 2.0? *by Greg Hager*

down/up() - semaphores; set/clear/test_bit()

Forum: Supporting Functions
Date: Tue, 25 Mar 1997 17:38:15 GMT
From: Erez Strauss <*unknown*>

The following features are almost not documented (AFAIK). semaphore locking with down() up() functions and the usage of them. The bit operations set_bit() clear_bit() and test_bit() are also missing usage information. Those functions are important for drivers programmers that should take care about SMP/resource locking. Please email me <erez@newplaces.com> refrences if you know about.

The KHG is missing an example section. Each function in the Linux kernel should have an example page in the KGH.

Bug in printk description!

Forum: Supporting Functions
Date: Wed, 19 Feb 1997 01:43:48 GMT
From: Theodore Ts'o <tytso@mit.edu>

The printk description states that (and I quote):

``printk() may cause implicit I/O, if the memory being accessed has been swapped out, and therefore pre-emption may occur at this point. Also, printk() will set the interrupt enable flag, so never use it in code protected by cli(). Because it causes I/O, it is not safe to use in protected code anyway, even it if didn't set the interrupt enable flag."

This is wrong! First of all, printk accesses kernel memory, which is never swapped out. Hence, there is no risk of causing implicit I/O. Secondly, printk doesn't use sti(); it uses save_flags()/restore_flags(), so it's safe to use it in an interrupt routine (although it will do horrible things to your interrupt latency, so you obviously only use it for debugging).

File access within a device driver?

Forum: Supporting Functions
Keywords: file access device driver
Date: Wed, 22 Jan 1997 10:51:25 GMT
From: Paul Osborn <pao20@cam.ac.uk>

I have a device driver which locates a custom ISA card in I/O space, and then needs to download a 6kb configuration file to an FPGA on the card.

Which functions should I use to read the datafile? Can stdio.h functions be used, or must special functions be used within the kernel?

man pages for reguest_region() and release_region() (?)

Forum: Supporting Functions
Keywords: release request
Date: Mon, 20 Jan 1997 16:15:26 GMT
From: <mharrison@i-55.com>

```
helo,

Recently I read a series of articles

on writing device drivers in the

Linux Journal.  The author mentions

two functions: release_region(),

request_region().  So far I have

been unlucky in finding man-pages

for these functions.


any clues or hints would be most
appreciated

cheers
Mike
```

Can register_*dev() assign an unused major number?

Forum: Supporting Functions
Date: Thu, 09 Jan 1997 06:32:55 GMT
From: <rgerharz@erols.com>

If you call register_*dev() with major=0, will it return and allocate an unused major number? If so, will it do this for modules, also?

Messages

1. Register_*dev() can assign an unused major number. *by Reinhold J. Gerharz*

Register_*dev() can assign an unused major number.

Forum: Supporting Functions
Re: Can register_*dev() assign an unused major number?
Keywords: register_chrdev major device
Date: Mon, 03 Feb 1997 17:48:13 GMT
From: Reinhold J. Gerharz <rgerharz@erols.com>

If the first parameter to register_chrdev() is zero (0), register_chrdev() will attempt to return an unused major device number. If it returns <0, then the return value is an error code.

(Moderator: Please delete this paragraph and replace my previous message, above, with this one.)

memcpy_*fs(): which way is "fs"?

Forum: Supporting Functions
Keywords: USER KERNEL SPACE MEMORY COPY
Date: Thu, 09 Jan 1997 05:00:55 GMT
From: Reinhold J. Gerharz <rgerharz@erols.com>

memcpy_*fs()

inline void memcpy_tofs(void * to, const void * from, unsigned long n)

inline void memcpy_fromfs(void * to, const void * from, unsigned long n)

It is not clear which way the copy occurs. Does "from" mean user space, or kernel space. Contrarily, does "to" mean kernel space or user space?

Assuming the "tofs" and "fromfs" refer to the Frame Segment register, can one assume it always points to user space? How does this carry over to other architectures? Do they have Frame Segment registers?

Messages

1. memcpy_tofs() and memcpy_fromfs() *by David Hinds*

memcpy_tofs() and memcpy_fromfs()

Forum: Supporting Functions
Re: memcpy_*fs(): which way is "fs"? (Reinhold J. Gerharz)
Keywords: USER KERNEL SPACE MEMORY COPY
Date: Mon, 13 Jan 1997 22:35:53 GMT
From: David Hinds <dhinds@hyper.stanford.edu>

In older versions of the Linux kernel, the i386 FS segment register pointed to user space. So, memcpy_tofs meant to user space, and memcpy_fromfs meant from user space. On other platforms, these did the right thing despite the non-existence of an FS register. These calls are deprecated in current kernels, however, and new code should use copy_from_user() and copy_to_user().

init_wait_queue()

Forum: Supporting Functions
Date: Tue, 19 Nov 1996 17:14:17 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Before calling `sleep_on()` or `wake_up()` on a wait queue, you must initialize it with the `init_wait_queue()` function.

request_irq(...,void *dev_id)

Forum: Supporting Functions
Keywords: request_irq
Date: Tue, 29 Oct 1996 14:54:25 GMT
From: Robert Wilhelm <robert@physiol.med.tu-muenchen.de>

request_irg() and free_irq() seem to take a new parameter in Linux 2.0.x. What is the magic behind this?

Messages

1. dev_id seems to be for IRQ sharing *by Steven Hunyady*

dev_id seems to be for IRQ sharing

Forum: Supporting Functions
Re: request_irq(...,void *dev_id) (Robert Wilhelm)
Keywords: request_irq dev_id IRQ-sharing
Date: Tue, 08 Apr 1997 02:11:34 GMT
From: Steven Hunyady <hunyady@kestrel.nmt.edu>

Look in Don Becker's 3c59x.c net driver. Apparently, IRQ sharing amongst like (or dissimilar?) cards developed progressively in the kernel, and this driver, usable in several major kernel versions, shows this ongoing adaptation. Most other device drivers have not yet allowed for multiple use of IRQ lines, hence they simply put "NULL" for this fifth parameter in the function request_irq() and the second in free_irq().

udelay should be mentioned

Forum: Supporting Functions
Keywords: udelay
Date: Tue, 22 Oct 1996 13:45:41 GMT
From: Klaus Lindemann <lindeman@nbi.dk>

Hi

I think that the function udelay() should be mentioned in this section, since it is not possible to use delay in kernel modules (or at least that how I understood it).

Regards

Klaus Lindemann

vprintk would be nice...

Forum: Supporting Functions
Keywords: printk
Date: Mon, 21 Oct 1996 18:58:25 GMT
From: Robert Baruch <baruch@oramp.com>

```
I wish there were a function analagous to vprintf except for
the kernel -- vprintk.
```

Messages

1. RE: vprintk would be nice...

RE: vprintk would be nice...

Forum: Supporting Functions

Re: vprintk would be nice... (Robert Baruch)

Keywords: printk

Date: Thu, 09 Jan 1997 05:19:03 GMT

From: <unknown>

What's wrong with using sprintf()? I do.

add_timer function errata?

Forum: Supporting Functions
Date: Mon, 07 Oct 1996 09:45:17 GMT
From: Tim Ferguson <timf@dgs.monash.edu.au>

It seems that when using the add_timer function in newer versions of the kernel (2.0.0+), the `expires' variable in the timer_list struct is the time rather than the length of time before the timer will be processed. To be backward compatible with older versions of linux, you need to do something like:

```
if the old version was:
   timer.expires = TIME_LENGTH;

new version would be:
   timer.expires = jiffies + TIME_LENGTH;
```

where TIME_LENGTH is the time in 1/100'ths of a second.

Could anyone tell me if they found this also to be the case, and if so, could the Linux hackers guide please be updated.

```
    thanks,
             Tim.
```

Messages

1. add_timer function errata *by Tom Bjorkholm*

add_timer function errata

Forum: Supporting Functions
Re: add_timer function errata? (Tim Ferguson)
Date: Mon, 17 Feb 1997 17:42:33 GMT
From: Tom Bjorkholm <tomb@mydata.se>

Tim,

You are correct.... or at least I have the same experience as you have. The time you should give is "jiffies + TIMEOUT"

Could someone fix this in the original documentation.

/Tom Bjorkholm

Very short waits

Forum: Supporting Functions
Keywords: short timer jiffies sleep wait
Date: Mon, 23 Sep 1996 20:02:38 GMT
From: Kenn Humborg <kenn@wombat.ie>

Is there any way to wait for less than a jiffy without spinning and tying up the CPU?

I'm trying to implement a key-click and kd_mksound can't make sounds shorter than 10ms.

Thanks

Kenn Humborg kenn@wombat.ie

Add the kill_xxx() family to Supporting functions?

Forum: Supporting Functions
Keywords: kill_xxx(), signaling
Date: Sun, 22 Sep 1996 15:11:54 GMT
From: Burkhard Kohl <b.kohl@ipn-b.comlink.apc.org>

For the development of a char driver I needed functionality to signal an interrupt to the process in user space. The KHG does not give any hint how to do that. Finally, after quite some browsing through kernel sources I came across the kill_xxxx() family in exit.c.

I found kill_pg() and kill_proc() widely used in a couple of char drivers. Another one is kill_fasync() which is mostly used by mouse drivers.

After some hacking I managed to use kill_proc() for my purpose. But I still don't know how to handle the priv parameter correctly. Obviously 0 means without and 1 with certain (what?) priviledges.

I have no idea what kill_fasync is used for.

Wouldn't it be nice to have the kill_xxxx() family described in the KHG? Michael, what do you think? Anyone willing to take this? I could do the stubs if someone who really knows will do the annotation.

Any comment, thoughts and flames are welcome.

Burkhard.

```
P.S. My email address is:
        b.kohl@ipn-b.comlink.apc.org
```

Allocating large amount of memory

Forum: Supporting Functions
Keywords: memory allocation
Date: Mon, 03 Jun 1996 22:25:40 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Matt Welsh has designed a solution to the need for very large areas of continuous physical areas of memory, which is specifically necessary for some DMA needs. If you need it, pick up a copy of bigphysarea, which should work with most modern kernels.

Messages

1. bigphysarea for Linux 2.0? *by Greg Hager*

bigphysarea for Linux 2.0?

Forum: Supporting Functions
Re: Allocating large amount of memory (Michael K. Johnson)
Keywords: memory allocation Linux 2.0
Date: Wed, 24 Jul 1996 08:47:41 GMT
From: Greg Hager <hager@cs.yale.edu>

I acquired the bigphsyarea patch (for a digitizer driver that I am writing), but unfortunately patch -p0 fails on Linux 2.0. Has anyone modifed the patch for 2.0.

Greg

Character Device Drivers

Initialization

Besides functions defined by the `file_operations` structure, there is at least one other function that you will have to write, the `foo_init()` function. You will have to change `chr_dev_init()` in drivers/char/mem.c to call your `foo_init()` function.

`foo_init()` should first call `register_chrdev()` to register itself and avoid device number contention. `register_chrdev()` takes three arguments:

`int major`
: This is the major number which the driver wishes to allocate.

`char *name`
: This is the symbolic name of the driver. This is used, among other things, to report the driver's name in the /proc filesystem.

`struct file_operations *f_ops`
: This is the address of your `file_operations` structure.

Returns:
: 0 if no other character device has registered with the same major number.
: non-0 if the call fails, presumably because another character device has already allocated that major number.

Generally, the `foo_init()` routine will then attempt to detect the hardware that it is supposed to be driving. It should make sure that all necessary data structures are filled out for all present hardware, and have some way of ensuring that non-present hardware does not get accessed. **[Detail different ways of doing this. In particular, document the `request_*` and related functions.]**

Interrupts vs. Polling

In a polling driver, the `foo_read()` and `foo_write()` functions are pretty easy to write. Here is an example of `foo_write()`:

```
static int foo_write(struct inode * inode, struct file * file, char * buf, int count)
{
    unsigned int minor = MINOR(inode->i_rdev);
    char ret;

    while (count > 0) {
        ret = foo_write_byte(minor);
        if (ret < 0) {
            foo_handle_error(WRITE, ret, minor);
            continue;
        }
        buf++ = ret; count--
    }
    return count;
}
```

`foo_write_byte()` and `foo_handle_error()` are either functions defined elsewhere in foo.c or pseudocode. `WRITE` would be a constant or `#define`.

It should be clear from this example how to code the `foo_read()` function as well.

Interrupt-driven drivers are a little more difficult. Here is an example of a `foo_write()` that is interrupt-driven:

```
static int foo_write(struct inode * inode, struct file * file, char * buf, int count)
{
    unsigned int minor = MINOR(inode->i_rdev);
    unsigned long copy_size;
    unsigned long total_bytes_written = 0;
    unsigned long bytes_written;
    struct foo_struct *foo = &foo_table[minor];

    do {
        copy_size = (count <= FOO_BUFFER_SIZE ? count : FOO_BUFFER_SIZE);
        memcpy_fromfs(foo->foo_buffer, buf, copy_size);

        while (copy_size) {
            /* initiate interrupts */

            if (some_error_has_occured)  {
                /* handle error condition */
            }

            current->timeout = jiffies + FOO_INTERRUPT_TIMEOUT;
                /* set timeout in case an interrupt has been missed */
            interruptible_sleep_on(&foo->foo_wait_queue);
            bytes_written = foo->bytes_xfered;
            foo->bytes_written = 0;
            if (current->signal & ~current->blocked) {
                if (total_bytes_written + bytes_written)
                    return total_bytes_written + bytes_written;
                else
                    return -EINTR; /* nothing was written, system
                                      call was interrupted, try again */
            }
        }

        total_bytes_written += bytes_written;
        buf += bytes_written;
        count -= bytes_written;

    } while (count > 0);

    return total_bytes_written;
}

static void foo_interrupt(int irq)
{
    struct foo_struct *foo = &foo_table[foo_irq[irq]];

    /* Here, do whatever actions ought to be taken on an interrupt.
       Look at a flag in foo_table to know whether you ought to be
       reading or writing. */

    /* Increment foo->bytes_xfered by however many characters were
       read or written */

    if (buffer too full/empty)
        wake_up_interruptible(&foo->foo_wait_queue);
}
```

Again, a `foo_read()` function is written analagously. `foo_table[]` is an array of structures, each of which has several members, some of which are `foo_wait_queue` and `bytes_xfered`, which can be used for both reading and writing. `foo_irq[]` is an array of 16 integers, and is used for looking up which entry in `foo_table[]` is associated with the `irq` generated and reported to the `foo_interrupt()` function.

To tell the interrupt-handling code to call `foo_interrupt()`, you need to use either `request_irq()` or `irqaction()`. This is either done when `foo_open()` is called, or if you want to keep things simple, when `foo_init()` is called. `request_irq()` is the simpler of the two, and works rather like an old-style signal handler. It takes two arguments: the first is the number of the `irq` you are requesting, and the second is a pointer to your interrupt handler, which must take an integer argument (the irq that was generated) and have a return type of `void`. `request_irq()` returns `-EINVAL` if `irq` > 15 or if the pointer to the interrupt handler is `NULL`, `-EBUSY` if that interrupt has already been taken, or 0 on success.

`irqaction()` works rather like the user-level `sigaction()`, and in fact reuses the `sigaction` structure. The `sa_restorer()` field of the sigaction structure is not used, but everything else is the same. See the entry for `irqaction()` in Supporting Functions, for further information about `irqaction()`.

Messages

3. release() method called when close is called
2. return value of foo_write(...) *by My name here*
1. TTY drivers *by Daniel Taylor*
    1. Is anything in the works? If not ... *by Andrew Manison*

release() method called when close is called

Forum: Character Device Drivers
Keywords: release method close fclose
Date: Sat, 26 Apr 1997 03:07:04 GMT
From: <unknown>

```
        I just finished a character device driver and I it appears that when fclose()
is called on the device the release() method is called as well even if the device has
been opened multiple times.
```

return value of foo_write(...)

Forum: Character Device Drivers
Keywords: return values
Date: Fri, 25 Apr 1997 21:42:46 GMT
From: My name here <wicksr@swami.indy.tce.com>

In this section I noticed the example foo_write function returns 0 all the time. If I do this as well with my driver and do this:

```
        echo "test" > /dev/foo_drv
```

the foo_write () function gets called indefinately. Furthermore, I have noticed that from the source of serial.c (from /usr/src/linux-2.0.0/drivers/char) always returns the number of characters transmitted. Do you have a typo?

Also, why isn't there a DEFINITIVE list of return values for all functions? This is a bit confusing, but still much better than programming under NT :).

```
        Thanks for the documentation anyhow!
```

-Rich

TTY drivers

Forum: Character Device Drivers
Keywords: serial tty section
Date: Fri, 27 Sep 1996 18:48:12 GMT
From: Daniel Taylor <danielt@dgii.com>

It is noted in several places that there is no section for serial drivers, and yet in this new medium there is not even a pointer to get started from. As the number of these drivers is increasing, even a bodiless section of the KHG would be useful, it can be entirely filled online.

Messages

1. Is anything in the works? If not ... *by Andrew Manison*

Is anything in the works? If not ...

Forum: Character Device Drivers
Re: TTY drivers (Daniel Taylor)
Keywords: serial tty section
Date: Fri, 13 Dec 1996 04:28:33 GMT
From: Andrew Manison <amanison@america.net>

I am in the process of writing a device driver for an intelligent multiport serial I/O controller. I am willing to write a section on tty drivers for the KHG if no-one else is. Let me know!

Block Device Drivers

[Note: This has not been updated since changes were made in the block device interface to support block device loadable modules. The changes shouldn't make it impossible for you to apply any of this...]

To mount a filesystem on a device, it must be a block device driven by a block device driver. This means that the device must be a random access device, not a stream device. In other words, you must be able to seek to any location on the physical device at any time.

You do not provide `read()` and `write()` routines for a block device. Instead, your driver uses `block_read()` and `block_write()`, which are generic functions, provided by the VFS, which will call the **strategy** routine, or `request()` function, which you write in place of `read()` and `write()` for your driver. This strategy routine is also called by the **buffer cache**, which is called by the VFS routines, which is how normal files on normal filesystems are read and written.

Requests for I/O are given by the buffer cache to a routine called `ll_rw_block()`, which constructs lists of requests ordered by an **elevator algorithm,** which sorts the lists to make accesses faster and more efficient. It, in turn, calls your `request()` function to actually do the I/O.

Note that although SCSI disks and CDROMs are considered block devices, they are handled specially (as are all SCSI devices). Refer to Writing a SCSI Driver for details. (Although SCSI disks and CDROMs are block devices, SCSI tapes, like other tapes, are generally character devices.)

Initialization

Initialization of block devices is a bit more complex than initialization of character devices, especially as some ``initialization'' has to be done at compile time. There is also a `register_blkdev()` call that corresponds to the character device `register_chrdev()` call, which the driver must call to say that it is present, working, and active.

The file blk.h

At the top of your driver code, after all other included header files, you need to write two lines of code:

```
#define MAJOR_NR DEVICE_MAJOR
#include "blk.h"
```

where *`DEVICE`*`_MAJOR` is the major number of your device. drivers/block/blk.h requires the use of the `MAJOR_NR` define to set up many other defines and macros for your driver.

Now you need to edit blk.h. Under `#ifdef MAJOR_NR`, there is a section of defines that are conditionally included for certain major numbers, protected by `#elif (MAJOR_NR == DEVICE_MAJOR)`. At the end of this list, you will add another section for your driver. In that section, the following lines are required:

```
#define DEVICE_NAME         "device"
#define DEVICE_REQUEST      do_dev_request
#define DEVICE_ON(device)   /* usually blank, see below */
#define DEVICE_OFF(device)  /* usually blank, see below */
#define DEVICE_NR(device)   (MINOR(device))
```

`DEVICE_NAME` is simply the device name. See the other entries in blk.h for examples.

`DEVICE_REQUEST` is your strategy routine, which will do all the I/O on the device. See The Strategy Routine for more details on the strategy routine.

`DEVICE_ON` and `DEVICE_OFF` are for devices that need to be turned on and off, like floppies. In fact, the floppy driver is currently the only device driver which uses these defines.

`DEVICE_NR(device)` is used to determine the number of the physical device from the minor device number. For instance, in the `hd` driver, since the second hard drive starts at minor 64, `DEVICE_NR(device)` is defined to be `(MINOR(device)>>6)`.

If your driver is interrupt-driven, you will also set

```
#define DEVICE_INTR do_dev
```

which will become a variable automatically defined and used by the remainder of blk.h, specifically by the `SET_INTR()` and `CLEAR_INTR` macros.

You might also consider setting these defines:

```
#define DEVICE_TIMEOUT DEV_TIMER
#define TIMEOUT_VALUE n
```

where *n* is the number of jiffies (clock ticks; hundredths of a second on Linux/386; thousandths or so on Linux/Alpha) to time out after if no interrupt is received. These are used if your device can become ``stuck'': a condition where the driver waits indefinitely for an interrupt that will never arrive. If you define these, they will automatically be used in `SET_INTR` to make your driver time out. Of course, your driver will have to be able to handle the possibility of being timed out by a timer.

Recognizing PC standard partitions

[Inspect the routines in genhd.c and include detailed, correct instructions on how to use them to allow your device to use the standard dos partitioning scheme. By now, bsd disklabel and sun's SMD labelling are also supported, and I still haven't gotten around to documenting this. Shame on me--but people seem to have been able to figure it out anyway `:-)`]

The Buffer Cache

[Here, it should be explained briefly how `ll_rw_block()` is called, about `getblk()` and `bread()` and `breada()` and `bwrite()`, etc. A real explanation of the buffer cache is reserved for the VFS reference section. Jean-Marc Lugrin wrote one, but I can't find him now.]

The Strategy Routine

All reading and writing of blocks is done through the **strategy routine**. This routine takes no arguments and returns nothing, but it knows where to find a list of requests for I/O (`CURRENT`, defined by default as `blk_dev[MAJOR_NR].current_request`), and knows how to get data from the device into the blocks. It is called with interrupts **disabled** so as to avoid race conditions, and is responsible for turning on interrupts with a call to `sti()` before returning.

The strategy routine first calls the `INIT_REQUEST` macro, which makes sure that requests are really on the request list and does some other sanity checking. `add_request()` will have already sorted the requests in the proper order according to the elevator algorithm (using an insertion sort, as it is called once for every request), so the strategy routine ``merely'' has to satisfy the request, call `end_request(1)`, which will take the request off the list, and then if there is still another request on the list, satisfy it and call `end_request(1)`, until there are no more requests on the list, at which time it returns.

If the driver is interrupt-driven, the strategy routine need only schedule the first request to occur, and have the interrupt-handler call `end_request(1)` and the call the strategy routine again, in order to schedule the next request. If the driver is not interrupt-driven, the strategy routine may not return until all I/O is complete.

If for some reason I/O fails permanently on the current request, `end_request(0)` must be called to destroy the request.

A request may be for a read or write. The driver determines whether a request is for a read or write by examining `CURRENT->cmd`. If `CURRENT->cmd == READ`, the request is for a read, and if `CURRENT->cmd == WRITE`, the request is for a write. If the device has seperate interrupt routines for handling reads and writes, `SET_INTR(`*n*`)` must be called to assure that the proper interrupt routine will be called.

[Here I need to include samples of both a polled strategy routine and an interrupt-driven one. The interrupt-driven one should provide seperate read and write interrupt routines to show the use of `SET_INTR`.]

Messages

1. non-block-cached block device? *by Neal Tucker*
2. Shall I explain elevator algorithm (+sawtooth etc) *by Michael De La Rue*

Writing a SCSI Device Driver

[Note that this document has not been revised since its copyright date of 1993. Most things still apply, but some of the facts like the list of currently supported SCSI host adaptors are rather out of date by now.]

Why You Want to Write a SCSI Driver

Currently, the Linux kernel contains drivers for the following SCSI host adapters: Adaptec 1542, Adaptec 1740, Future Domain TMC-1660/TMC-1680, Seagate ST-01/ST-02, UltraStor 14F, and Western Digital WD-7000. You may want to write your own driver for an unsupported host adapter. You may also want to re-write or update one of the existing drivers.

What is SCSI?

The foreword to the SCSI-2 standard draft [ANS] gives a succinct definition of the Small Computer System Interface and briefly explains how SCSI-2 is related to SCSI-1 and CCS:

> The SCSI protocol is designed to provide an efficient peer-to-peer I/O bus with up to 8 devices, including one or more hosts. Data may be transferred asynchronously at rates that only depend on device implementation and cable length. Synchronous data transfers are supported at rates up to 10 mega-transfers per second. With the 32 bit wide data transfer option, data rates of up to 40 megabytes per second are possible.
>
> SCSI-2 includes command sets for magnetic and optical disks, tapes, printers, processors, CD-ROMs, scanners, medium changers, and communications devices.
>
> In 1985, when the first SCSI standard was being finalized as an American National Standard, several manufacturers approached the X3T9.2 Task Group. They wanted to increase the mandatory requirements of SCSI and to define further features for direct-access devices. Rather than delay the SCSI standard, X3T9.2 formed an ad hoc group to develop a working paper that was eventually called the Common Command Set (CCS). Many disk products were designed using this working paper in conjunction with the SCSI standard.

In parallel with the development of the CCS working paper, X3T9.2 began work on an enhanced SCSI standard which was named SCSI-2. SCSI-2 included the results of the CCS working paper and extended them to all device types. It also added caching commands, performance enhancement features, and other functions that X3T9.2 deemed worthwhile. While SCSI-2 has gone well beyond the original SCSI standard (now referred to as SCSI-1), it retains a high degree of compatibility with SCSI-1 devices.

SCSI phases

The ``SCSI bus" transfers data and state information between interconnected SCSI devices. A single transaction between an ``initiator" and a ``target" can involve up to 8 distinct ``phases." These phases are almost entirely determined by the target (e.g., the hard disk drive). The current phase can be determined from an examination of five SCSI bus signals, as shown in this table [LXT91, p. 57].

-SEL	-BSY	-MSG	-C/D	-I/O	PHASE
HI	HI	?	?	?	BUS FREE
HI	LO	?	?	?	ARBITRATION
I	I&T	?	?	?	SELECTION
T	I&T	?	?	?	RESELECTION
HI	LO	HI	HI	HI	DATA OUT
HI	LO	HI	HI	LO	DATA IN
HI	LO	HI	LO	HI	COMMAND
HI	LO	HI	LO	LO	STATUS
HI	LO	LO	LO	HI	MESSAGE OUT
HI	LO	LO	LO	LO	MESSAGE IN

I = Initiator Asserts, T = Target Asserts, ? = HI or LO

Some controllers (notably the inexpensive Seagate controller) require direct manipulation of the SCSI bus--other controllers automatically handle these low-level details. Each of the eight phases will be described in detail.

BUS FREE Phase
: The BUS FREE phase indicates that the SCSI bus is idle and is not currently being used.

ARBITRATION Phase
: The ARBITRATION phase is entered when a SCSI device attempts to gain control of the SCSI bus. Arbitration can start only if the bus was previously in the BUS FREE phase. During arbitration, the arbitrating device asserts its SCSI ID on the DATA BUS. For example, if the arbitrating device's SCSI ID is 2, then the device will assert `0x04`. If multiple devices attempt simultaneous arbitration, the device with the highest SCSI ID will win. Although ARBITRATION is optional in the SCSI-1 standard, it is a required phase in the SCSI-2 standard.

SELECTION Phase
: After ARBITRATION, the arbitrating device (now called the initiator) asserts the SCSI ID of the

target on the DATA BUS. The target, if present, will acknowledge the selection by raising the -BSY line. This line remains active as long as the target is connected to the initiator.

RESELECTION Phase

The SCSI protocol allows a device to disconnect from the bus while processing a request. When the device is ready, it reconnects to the host adapter. The RESELECTION phase is identical to the SELECTION phase, with the exception that it is used by the disconnected target to reconnect to the original initiator. Drivers which do not currently support RESELECTION do not allow the SCSI target to disconnect. RESELECTION should be supported by all drivers, however, so that multiple SCSI devices can simultaneously process commands. This allows dramatically increased throughput due to interleaved I/O requests.

COMMAND Phase

During this phase, 6, 10, or 12 bytes of command information are transferred from the initiator to the target.

DATA OUT and DATA IN Phases

During these phases, data are transferred between the initiator and the target. For example, the DATA OUT phase transfers data from the host adapter to the disk drive. The DATA IN phase transfers data from the disk drive to the host adapter. If the SCSI command does not require data transfer, then neither phase is entered.

STATUS Phase

This phase is entered after completion of all commands, and allows the target to send a status byte to the initiator. There are nine valid status bytes, as shown in the table below [ANS, p. 77]. Note that since bits 1-5 (bit 0 is the least significant bit) are used for the status code (the other bits are reserved), the status byte should be masked with `0x3e` before being examined.

Value*	**Status**
0x00	GOOD
0x02	CHECK CONDITION
0x04	CONDITION MET
0x08	BUSY
0x10	INTERMEDIATE
0x14	INTERMEDIATE-CONDITION MET
0x18	RESERVATION CONFLICT
0x22	COMMAND TERMINATED
0x28	QUEUE FULL

*After masking with 0x3e

The meanings of the three most important status codes are outlined below:

GOOD

The operation completed successfully.

CHECK CONDITION

An error occurred. The REQUEST SENSE command should be used to find out more information about the error (see SCSI Commands).

BUSY

The device was unable to accept a command. This may occur during a self-test or shortly after power-up.

MESSAGE OUT and MESSAGE IN Phases

Additional information is transferred between the target and the initiator. This information may regard the status of an outstanding command, or may be a request for a change of protocol. Multiple MESSAGE IN and MESSAGE OUT phases may occur during a single SCSI transaction. If RESELECTION is supported, the driver must be able to correctly process the SAVE DATA POINTERS, RESTORE POINTERS, and DISCONNECT messages. Although required by the SCSI-2 standard, some devices do not automatically send a SAVE DATA POINTERS message prior to a DISCONNECT message.

SCSI Commands

Each SCSI command is 6, 10, or 12 bytes long. The following commands must be well understood by a SCSI driver developer.

REQUEST SENSE

Whenever a command returns a CHECK CONDITION status, the high-level Linux SCSI code automatically obtains more information about the error by executing the REQUEST SENSE. This command returns a sense key and a sense code (called the ``additional sense code,'' or ASC, in the SCSI-2 standard [ANS]). Some SCSI devices may also report an ``additional sense code qualifier'' (ASCQ). The 16 possible sense keys are described in the next table. For information on the ASC and ASCQ, please refer to the SCSI standard [ANS] or to a SCSI device technical manual.

Sense Key	**Description**
`0x00`	NO SENSE
`0x01`	RECOVERED ERROR
`0x02`	NOT READY
`0x03`	MEDIUM ERROR
`0x04`	HARDWARE ERROR
`0x05`	ILLEGAL REQUEST
`0x06`	UNIT ATTENTION
`0x07`	DATA PROTECT
`0x08`	BLANK CHECK
`0x09`	(Vendor specific error)
`0x0a`	COPY ABORTED
`0x0b`	ABORTED COMMAND
`0x0c`	EQUAL
`0x0d`	VOLUME OVERFLOW
`0x0e`	MISCOMPARE

0x0f	RESERVED

TEST UNIT READY

This command is used to test the target's status. If the target can accept a medium-access command (e.g., a READ or a WRITE), the command returns with a GOOD status. Otherwise, the command returns with a CHECK CONDITION status and a sense key of NOT READY. This response usually indicates that the target is completing power-on self-tests.

INQUIRY

This command returns the target's make, model, and device type. The high-level Linux code uses this command to differentiate among magnetic disks, optical disks, and tape drives (the high-level code currently does not support printers, processors, or juke boxes).

READ and WRITE

These commands are used to transfer data from and to the target. You should be sure your driver can support simpler commands, such as TEST UNIT READY and INQUIRY, before attempting to use the READ and WRITE commands.

Getting Started

The author of a low-level device driver will need to have an understanding of how interruptions are handled by the kernel. At minimum, the kernel functions that disable (`cli()`) and enable (`sti()`) interruptions should be understood. The scheduling functions (e.g., `schedule()`, `sleepon()`, and `wakeup()`) may also be needed by some drivers. A detailed explanation of these functions can be found in Supporting Functions.

Before You Begin: Gathering Tools

Before you begin to write a SCSI driver for Linux, you will need to obtain several resources.

The most important is a bootable Linux system--preferably one which boots from an IDE, RLL, or MFM hard disk. During the development of your new SCSI driver, you will rebuild the kernel and reboot your system many times. Programming errors may result in the destruction of data on your SCSI drive *and* on your non-SCSI drive. *Back up your system before you begin*.

The installed Linux system can be quite minimal: the GCC compiler distribution (including libraries and the binary utilities), an editor, and the kernel source are all you need. Additional tools like `od`, `hexdump`, and `less` will be quite helpful. All of these tools will fit on an inexpensive 20-30~MB hard disk. (A used 20 MB MFM hard disk and controller should cost less than US$100.)

Documentation is essential. At minimum, you will need a technical manual for your host adapter. Since Linux is freely distributable, and since you (ideally) want to distribute your source code freely, avoid non-disclosure agreements (NDA). Most NDA's will prohibit you from releasing your source code--you might be allowed to release an object file containing your driver, but this is simply not acceptable in the Linux community at this time.

A manual that explains the SCSI standard will be helpful. Usually the technical manual for your disk drive will be sufficient, but a copy of the SCSI standard will often be helpful. (The October 17, 1991, draft of the SCSI-2 standard document is available via anonymous ftp from `sunsite.unc.edu` in `/pub/Linux/development/scsi-2.tar.Z`, and is available for purchase from Global Engineering Documents (2805 McGaw, Irvine, CA 92714), (800)-854-7179 or (714)-261-1455. Please refer to document X3.131-199X. In early 1993, the manual cost US$60--70.)

Before you start, make hard copies of `hosts.h`, `scsi.h`, and one of the existing drivers in the Linux kernel. These will prove to be useful references while you write your driver.

The Linux SCSI Interface

The high-level SCSI interface in the Linux kernel manages all of the interaction between the kernel and the low-level SCSI device driver. Because of this layered design, a low-level SCSI driver need only provide a few basic services to the high-level code. The author of a low-level driver does not need to understand the intricacies of the kernel I/O system and, hence, can write a low-level driver in a relatively short amount of time.

Two main structures (`Scsi_Host` and `Scsi_Cmnd`) are used to communicate between the high-level code and the low-level code. The next two sections provide detailed information about these structures and the requirements of the low-level driver.

The `Scsi_Host` Structure

The `Scsi_Host` structure serves to describe the low-level driver to the high-level code. Usually, this description is placed in the device driver's header file in a C preprocessor definition:

```
#define FDOMAIN_16X0  { "Future Domain TMC-16x0",          \
                        fdomain_16x0_detect,                \
                        fdomain_16x0_info,                  \
                        fdomain_16x0_command,               \
                        fdomain_16x0_queue,                 \
                        fdomain_16x0_abort,                 \
                        fdomain_16x0_reset,                 \
                        NULL,                               \
                        fdomain_16x0_biosparam,             \
                        1, 6, 64, 1 ,0, 0}
#endif
```

The `Scsi_Host` structure is presented next. Each of the fields will be explained in detail later in this section.

```
typedef struct
{
  char                 *name;
  int                  (* detect)(int);
  const char           *(* info)(void);
  int                  (* queuecommand)(Scsi_Cmnd *,
                      void (*done)(Scsi_Cmnd *));
  int                  (* command)(Scsi_Cmnd *);
  int                  (* abort)(Scsi_Cmnd *, int);
  int                  (* reset)(void);
  int                  (* slave_attach)(int, int);
  int                  (* bios_param)(int, int, int []);
  int                  can_queue;
  int                  this_id;
  short unsigned int sg_tablesize;
  short                cmd_per_lun;
  unsigned             present:1;
  unsigned             unchecked_isa_dma:1;
} Scsi_Host;
```

Variables in the `Scsi_Host` structure

In general, the variables in the `Scsi_Host` structure are not used until after the `detect()` function (see section `detect()`) is called. Therefore, any variables which cannot be assigned before host adapter detection should be assigned during detection. This situation might occur, for example, if a single driver provided support for several host adapters with very similar characteristics. Some of the parameters in the `Scsi_Host` structure might then depend on the specific host adapter detected.

`name`

`name` holds a pointer to a short description of the SCSI host adapter.

`can_queue`

`can_queue` holds the number of outstanding commands the host adapter can process. Unless RESELECTION is supported by the driver and the driver is interrupt-driven, (some of the early Linux drivers were not interrupt driven and, consequently, had very poor performance) this variable should be set to 1.

`this_id`

Most host adapters have a specific SCSI ID assigned to them. This SCSI ID, usually 6 or 7, is used for RESELECTION. The `this_id` variable holds the host adapter's SCSI ID. If the host adapter does not have an assigned SCSI ID, this variable should be set to -1 (in this case, RESELECTION cannot be supported).

`sg_tablesize`

The high-level code supports ``scatter-gather,'' a method of increasing SCSI throughput by combining many small SCSI requests into a few large SCSI requests. Since most SCSI disk drives are formatted with 1:1 interleave, (``1:1 interleave'' means that all of the sectors in a single track appear consecutively on the disk surface) the time required to perform the SCSI ARBITRATION and SELECTION phases is longer than the rotational latency time between sectors. (This may be an over-simplification. On older devices, the actual command processing can be significant. Further, there is a great deal of layered overhead in the kernel: the high-level SCSI code, the buffering code, and the file-system code all contribute to poor SCSI performance.) Therefore, only one SCSI request can be processed per disk revolution, resulting in a throughput of about 50 kilobytes per second. When scatter-gather is supported, however, average throughput is usually over 500 kilobytes per second.

The `sg_tablesize` variable holds the maximum allowable number of requests in the scatter-gather list. If the driver does not support scatter-gather, this variable should be set to `SG_NONE`. If the driver can support an unlimited number of grouped requests, this variable should be set to `SG_ALL`. Some drivers will use the host adapter to manage the scatter-gather list and may need to limit `sg_tablesize` to the number that the host adapter hardware supports. For example, some Adaptec host adapters require a limit of 16.

`cmd_per_lun`

The SCSI standard supports the notion of ``linked commands.'' Linked commands allow several commands to be queued consecutively to a single SCSI device. The `cmd_per_lun` variable specifies the number of linked commands allowed. This variable should be set to 1 if command linking is not supported. At this time, however, the high-level SCSI code will not take advantage of this feature.

Linked commands are fundamentally different from multiple outstanding commands (as described by the `can_queue` variable). Linked commands always go to the same SCSI target and do not necessarily involve a RESELECTION phase. Further, linked commands eliminate the ARBITRATION, SELECTION, and MESSAGE OUT phases on all commands after the first one in the set. In contrast, multiple outstanding commands may be sent to an arbitrary SCSI target, and *require* the ARBITRATION, SELECTION, MESSAGE OUT, and RESELECTION phases.

`present`

The `present` bit is set (by the high-level code) if the host adapter is detected.

`unchecked_isa_dma`

Some host adapters use Direct Memory Access (DMA) to read and write blocks of data directly from or to the computer's main memory. Linux is a virtual memory operating system that can use more than 16 MB of physical memory. Unfortunately, on machines using the ISA bus (the so-called ``Industry Standard Architecture'' bus was introduced with the IBM PC/XT and IBM PC/AT computers), DMA is limited to the low 16 MB of physical memory.

If the `unchecked_isa_dma` bit is set, the high-level code will provide data buffers which are guaranteed to be in the low 16 MB of the physical address space. Drivers written for host adapters that do not use DMA should set this bit to zero. Drivers specific to EISA bus (the ``Extended Industry Standard Architecture'' bus is a non-proprietary 32-bit bus for 386 and i486 machines) machines should also set this bit to zero, since EISA bus machines allow unrestricted DMA access.

Functions in the `Scsi_Host` Structure

`detect()`

The `detect()` function's only argument is the ``host number,'' an index into the `scsi_hosts` variable (an array of type `struct Scsi_Host`). The `detect()` function should return a non-zero value if the host adapter is detected, and should return zero otherwise.

Host adapter detection must be done carefully. Usually the process begins by looking in the ROM area for the ``BIOS signature'' of the host adapter. On PC/AT-compatible computers, the use of the address space between `0xc0000` and `0xfffff` is fairly well defined. For example, the video BIOS on most machines starts at `0xc0000` and the hard disk BIOS, if present, starts at `0xc8000`. When a PC/AT-compatible computer boots, every 2-kilobyte block from `0xc0000` to `0xf8000` is examined for the 2-byte signature (`0x55aa`) which indicates that a valid BIOS extension is present [Nor85].

The BIOS signature usually consists of a series of bytes that uniquely identifies the BIOS. For example, one Future Domain BIOS signature is the string

```
    FUTURE DOMAIN CORP. (C) 1986-1990 1800-V2.07/28/89
```

found exactly five bytes from the start of the BIOS block.

After the BIOS signature is found, it is safe to test for the presence of a functioning host adapter in more specific ways. Since the BIOS signatures are hard-coded in the kernel, the release of a new BIOS can cause the driver to mysteriously fail. Further, people who use the SCSI adapter exclusively for Linux may want to disable the BIOS to speed boot time. For these reasons, if the adapter can be detected safely without examining the BIOS, then that alternative method should be used.

Usually, each host adapter has a series of I/O port addresses which are used for communications. Sometimes these addresses will be hard coded into the driver, forcing all Linux users who have this host adapter to use a specific set of I/O port addresses. Other drivers are more flexible, and find the current I/O port address by scanning all possible port addresses. Usually each host adapter will allow 3 or 4 sets of addresses, which are selectable via hardware jumpers on the host adapter card.

After the I/O port addresses are found, the host adapter can be interrogated to confirm that it is, indeed, the expected host adapter. These tests are host adapter specific, but commonly include methods to determine the BIOS base address (which can then be compared to the BIOS address found during the BIOS signature search) or to verify a unique identification number associated with the board. For MCA bus (the ``Micro-Channel Architecture'' bus is IBM's proprietary 32 bit bus for 386 and i486 machines) machines, each type of board is given a unique identification number which no other manufacturer can use--several Future Domain host adapters, for example, also use this number as a unique identifier on ISA bus machines. Other methods of verifying the host adapter existence and function will be available to the programmer.

Requesting the IRQ

After detection, the `detect()` routine must request any needed interrupt or DMA channels from the kernel. There are 16 interrupt channels, labeled IRQ 0 through IRQ 15. The kernel provides two methods for setting up an IRQ handler: `irqaction()` and `request_irq()`.

The `request_irq()` function takes two parameters, the IRQ number and a pointer to the handler routine. It then sets up a default `sigaction` structure and calls `irqaction()`. The code (Linux 0.99.7 kernel source code, `linux/kernel/irq.c`) for the `request_irq()` function is shown below. I will limit my discussion to the more general `irqaction()` function.

```
int request_irq( unsigned int irq, void (*handler)( int ) )
{
  struct sigaction sa;

  sa.sa_handler  = handler;
  sa.sa_flags    = 0;
  sa.sa_mask     = 0;
  sa.sa_restorer = NULL;
  return irqaction( irq, &sa );
}
```

The declaration (Linux 0.99.5 kernel source code, `linux/kernel/irq.c`) for the `irqaction()` function is

```
int irqaction( unsigned int irq, struct sigaction *new )
```

where the first parameter, `irq`, is the number of the IRQ that is being requested, and the second parameter, `new`, is a structure with the definition (Linux 0.99.5 kernel source code, `linux/include/linux/signal.h`) shown here:

```
struct sigaction
{
  __sighandler_t sa_handler;
  sigset_t       sa_mask;
  int            sa_flags;
  void           (*sa_restorer)(void);
};
```

In this structure, `sa_handler` should point to your interrupt handler routine, which should have a definition similar to the following:

```
void fdomain_16x0_intr( int irq )
```

where `irq` will be the number of the IRQ which caused the interrupt handler routine to be invoked.

The `sa_mask` variable is used as an internal flag by the `irqaction()` routine. Traditionally, this variable is set to zero prior to calling `irqaction()`.

The `sa_flags` variable can be set to zero or to `SA_INTERRUPT`. If zero is selected, the interrupt handler will run with other interrupts enabled, and will return via the signal-handling return functions. This option is recommended for relatively slow IRQ's, such as those associated with the keyboard and timer interrupts. If `SA_INTERRUPT` is selected, the handler will be called with interrupts disabled and return will avoid the signal-handling return functions. `SA_INTERRUPT` selects ``fast'' IRQ handler invocation routines, and is recommended for interrupt driven hard disk routines. The interrupt handler should turn interrupts on as soon as possible, however, so that other interrupts can be processed.

The `sa_restorer` variable is not currently used, and is traditionally set to `NULL`.

The `request_irq()` and `irqaction()` functions will return zero if the IRQ was successfully assigned to the specified interrupt handler routine. Non-zero result codes may be interpreted as follows:

`-EINVAL`
: Either the IRQ requested was larger than 15, or a `NULL` pointer was passed instead of a valid pointer to the interrupt handler routine.

`-EBUSY`
: The IRQ requested has already been allocated to another interrupt handler. This situation should never occur, and is reasonable cause for a call to `panic()`.

The kernel uses an Intel ``interrupt gate'' to set up IRQ handler routines requested via the `irqaction()` function. The Intel i486 manual [Int90, p. 9-11] explains the interrupt gate as follows:

> Interrupts using... interrupt gates... cause the TF flag [trap flag] to be cleared after its current value is saved on the stack as part of the saved contents of the EFLAGS register. In so doing, the processor prevents instruction tracing from affecting interrupt response. A subsequent IRET [interrupt return] instruction restores the TF flag to the value in the saved contents of the EFLAGS register on the stack.
>
> ... An interrupt which uses an interrupt gate clears the IF flag [interrupt-enable flag], which prevents other interrupts from interfering with the current interrupt handler. A subsequent IRET instruction restores the IF flag to the value in the saved contents of the EFLAGS register on the stack.

Requesting the DMA channel

Some SCSI host adapters use DMA to access large blocks of data in memory. Since the CPU does not have to deal with the individual DMA requests, data transfers are faster than CPU-mediated transfers and allow the CPU to do other useful work during a block transfer (assuming interrupts are enabled).

The host adapter will use a specific DMA channel. This DMA channel will be determined by the `detect()` function and requested from the kernel with the `request_dma()` function. This function takes the DMA channel number as its only parameter and returns zero if the DMA channel was successfully allocated. Non-zero results may be interpreted as follows:

`-EINVAL`
: The DMA channel number requested was larger than 7.

`-EBUSY`
: The requested DMA channel has already been allocated. This is a very serious situation, and will probably cause any SCSI requests to fail. It is worthy of a call to `panic()`.

`info()`

The `info()` function merely returns a pointer to a static area containing a brief description of the low-level driver. This description, which is similar to that pointed to by the `name` variable, will be printed at boot time.

`queuecommand()`

The `queuecommand()` function sets up the host adapter for processing a SCSI command and then returns. When the command is finished, the `done()` function is called with the `Scsi_Cmnd` structure pointer as a parameter. This allows the SCSI command to be executed in an interrupt-driven fashion. Before returning, the `queuecommand()` function must do several things:

1. Save the pointer to the `Scsi_Cmnd` structure.
2. Save the pointer to the `done()` function in the `scsi_done()` function pointer in the `Scsi_Cmnd` structure. See section `done()` for more information.
3. Set up the special `Scsi_Cmnd` variables required by the driver. See section The `Scsi_Cmnd` Structure for detailed information on the `Scsi_Cmnd` structure.
4. Start the SCSI command. For an advanced host adapter, this may be as simple as sending the command to a host adapter ``mailbox." For less advanced host adapters, the ARBITRATION phase is manually started.

The `queuecommand()` function is called *only* if the `can_queue` variable (see section `can_queue`) is non-zero. Otherwise the `command()` function is used for all SCSI requests. The `queuecommand()` function should return zero on success (the current high-level SCSI code presently ignores the return value).

`done()`

The `done()` function is called after the SCSI command completes. The single parameter that this command requires is a pointer to the same `Scsi_Cmnd` structure that was previously passed to the `queuecommand()` function. Before the `done()` function is called, the `result` variable must be set correctly. The `result` variable is a 32 bit integer, each byte of which has specific meaning:

Byte 0 (LSB)
: This byte contains the SCSI STATUS code for the command, as described in section SCSI phases.

Byte 1
: This byte contains the SCSI MESSAGE, as described in section SCSI phases.

Byte 2

This byte holds the host adapter's return code. The valid codes for this byte are given in `scsi.h` and are described below:

`DID_OK`
: No error.

`DID_NO_CONNECT`
: SCSI SELECTION failed because there was no device at the address specified.

`DID_BUS_BUSY`
: SCSI ARBITRATION failed.

`DID_TIME_OUT`
: A time-out occurred for some unknown reason, probably during SELECTION or while waiting for RESELECTION.

`DID_BAD_TARGET`
: The SCSI ID of the target was the same as the SCSI ID of the host adapter.

`DID_ABORT`
: The high-level code called the low-level `abort()` function (see section `abort()`).

`DID_PARITY`
: A SCSI PARITY error was detected.

`DID_ERROR`
: An error occurred which lacks a more appropriate error code (for example, an internal host adapter error).

`DID_RESET`
: The high-level code called the low-level `reset()` function (see section `reset()`).

`DID_BAD_INTR`
: An unexpected interrupt occurred *and* there is no appropriate way to handle this interrupt.

Note that returning `DID_BUS_BUSY` will force the command to be retried, whereas returning `DID_NO_CONNECT` will abort the command.

Byte 3 (MSB)
: This byte is for a high-level return code, and should be left as zero by the low-level code.

Current low-level drivers do not uniformly (or correctly) implement error reporting, so it may be better to consult scsi.c to determine exactly how errors should be reported, rather than exploring existing drivers.

`command()`

The `command()` function processes a SCSI command and returns when the command is finished. When the original SCSI code was written, interrupt-driven drivers were not supported. The old drivers are much less efficient (in terms of response time and latency) than the current interrupt-driven drivers, but are also much easier to write. For new drivers, this command can be replaced with a call to the `queuecommand()` function, as demonstrated here. (Linux 0.99.5 kernel, linux/kernel/blk_drv/scsi/aha1542.c, written by Tommy Thorn.)

```
static volatile int internal_done_flag    = 0;
static volatile int internal_done_errcode = 0;
static void         internal_done( Scsi_Cmnd *SCpnt )
{
  internal_done_errcode = SCpnt->result;
  ++internal_done_flag;
}

int aha1542_command( Scsi_Cmnd *SCpnt )
{
  aha1542_queuecommand( SCpnt, internal_done );

  while (!internal_done_flag);
  internal_done_flag = 0;
  return internal_done_errcode;
}
```

The return value is the same as the `result` variable in the `Scsi_Cmnd` structure. Please see sections `done()` and The `Scsi_Cmnd` Structure for more details.

`abort()`

The high-level SCSI code handles all timeouts. This frees the low-level driver from having to do timing, and permits different timeout periods to be used for different devices (e.g., the timeout for a SCSI tape drive is nearly infinite, whereas the timeout for a SCSI disk drive is relatively short).

The `abort()` function is used to request that the currently outstanding SCSI command, indicated by the `Scsi_Cmnd` pointer, be aborted. After setting the `result` variable in the `Scsi_Cmnd` structure, the `abort()` function returns zero. If `code`, the second parameter to the `abort()` function, is zero, then `result` should be set to `DID_ABORT`. Otherwise, `result` shoudl be set equal to `code`. If `code` is not zero, it is usually `DID_TIME_OUT` or `DID_RESET`.

Currently, none of the low-level drivers is able to correctly abort a SCSI command. The initiator should request (by asserting the `-ATN` line) that the target enter a MESSAGE OUT phase. Then, the initiator should send an ABORT message to the target.

`reset()`

The `reset()` function is used to reset the SCSI bus. After a SCSI bus reset, any executing command should fail with a `DID_RESET` result code (see section `done()`).

Currently, none of the low-level drivers handles resets correctly. To correctly reset a SCSI command, the initiator should request (by asserting the `-ATN` line) that the target enter a MESSAGE OUT phase. Then, the initiator should send a BUS DEVICE RESET message to the target. It may also be necessary to initiate a SCSI RESET by asserting the `-RST` line, which will cause all target devices to be reset. After a reset, it may be necessary to renegotiate a synchronous communications protocol with the targets.

`slave_attach()`

The `slave_attach()` function is *not* currently implemented. This function would be used to negotiate synchronous communications between the host adapter and the target drive. This negotiation requires an exchange of a pair of SYNCHRONOUS DATA TRANSFER REQUEST messages between the initiator and the target. This exchange should occur under the following conditions [LXT91]:

> A SCSI device that supports synchronous data transfer recognizes it has not communicated with the other SCSI device since receiving the last ``hard'' RESET.
>
> A SCSI device that supports synchronous data transfer recognizes it has not communicated with the other SCSI device since receiving a BUS DEVICE RESET message.

`bios_param()`

Linux supports the MS-DOS (MS-DOS is a registered trademark of Microsoft Corporation) hard disk partitioning system. Each disk contains a ``partition table'' which defines how the disk is divided into logical sections. Interpretation of this partition table requires information about the size of the disk in terms of cylinders, heads, and sectors per cylinder. SCSI disks, however, hide their physical geometry and are accessed logically as a contiguous list of sectors. Therefore, in order to be compatible with MS-DOS, the SCSI host adapter will ``lie'' about its geometry. The physical geometry of the SCSI disk, while available, is seldom used as the ``logical geometry.'' (The reasons for this involve archaic and arbitrary limitations imposed by MS-DOS.)

Linux needs to determine the ``logical geometry'' so that it can correctly modify and interpret the partition table. Unfortunately, there is no standard method for converting between physical and logical geometry. Hence, the `bios_param()` function was introduced in an attempt to provide access to the host adapter geometry information.

The `size` parameter is the size of the disk in sectors. Some host adapters use a deterministic formula based on this number to calculate the logical geometry of the drive. Other host adapters store geometry information in tables which the driver can access. To facilitate this access, the `dev` parameter contains the drive's device number. Two macros are defined in `linux/fs.h` which will help to interpret this value: `MAJOR(dev)` is the device's major number, and `MINOR(dev)` is the device's minor number. These are the same major and minor device numbers used by the standard Linux **mknod** command to create the device in the /dev directory. The `info` parameter points to an array of three integers that the `bios_param()` function will fill in before returning:

`info[0]`
: Number of heads

`info[1]`
: Number of sectors per cylinder

`info[2]`
: Number of cylinders

The information in `info` is *not* the physical geometry of the drive, but only a *logical* geometry that is identical to the *logical* geometry used by MS-DOS to access the drive. The distinction between physical and logical geometry cannot be overstressed.

The Scsi_Cmnd Structure

The Scsi_Cmnd structure, (Linux 0.99.7 kernel, linux/kernel/blk_drv/scsi/scsi.h) as shown below, is used by the high-level code to specify a SCSI command for execution by the low-level code. Many variables in the Scsi_Cmnd structure can be ignored by the low-level device driver--other variables, however, are extremely important.

```
    typedef struct scsi_cmnd
    {
      int                host;
      unsigned char      target,
                         lun,
                         index;
      struct scsi_cmnd   *next,
                         *prev;

      unsigned char      cmnd[10];
      unsigned           request_bufflen;
      void               *request_buffer;

      unsigned char      data_cmnd[10];
      unsigned short     use_sg;
      unsigned short     sglist_len;
      unsigned           bufflen;
      void               *buffer;

      struct request     request;
      unsigned char      sense_buffer[16];
      int                retries;
      int                allowed;
      int                timeout_per_command,
                         timeout_total,
                         timeout;
      unsigned char      internal_timeout;
      unsigned           flags;

      void (*scsi_done)(struct scsi_cmnd *);
      void (*done)(struct scsi_cmnd *);

      Scsi_Pointer       SCp;
      unsigned char      *host_scribble;
      int                result;

    } Scsi_Cmnd;
```

Reserved Areas

Informative Variables

`host` is an index into the `scsi_hosts` array.

`target` stores the SCSI ID of the target of the SCSI command. This information is important if multiple outstanding commands or multiple commands per target are supported.

`cmnd` is an array of bytes which hold the actual SCSI command. These bytes should be sent to the SCSI target during the COMMAND phase. `cmnd[0]` is the SCSI command code. The `COMMAND_SIZE` macro, defined in `scsi.h`, can be used to determine the length of the current SCSI command.

`result` is used to store the result code from the SCSI request. Please see section `done()` for more information about this variable. This variable *must* be correctly set before the low-level routines return.

The Scatter-Gather List

`use_sg` contains a count of the number of pieces in the scatter-gather chain. If `use_sg` is zero, then `request_buffer` points to the data buffer for the SCSI command, and `request_bufflen` is the length of this buffer in bytes. Otherwise, `request_buffer` points to an array of `scatterlist` structures, and `use_sg` will indicate how many such structures are in the array. The use of `request_buffer` is non-intuitive and confusing.

Each element of the `scatterlist` array contains an `address` and a `length` component. If the `unchecked_isa_dma` flag in the `Scsi_Host` structure is set to 1 (see section `unchecked_isa_dma` for more information on DMA transfers), the address is guaranteed to be within the first 16 MB of physical memory. Large amounts of data will be processed by a single SCSI command. The length of these data will be equal to the sum of the lengths of all the buffers pointed to by the `scatterlist` array.

Scratch Areas

Depending on the capabilities and requirements of the host adapter, the scatter-gather list can be handled in a variety of ways. To support multiple methods, several scratch areas are provided for the exclusive use of the low-level driver.

The `scsi_done()` Pointer

This pointer should be set to the `done()` function pointer in the `queuecommand()` function (see section `queuecommand()` for more information). There are no other uses for this pointer.

The `host_scribble` Pointer

The high-level code supplies a pair of memory allocation functions, `scsi_malloc()` and `scsi_free()`, which are guaranteed to return memory in the first 16 MB of physical memory. This memory is, therefore, suitable for use with DMA. The amount of memory allocated per request *must* be a multiple of 512 bytes, and *must* be less than or equal to 4096 bytes. The total amount of memory available via `scsi_malloc()` is a complex function of the `Scsi_Host` structure variables `sg_tablesize`, `cmd_per_lun`, and `unchecked_isa_dma`.

The `host_scribble` pointer is available to point to a region of memory allocated with `scsi_malloc()`. The low-level SCSI driver is responsible for managing this pointer and its associated memory, and should free the area when it is no longer needed.

The `Scsi_Pointer` Structure

The `SCp` variable, a structure of type `Scsi_Pointer`, is described here:

```
    typedef struct scsi_pointer
    {
      char                *ptr;             /* data pointer */
      int                 this_residual;    /* left in this buffer */
      struct scatterlist *buffer;           /* which buffer */
      int                 buffers_residual; /* how many buffers left */

      volatile int        Status;
      volatile int        Message;
      volatile int        have_data_in;
      volatile int        sent_command;
      volatile int        phase;
    } Scsi_Pointer;
```

The variables in this structure can be used in *any* way necessary in the low-level driver. Typically, `buffer` points to the current entry in the `scatterlist`, `buffers_residual` counts the number of entries remaining in the `scatterlist`, `ptr` is used as a pointer into the buffer, and `this_residual` counts the characters remaining in the transfer. Some host adapters require support of this detail of interaction--others can completely ignore this structure.

The second set of variables provide convenient locations to store SCSI status information and various pointers and flags.

Acknowledgements

Thanks to Drew Eckhardt, Michael K. Johnson, Karin Boes, Devesh Bhatnagar, and Doug Hoffman for reading early versions of this paper and for providing many helpful comments. Special thanks to my official COMP-291 (Professional Writing in Computer Science) ``readers,'' Professors Peter Calingaert and Raj Kumar Singh.

Bibliography

[ANS]

Draft Proposed American National Standard for Information Systems: Small Computer System Interface-2 (SCSI-2). (X3T9.2/86-109, revision 10h, October 17, 1991).

[Int90]
Intel. *i486 Processor Programmer's Reference Manual.* Intel/McGraw-Hiull, 1990.

[LXT91]
LXT SCSI Products: Specification and OEM Technical Manual, 1991.

[Nor85]
Peter Norton. *The Peter Norton Programmer's Guide to the IBM PC.* Bellevue, Washington: Microsoft Press, 1985.

Messages

1. Writing a SCSI Device Driver *by rohit patil*

Writing a SCSI Device Driver

Forum: Writing a SCSI Device Driver
Keywords: Good work!
Date: Thu, 02 Jan 1997 04:10:44 GMT
From: rohit patil <rohit@techie.com>

```
hi!

this is superb stuff. thanks. will let you know more after
i go thro' it. good work :)

-rohit.
```

non-block-cached block device?

Forum: Block Device Drivers
Keywords: block device cache
Date: Thu, 30 May 1996 11:26:41 GMT
From: Neal Tucker <ntucker@adobe.com>

```
I have a question/idea regarding the block device interface...

First, some premises upon which my idea relies
1) All block device access goes through the block cache.
2) Filesystems must be mounted from block devices.
3) A block device read which is not a cache hit always puts
the calling process to sleep, which means that even if the
IO completes quickly (ie with a RAM disk), the process still
has to wait to be scheduled again.

So...
It seems to me that these three things could lead to very
poor RAM disk performance, which leads me to suggest that
it might be a advantageous to allow block devices which do
not go through the block cache.

I can see three possible reasons this isn't a good idea:
1) With the current design, it would be really hard to do.
2) It doesn't make enough of a difference that people care.
3) I'm completely wrong.

What do people think?
```

Shall I explain elevator algorithm (+sawtooth etc)

Forum: Block Device Drivers
Keywords: block device elevator sawtooth minimum algorithm
Date: Sat, 10 Aug 1996 11:12:11 GMT
From: Michael De La Rue <miked@ed.ac.uk>

I just wrote a response about it to the kernel list, so would a discussion of the elevator algorithm, and sawtooth algorithm (plus mention of minimum movement) be appreciated if I get it checked over by `someone who knows?'

Annotated Bibliography

This annotated bibliography covers books on operating system theory as well as different kinds of programming in a Unix environment. The price marked may or may not be an exact price, but should be close enough for government work. **If you have a book that you think should go in the bibliography, please write a short review of it and send all the necessary information (title, author, publisher, ISBN, and approximate price) and the review to johnsonm@redhat.com**

The Design of the UNIX Operating System

Author:	Maurice J. Bach
Publisher:	Prentice Hall, 1986
ISBN:	0-13-201799-7
Price:	$65.00

This is one of the books that Linus used to design Linux. It is a description of the data structures used in the System V kernel. Many of the names of the important functions in the Linux source come from this book, and are named after the algorithms presented here. For instance, if you can't quite figure out what exactly getblk(), brelse(), bread(), breada(), and bwrite() are, chapter 3 explains very well.

While most of the algorithms are similar or the same, a few differences are worth noting:

- The Linux buffer cache is dynamically resized, so the algorithm for dealing with getting new buffers is a bit different. Therefore the above referenced explanation of getblk() is a little different than the getblk() in Linux.
- Linux does not currently use streams, and if/when streams are implemented for Linux, they are likely to have somewhat different semantics.
- The semantics and calling structure for device drivers is different. The concept is similar, and the chapter on device drivers is still worth reading, but for details on the device driver structures, the KHG is the proper reference.
- The memory management algorithms are somewhat different.

There are other small differences as well, but a good understanding of this text will help you understand the Linux source.

Advanced Programming in the UNIX Environment

Author:	W. Richard Stevens
Publisher:	Addison Wesley, 1992

ISBN: 0-201-56317-7
Price: $50.00

This excellent tome covers the stuff you *really* have to know to write *real* Unix programs. It includes a discussion of the various standards for Unix implementations, including POSIX, X/Open XPG3, and FIPS, and concentrates on two implementations, SVR4 and pre-release 4.4 BSD, which it refers to as 4.3+BSD. The book concentrates heavily on application and fairly complete specification, and notes which features relate to which standards and releases.

The chapters include: Unix Standardization and Implementations, File I/O, Files and Directories, Standard I/O Library, System Data Files and Information, The Environment of a Unix Process, Process Control, Process Relationships, Signals, Terminal I/O, Advanced I/O (non-blocking, streams, async, memory-mapped, etc.), Daemon Processes, Interprocess Communication, Advanced Interprocess Communication, and some example applications, including chapters on A Database Library, Commmunicating with a PostScript Printer, A Modem Dialer, and then a seemingly misplaced final chapter on Pseudo Terminals.

I have found that this book makes it possible for me to write useable programs for Unix. It will help you achieve POSIX compliance in ways that won't break SVR4 or BSD, as a general rule. This book will save you ten times its cost in frustration.

Advanced 80386 Programming Techniques

Author: James L. Turley
Publisher: Osborne McGraw-Hill, 1988
ISBN: 0-07-881342-5
Price: $22.95

This book covers the 80386 quite well, without touching on any other hardware. Some code samples are included. All major features are covered, as are many of the concepts needed. The chapters of this book are: Basics, Memory Segmentation, Privilege Levels, Paging, Multitasking, Communicating Among Tasks, Handling Faults and Interrupts, 80286 Emulation, 8086 Emulation, Debugging, The 80387 Numeric Processor Extension, Programming for Performance, Reset and Real Mode, Hardware, and a few appendices, including tables of the memory management structures as a handy reference.

The author has a good writing style: If you are technically minded, you will find yourself caught up just reading this book. One strong feature of this book for Linux is that the author is very careful not to explain how to do things under DOS, nor how to deal with particular hardware. In fact, the only times he mentions DOS and PC-compatible hardware are in the introduction, where he promises never to mention them again.

The C Programming Language, second edition

Author:	Brian W. Kernighan and Dennis M. Ritchie
Publisher:	Prentice Hall, 1988
ISBN:	0-13-110362-8 (paper) 0-13-110370-9 (hard)
Price:	$35.00

The C programming bible. Includes a C tutorial, Unix interface reference, C reference, and standard library reference.

You program in C, you buy this book. It's that simple.

Operating Systems: Design and Implementation

Author:	Andrew S. Tanenbaum
Publisher:	Prentice Hall, 1987
ISBN:	0-13-637406-9
Price:	$50.00

This book, while a little simplistic in spots, and missing some important ideas, is a fairly clear exposition of what it takes to write an operating system. Half the book is taken up with the source code to a Unix clone called Minix, which is based on a microkernel, unlike Linux, which sports a monolithic design. It has been said that Minix shows that it is possible to to write a microkernel-based Unix, but does not adequately explain *why* one would do so.

Linux was originally intended to be a free Minix replacement (Linus' Minix, Linus tells us). In fact, it was originally to be binary-compatible with Minix-386. Minix-386 was the development environment under which Linux was bootstrapped. No Minix code is in Linux, but vesitiges of this heritage live on in such things as the minix filesystem in Linux.

However, this book might still prove worthwhile for those who want a basic explanation of OS concepts, as Tanenbaum's explanations of the basic concepts remain some of the clearer (and more entertaining, if you like to be entertained) available. Unfortunately, basic is the key work here, as many things such as virtual memory are not covered at all.

Modern Operating Systems

Author:	Andrew S. Tanenbaum
Publisher:	Prentice Hall, 1992
ISBN:	0-13-588187-0
Price:	$51.75

The first half of this book is a rewrite of Tanenbaum's earlier *Operating Systems*, but this book covers several things that the earlier book missed, including such things as virtual memory. Minix is not included, but overviews of MS-DOS and several distributed systems are. This book is probably more useful to someone who wants to do something with his or her knowlege than Tanenbaum's earlier *Operating Systems: Design and Implementation*. Some clue as to the reason may be found in the title... However, what DOS is doing in a book on *modern* operating systems, many have failed to discover.

Operating Systems

Author:	William Stallings
Publisher:	Macmillan, 1992 (800-548-9939)
ISBN:	0-02-415481-4
Price:	No one at Macmillan could find one...

A very thorough text on operating systems, this book gives more in-depth coverage of the topics covered in Tannebaum's books, and covers more topics, in a much brisker style. This book covers all the major topics that you would need to know to build an operating system, and does so in a clear way. The author uses examples from three major systems, comparing and contrasting them: Unix, OS/2, and MVS. With each topic covered, these example systems are used to clarify the points and provide an example of an implementation.

Topics covered in *Operating Systems* include threads, real-time systems, multiprocessor scheduling, distributed systems, process migration, and security, as well as the standard topics like memory management and scheduling. The section on distributed processing appears to be up-to-date, and I found it very helpful.

UNIX Network Programming

Author:	W. Richard Stevens
Publisher:	Prentice Hall, 1990
ISBN:	0-13-949876-1
Price:	$48.75

This book covers several kinds of networking under Unix, and provides very thorough references to the forms of networking that it does not cover directly. It covers TCP/IP and XNS most heavily, and fairly exhaustively describes how all the calls work. It also has a description and sample code using System V's TLI, and pretty complete coverage of System V IPC. This book contains a lot of source code examples to get you started, and many useful proceedures. One example is code to provide useable semaphores, based on the partially broken implementation that System V provides.

Programming in the UNIX environment

Author: Brian W. Kernighan and Robert Pike
Publisher: Prentice Hall, 1984
ISBN: 0-13-937699 (hardcover) 0-13-937681-X (paperback)
Price: ?

Writing UNIX Device Drivers

Author: George Pajari
Publisher: Addison Wesley, 1992
ISBN: 0-201-52374-4
Price: $32.95

This book is written by the President and founder of Driver Design Labs, a company which specializes in the development of Unix device drivers. This book is an excellent introduction to the sometimes wacky world of device driver design. The four basic types of drivers (character, block, tty, STREAMS) are first discussed briefly. Many full examples of device drivers of all types are given, starting with the simplest and progressing in complexity. All examples are of drivers which deal with Unix on PC-compatible hardware.

Chapters include: Character Drivers I: A Test Data Generator Character Drivers II: An A/D Converter Character Drivers III: A Line Printer Block Drivers I: A Test Data Generator Block Drivers II: A RAM Disk Driver Block Drivers III: A SCSI Disk Driver Character Drivers IV: The Raw Disk Driver Terminal Drivers I: The COM1 Port Character Drivers V: A Tape Drive STREAMS Drivers I: A Loop-Back Driver STREAMS Drivers II: The COM1 Port (Revisited) Driver Installation Zen and the Art of Device Driver Writing

Although many of the calls used in the book are not Linux-compatible, the general idea is there, and many of the ideas map directly into Linux.

Messages

1. Please replace K&R reference by Harbison/Steele *by Markus Kuhn*
    1. Replace, no; supplement, yes *by Michael K. Johnson*
    -> Right you are Mike! *by rohit patil*
2. 80386 book is apparently out of print now *by Austin Donnelly*
    1. Very unfortunate *by Michael K. Johnson*
3. Linux Kernel Internals-> Kernel MM IPC fs drivers net modules *by Alex Stewart*

Please replace K&R reference by Harbison/Steele

Forum: Annotated Bibliography
Keywords: C, textbook, K%0Aamp;R replacement, reference manual
Date: Sun, 19 May 1996 12:06:58 GMT
From: Markus Kuhn <mskuhn@cip.informatik.uni-erlangen.de>

I suggest that you replace the K&R reference by the following much better and more up-to-date one. I have never touched my K&R again since I bought the following book. If you don't want to throw K&R out, please add at least this reference and add to the K&R review that this old book does not cover the full ISO C run-time library (e.g. the wide character and locale support is missing almost completely) nor the 1994 C language extensions.

C - A Reference Manual, fourth edition

```
Author: Samuel P. Harbison and Guy L. Steele Jr.

Publisher: Prentice Hall, 1995

ISBN: 0-13-326232-4 (hard) 0-13-326224-3 (paper)

Pages: 455

Price: ??.?? USD, 73.80 DEM
```

This book is an authoritative reference manual that provides a complete and precise description of the C language and the run-time library. It also teaches a C programming style that emphasizes correctness, portability, and maintainability. If you program in C, you want to have this book on your desk, even if you are already a C expert. The authors have been members of the ANSI/ISO C standards committee.

The Harbison/Steele has by now taken over the role of being the C bible from the traditional C book by Kernighan/Ritchie. In contrast to K&R, the Harbison/Steele covers the full ISO C standard, including the 1994 extensions. It also covers the old K&R C language as well as C++ compatibility issues. Especially the description of the standard C library, which every C programmer needs for daily reference, is considerably more complete and precise than the one found in appendix B of K&R.

Messages

1. Replace, no; supplement, yes *by Michael K. Johnson*
-> Right you are Mike! *by rohit patil*

Replace, no; supplement, yes

Forum: Annotated Bibliography
Re: Please replace K&R reference by Harbison/Steele (Markus Kuhn)
Keywords: C, textbook, reference manual
Date: Sun, 19 May 1996 15:27:40 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Since this is the **kernel** hackers' guide, and since the kernel doesn't use the run-time library, the fact that Harbison and Steele document the run-time library fully is rather irrelevant for the purposes of *this* document, even though it is relevant to C programmers in general.

I agree that H&S should be included, but not that K&R should be excluded.

I happen to like K&R and find it easy to read and look things up in; I occasionally supplement it with an annotated (sometimes poorly, IMHO) copy of the ANSI standard published by Osborne/McGraw Hill, ISBN 0-07-881952-90. Even if (like me) you ignore the annotation, it's still cheaper than an official copy of the standard, or was last time I checked. I should probably add it to the bibliography, along with a pointer to the GNU C documentation, since the linux kernel does use a few GNU C extensions.

Messages

1. Right you are Mike! *by rohit patil*

Right you are Mike!

Forum: Annotated Bibliography
Re: Please replace K&R reference by Harbison/Steele (Markus Kuhn)
Re: Replace, no; supplement, yes (Michael K. Johnson)
Keywords: C, textbook, reference manual
Date: Thu, 02 Jan 1997 04:15:15 GMT
From: rohit patil <rohit@techie.com>

```
Yup! Can't replace K&R. Supplement, maybe :)
```

⚠ 80386 book is apparently out of print now

Forum: Annotated Bibliography
Keywords: 80386 programming, out of print
Date: Thu, 23 May 1996 22:54:21 GMT
From: Austin Donnelly <and1000@cam.ac.uk>

I recently tried to buy "Advanced 80386 Programming", and was told that it has just recently gone out of print. This is a great shame, as I can't seem to find a similarly unbiased book. All the books on the subject these days either tell you how to optimise for a Pentium, without telling you about MOV etc, or tell you how to program using DOS interupts. Could the KHG be ammended to include a note saying that finding this book could be trickey? Cheers, Austin

Messages

1. ☹ Very unfortunate *by Michael K. Johnson*

Very unfortunate

Forum: Annotated Bibliography
Re: 80386 book is apparently out of print now (Austin Donnelly)
Keywords: 80386 programming, out of print
Date: Sun, 26 May 1996 17:45:43 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Osborne McGraw-Hill may be bringing the book in and out of print. When I got my copy in 1992, it was out of print but still available in bookstores, so if it just recently went out of print, they probably do infrequent printings and enough noise from potential readers may be sufficient to convince them to bring the book out of retirement.

Write to Osborne McGraw-Hill and let them know you want to buy a copy. Bookstores don't tell them when one person goes in and it's out of print, but publishers do sometimes listen when plenty of potential readers write to them. Their address is (or was when the book was published...)
Osborne McGraw-Hill
2600 Tenth Street
Berkeley, California 94710
U.S.A.

Good luck!

Linux Kernel Internals-> Kernel MM IPC fs drivers net modules

Forum: Annotated Bibliography
Keywords: Must have
Date: Tue, 15 Oct 1996 15:43:14 GMT
From: Alex Stewart <Alex@auriga.rose.brandeis.edu>

Linux Kernel Internals ISBN 0-201-87741-4 Addison Wesley Longman 1996 Beck,Boehme,Dziadzka(!),Kunitz,Magnus,Verworner $45 at Borders

A guide to the kernel code, indispensible to someone like me, who has to make hardware work with Linux, but is unfamiliar with OS details outside of windows/dos. Supprisingly easy to read given the subject matter. This book talks you through the timer and scheduler code and fast and slow H/W interrrupt handlers for instance, so I can see what things like get/setitimer will do for me. Based on 1.2.13 and 1.3.x.

using XX_select() for device without interrupts

Forum: Device Driver Basics
Keywords: select interrupts polling sleeping
Date: Thu, 25 Jul 1996 14:59:48 GMT
From: Elwood Downey <ecdowney@noao.edu>

```
Hello;

I have need for a select() entry point in my driver but my
device is not using interrupts so I'm not sure how to have
the os call my select() to let me poll the device. I think I must use a timer,
with select_wait(), so the system will call my select() until my device becomes
active. The trouble is I can not seem to get the timer work. The entire system
hangs _solid_ whenever it gets activated.

Below is my select() code. I use wake_up_interruptible() as the function the
timer will call in the future to just make this process runnable again, in lieu
of calling it from an interrupt service routine. A few specific questions:

1) my driver permits several processes to have the device open at once. Am I
   correct in assuming that if this general approach works I will need a
   separate timer_list and wait_queue for each open process instance?

2) In no examples do I ever see the wait_queue pointer ever _set_ to point at
   an actual wait_queue instance. Is this correct?


Any comments would be greatly appreciated. Rememeber, the only real goal here
is some way to get the os to call us occasionally to let us poll the device,
but the device is not using interrupts.

Thank you in advance;

Elwood Downey

static int
pc39_select (struct inode *inode, struct file *file, int sel_type,
select_table *wait)
{
        static struct timer_list pc39_tl;
        static struct wait_queue *pc39_wq;

        switch (sel_type) {
        case SEL_EX:
            return (0); /* never any exceptions */
        case SEL_IN:
            if (IBF())
                return (1);
            break;
        case SEL_OUT:
            if (TBE())
                return (1);
            break;
        }

        /* nothing ready -- set timer to try again later if necessary */
        if (wait) {
            init_timer (&pc39_tl);
            pc39_tl.expires = PC39_SELTO;
            pc39_tl.function = (void(*)(unsigned long))wake_up_interruptible;
            pc39_tl.data = (unsigned long) &pc39_wq;
            add_timer (&pc39_tl);
            select_wait (&pc39_wq, wait);
        }
        return (0);
}
```

found reason for select() problem

Forum: Device Driver Basics
Keywords: select add_timer() del_timer()
Date: Wed, 13 Nov 1996 14:45:59 GMT
From: *<unknown>*

```
Hello again;

Evidently not many folks read this -- no responses after
4 months -- so I'll answer my own question :-)

There were several problems with the original approach. These
were all discovered through trial-and-error so I suppose
there might still be other theoretical problems but at least
now everthing seems to work.

1) call del_timer(&pc39_tl) before starting a new one.
2) always call select_wait (&pc39_wq, wait), not just when
   wait != 0.
3) pc39_tl.expires is the jiffy to wake up on, not the number
   of elapsed jiffies as it says in the KHG. so, it should be:
   pc39_tl.expires = jiffies + PC39_SELTO;

Hope this helps someone else someday. If this is getting
too hard to follow, I'll be happy to send you the whole
driver.

Elwood Downey
ecdowney@noao.edu
```

Why do VFS functions get both structs inode and file?

Forum: Device Driver Basics
Date: Thu, 09 Jan 1997 05:47:10 GMT
From: Reinhold J. Gerharz <rgerharz@erols.com>

It appears that "struct file" contains a "struct inode *", yet both are passed to the VFS functions. Why not simply pass "struct file *" alone?

Network Buffers And Memory Management

Reprinted with permission of Linux Journal, from issue 29, September 1996. Some changes have been made to accomodate the web. This article was originally written for the Kernel Korner column. The Kernel Korner series has included many other articles of interest to Linux kernel hackers, as well.

by Alan Cox

The Linux operating system implements the industry-standard Berkeley socket API, which has its origins in the BSD unix developments (4.2/4.3/4.4 BSD). In this article, we will look at the way the memory management and buffering is implemented for network layers and network device drivers under the existing Linux kernel, as well as explain how and why some things have changed over time.

Core Concepts

The networking layer tries to be fairly object-oriented in its design, as indeed is much of the Linux kernel. The core structure of the networking code goes back to the initial networking and socket implementations by Ross Biro and Orest Zborowski respectively. The key objects are:

Device or Interface:
: A network interface represents a thing which sends and receives packets. This is normally interface code for a physical device like an ethernet card. However some devices are software only such as the loopback device which is used for sending data to yourself.

Protocol:
: Each protocol is effectively a different language of networking. Some protocols exist purely because vendors chose to use proprietary networking schemes, others are designed for special purposes. Within the Linux kernel each protocol is a seperate module of code which provides services to the socket layer.

Socket:
: So called from the notion of plugs and sockets. A socket is a connection in the networking that provides unix file I/O and exists to the user program as a file descriptor. In the kernel each socket is a pair of structures that represent the high level socket interface and low level protocol interface.

sk_buff:
: All the buffers used by the networking layers are `sk_buffs`. The control for these is provided by core low-level library routines available to the whole of the networking. `sk_buffs` provide the general buffering and flow control facilities needed by network protocols.

Implementation of `sk_buffs`

The primary goal of the `sk_buff` routines is to provide a consistent and efficient buffer handling method for all of the network layers, and by being consistent to make it possible to provide higher level `sk_buff` and socket handling facilities to all the protocols.

An `sk_buff` is a control structure with a block of memory attached. There are two primary sets of functions provided in the `sk_buff` library. Firstly routines to manipulate doubly linked lists of `sk_buffs`, secondly functions for controlling the attached memory. The buffers are held on linked lists optimised for the common network operations of append to end and remove from start. As so much of the networking functionality occurs during interrupts these routines are written to be atomic. The small extra overhead this causes is well worth the pain it saves in bug hunting.

We use the list operations to manage groups of packets as they arrive from the network, and as we send them to the physical interfaces. We use the memory manipulation routines for handling the contents of packets in a standardised and efficient manner.

At its most basic level, a list of buffers is managed using functions like this:

```
void append_frame(char *buf, int len)
{
  struct sk_buff *skb=alloc_skb(len, GFP_ATOMIC);
  if(skb==NULL)
    my_dropped++;
  else
  {
    skb_put(skb,len);
    memcpy(skb->data,data,len);
    skb_append(&my_list, skb);
  }
}

void process_queue(void)
{
  struct sk_buff *skb;
  while((skb=skb_dequeue(&my_list))!=NULL)
  {
    process_data(skb);
    kfree_skb(skb, FREE_READ);
  }
}
```

These two fairly simplistic pieces of code actually demonstrate the receive packet mechanism quite accurately. The `append_frame()` function is similar to the code called from an interrupt by a device driver receiving a packet, and `process_frame()` is similar to the code called to feed data into the protocols. If you go and look in net/core/dev.c at `netif_rx()` and `net_bh()`, you will see that they manage buffers similarly. They are far more complex, as they have to feed packets to the right protocol and manage flow control, but the basic operations are the same. This is just as true if you look at buffers going from the protocol code to a user application.

The example also shows the use of one of the data control functions, `skb_put()`. Here it is used to reserve space in the buffer for the data we wish to pass down.

Let's look at `append_frame()`. The `alloc_skb()` fucntion obtains a buffer of `len` bytes (Figure 1), which consists of:

- 0 bytes of room at the head of the buffer
- 0 bytes of data, and
- `len` bytes of room at the end of the data.

The `skb_put()` function (Figure 4) grows the **data** area upwards in memory through the free space at the buffer end and thus reserves space for the `memcpy()`. Many network operations used in sending add to the start of the frame each time in order to add headers to packets, so the `skb_push()` function (Figure 5) is provided to allow you to move the start of the data frame down through memory, providing enough space has been reserved to leave room for doing this.

Immediately after a buffer has been allocated, all the available room is at the end. A further function named `skb_reserve()` (Figure 2) can be called before data is added allows you to specify that some of the room should be at the beginning. Thus, many sending routines start with something like:

```
    skb=alloc_skb(len+headspace, GFP_KERNEL);
    skb_reserve(skb, headspace);
    skb_put(skb,len);
    memcpy_fromfs(skb->data,data,len);
    pass_to_m_protocol(skb);
```

In systems such as BSD unix you don't need to know in advance how much space you will need as it uses chains of small buffers (mbufs) for its network buffers. Linux chooses to use linear buffers and save space in advance (often wasting a few bytes to allow for the worst case) because linear buffers make many other things much faster.

Now to return to the list functions. Linux provides the following operations:

- `skb_dequeue()` takes the first buffer from a list. If the list is empty a `NULL` pointer is returned. This is used to pull buffers off queues. The buffers are added with the routines `skb_queue_head()` and `skb_queue_tail()`.
- `skb_queue_head()` places a buffer at the start of a list. As with all the list operations, it is atomic.
- `skb_queue_tail()` places a buffer at the end of a list, which is the most commonly used function. Almost all the queues are handled with one set of routines queueing data with this function and another set removing items from the same queues with `skb_dequeue()`.

- `skb_unlink()` removes a buffer from whatever list it was on. The buffer is not freed, merely removed from the list. To make some operations easier, you need not know what list the buffer is on, and you can always call `skb_unlink()` on a buffer which is not in a list. This enables network code to pull a buffer out of use even when the network protocol has no idea who is currently using it. A seperate locking mechanism is provided so device drivers do not find someone removing a buffer they are using at that moment.
- Some more complex protocols like TCP keep frames in order and re-order their input as data is received. Two functions, `skb_insert()` and `skb_append()`, exist to allow users to place `sk_buff`s before or after a specific buffer in a list.
- `alloc_skb()` creates a new `sk_buff` and initialises it. The returned buffer is ready to use but does assume you will fill in a few fields to indicate how the buffer should be freed. Normally this is `skb->free=1`. A buffer can be told not to be freed when `kfree_skb()` (see below) is called.
- `kfree_skb()` releases a buffer, and if `skb->sk` is set it lowers the memory use counts of the socket (`sk`). It is up tothe socket and protocol-level routines to have incremented these counts and to avoid freeing a socket with outstanding buffers. The memory counts are very important, as the kernel networking layers need to know how much memory is tied up by each connection in order to prevent remote machines or local processes from using too much memory.
- `skb_clone()` makes a copy of an `sk_buff` but does not copy the data area, which must be considered read only.
- For some things a copy of the data is needed for editing, and `skb_copy()` provides the same facilities but also copies the data (and thus has a much higher overhead).

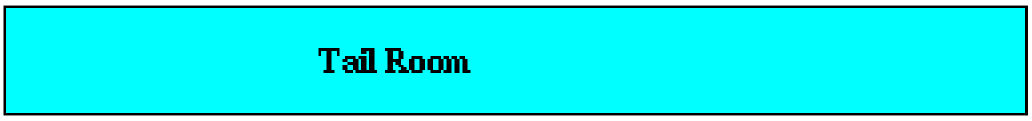


Figure 1: After alloc_skb

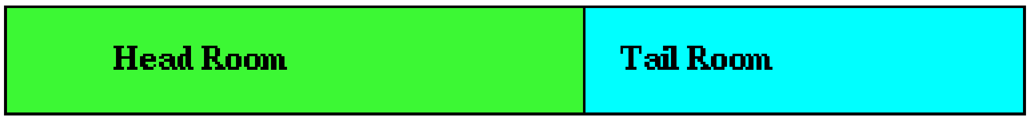


Figure 2: After skb_reserve

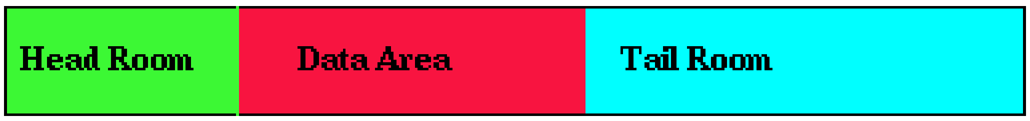


Figure 3: An sk_buff containing data

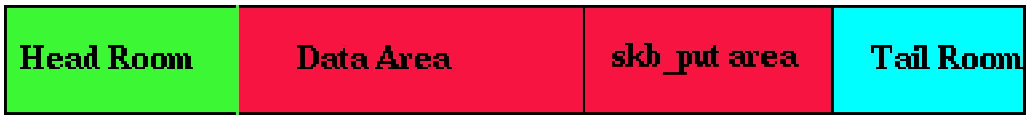


Figure 4: After skb_put has been called on the buffer

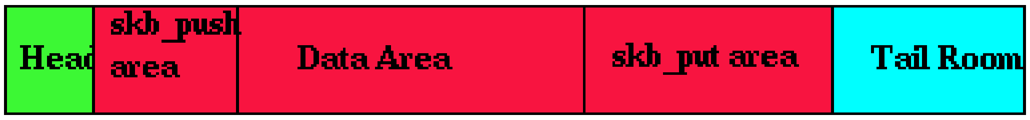


Figure 5: After an skb_push has occured on the previous buffer

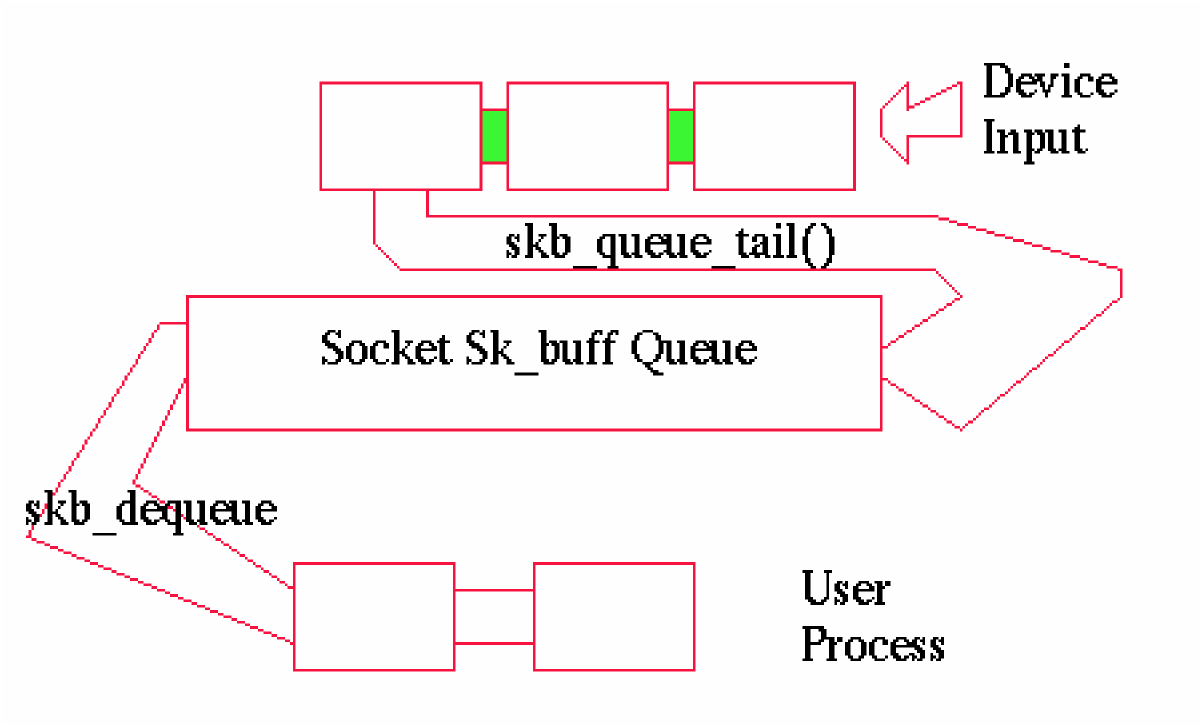


Figure 6: Network device data flow

Higher Level Support Routines

The semantics of allocating and queueing buffers for sockets also involve flow control rules and for sending a whole list of interactions with signals and optional settings such as non blocking. Two routines are designed to make this easy for most protocols.

The `sock_queue_rcv_skb()` function is used to handle incoming data flow control and is normally used in the form:

```
    sk=my_find_socket(whatever);
    if(sock_queue_rcv_skb(sk,skb)==-1)
    {
        myproto_stats.dropped++;
        kfree_skb(skb,FREE_READ);
        return;
    }
```

This function uses the socket read queue counters to prevent vast amounts of data being queued to a socket. After a limit is hit, data is discarded. It is up to the application to read fast enough, or as in TCP, for the protocol to do flow control over the network. TCP actually tells the sending machine to shut up when it can no longer queue data.

On the sending side, `sock_alloc_send_skb()` handles signal handling, the non blocking flag, and all the semantics of blocking until there is space in the send queue so you cannot tie up all of memory with data queued for a slow interface. Many protocol send routines have this function doing almost all the work:

```
    skb=sock_alloc_send_skb(sk,....)
    if(skb==NULL)
        return -err;
    skb->sk=sk;
    skb_reserve(skb, headroom);
    skb_put(skb,len);
    memcpy(skb->data, data, len);
    protocol_do_something(skb);
```

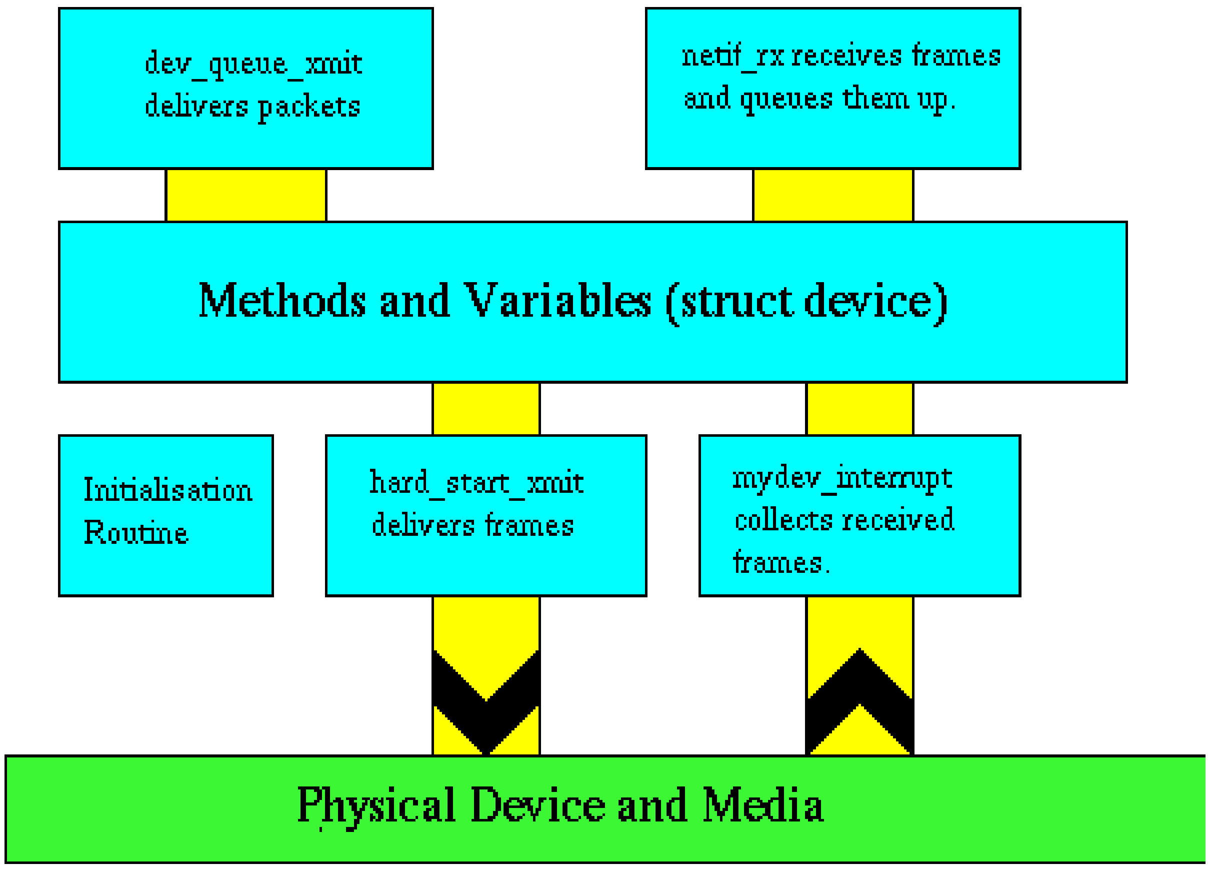
Most of this we have met before. The very important line is `skb->sk=sk`. The `sock_alloc_send_skb()` has charged the memory for the buffer to the socket. By setting `skb->sk` we tell the kernel that whoever does a `kfree_skb()` on the buffer should cause the socket to be credited the memory for the buffer. Thus when a device has sent a buffer and frees it the user will be able to send more.

Network Devices

All Linux network devices follow the same interface although many functions available in that interface will not be needed for all devices. An object oriented mentality is used and each device is an object with a series of methods that are filled into a structure. Each method is called with the device itself as the first argument. This is done to get around the lack of the C++ concept of `this` within the C language.

The file drivers/net/skeleton.c contains the skeleton of a network device driver. View or print a copy from a recent kernel and follow along throughout the rest of the article.

Basic Structure


dev_queue_xmit delivers packets
netif_rx receives frames and queues them up.
Methods and Variables (struct device)
Initialisation Routine
hard_start_xmit delivers frames
mydev_interrupt collects received frames.
Physical Device and Media

Each network device deals entirely in the transmission of network buffers from the protocols to the physical media, and in receiving and decoding the responses the hardware generates. Incoming frames are turned into network buffers, identified by protocol and delivered to `netif_rx()`. This function then passes the frames off to the protocol layer for further processing.

Each device provides a set of additional methods for the handling of stopping, starting, control and physical encapsulation of packets. These and all the other control information are collected together in the device structures that are used to manage each device.

Naming

All Linux network devices have a unique name. This is not in any way related to the file system names devices may have, and indeed network devices do not normally have a filesystem representation, although you may create a device which is tied to device drivers. Traditionally the name indicates only the type of a device rather than its maker. Multiple devices of the same type are numbered upwards from 0. Thus ethernet devices are known as ``eth0'', ``eth1'', ``eth2'' etc. The naming scheme is important as it allows users to write programs or system configuration in terms of ``an ethernet card'' rather than worrying about the manufacturer of the board and forcing reconfiguration if a board is changed.

The following names are currently used for generic devices:

eth*n*
: Ethernet controllers, both 10 and 100Mb/second

tr*n*
: Token ring devices.

sl*n*
: SLIP devices. Also used in AX.25 KISS mode.

ppp*n*
: PPP devices both asynchronous and synchronous.

plip*n*
: PLIP units. The number matches the printer port.

tunl*n*
: IPIP encapsulated tunnels

nr*n*
: NetROM virtual devices

isdn*n*
: ISDN interfaces handled by isdn4linux. (*)

dummy*n*
: Null devices

lo
: The loopback device

(*) At least one ISDN interface is an ethernet impersonator, that is the Sonix PC/Volante driver. Therefore, it uses an ``eth" device name as it behaves in all aspects as if it was ethernet rather than ISDN.

If possible, a new device should pick a name that reflects existing practice. When you are adding a whole new physical layer type you should look for other people working on such a project and use a common naming scheme.

Certain physical layers present multiple logical interfaces over one media. Both ATM and Frame Relay have this property, as does multi-drop KISS in the amateur radio environment. Under such circumstances a driver needs to exist for each active channel. The Linux networking code is structured in such a way as to make this managable without excessive additional code, and the name registration scheme allows you to create and remove interfaces almost at will as channels come into and out of existance. The proposed convention for such names is still under some discussion, as the simple scheme of ``sl0a", ``sl0b", "sl0c" works for basic devices like multidrop KISS, but does not cope with multiple frame relay connections where a virtual channel may be moved across physical boards.

Registering A Device

Each device is created by filling in a `struct device` object and passing it to the `register_netdev(struct device *)` call. This links your device structure into the kernel network device tables. As the structure you pass in is used by the kernel, you must not free this until you have unloaded the device with `void unregister_netdev(struct device *)` calls. These calls are normally done at boot time, or module load and unload.

The kernel will not object if you create multiple devices with the same name, it will break. Therefore, if your driver is a loadable module you should use the `struct device *dev_get(const char *name)` call to ensure the name is not already in use. If it is in use, you should fail or pick another name. You may not use `unregister_netdev()` to unregister the other device with the name if you discover a clash!

A typical code sequence for registration is:

```
int register_my_device(void)
{
  int i=0;
  for(i=0;i<100;i++)
  {
    sprintf(mydevice.name,"mydev%d",i);
    if(dev_get(mydevice.name)==NULL)
    {
      if(register_netdev(&mydevice)!=0)
        return -EIO;
      return 0;
    }
  }
  printk("100 mydevs loaded. Unable to load more.\n");
  return -ENFILE;
}
```

The Device Structure

All the generic information and methods for each network device are kept in the device structure. To create a device you need to fill most of these in. This section covers how they should be set up.

Naming

First, the name field holds the device name. This is a string pointer to a name in the formats discussed previously. It may also be `"    "` (four spaces), in which case the kernel will automatically assign an eth*n* name to it. This is a special feature that is best not used. After Linux 2.0, we intend to change to a simple support function of the form `dev_make_name("eth")`.

Bus Interface Parameters

The next block of parameters are used to maintain the location of a device within the device address spaces of the architecture. The `irq` field holds the interrupt (IRQ) the device is using. This is normally set at boot, or by the initialization function. If an interrupt is not used, not currently known, or not assigned, the value zero should be used. The interrupt can be set in a variety of fashions. The auto-irq facilities of the kernel may be used to probe for the device interrupt, or the interrupt may be set when loading the network module. Network drivers normally use a global int called `irq` for this so that users can load the module with `insmod mydevice irq=5` style commands. Finally, the IRQ may be set dynamically from the ifconfig command. This causes a call to your device that will be discussed later on.

The `base_addr` field is the base I/O space address the device resides at. If the device uses no I/O locations or is running on a system with no I/O space concept this field should be zero. When this is user settable, it is normally set by a global variable called `io`. The interface I/O address may also be set with ifconfig.

Two hardware shared memory ranges are defined for things like ISA bus shared memory ethernet cards. For current purposes, the `rmem_start` and `rmem_end` fields are obsolete and should be loaded with 0. The `mem_start` and `mem_end` addresses should be loaded with the start and end of the shared memory block used by this device. If no shared memory block is used, then the value 0 should be stored. Those devices that allow the user to specify this parameter use a global variable called `mem` to set the memory base, and set the `mem_end` appropriately themselves.

The `dma` variable holds the DMA channel in use by the device. Linux allows DMA (like interrupts) to be automatically probed. If no DMA channel is used, or the DMA channel is not yet set, the value 0 is used. This may have to change, since the latest PC boards allow ISA bus DMA channel 0 to be used by hardware boards and do not just tie it to memory refresh. If the user can set the DMA channel the global variable `dma` is used.

It is important to realise that the physical information is provided for control and user viewing (as well as the driver's internal functions), and does not register these areas to prevent them being reused. Thus the device driver must also allocate and register the I/O, DMA and interrupt lines it wishes to use, using the same kernel functions as any other device driver. [See the recent Kernel Korner articles on writing a character device driver in issues 23, 24, 25, 26, and 28 of Linux Journal.]

The `if_port` field holds the physical media type for multi-media devices such as combo ethernet boards.

Protocol Layer Variables

In order for the network protocol layers to perform in a sensible manner, the device has to provide a set of capability flags and variables. These are also maintained in the device structure.

The `mtu` is the largest payload that can be sent over this interface (that is, the largest packet size not including any bottom layer headers that the device itself will provide). This is used by the protocol layers such as IP to select suitable packet sizes to send. There are minimums imposed by each protocol. A device is not usable for IPX without a 576 byte frame size or higher. IP needs at least 72 bytes, and does not perform sensibly below about 200 bytes. It is up to the protocol layers to decide whether to co-operate with your device.

The `family` is always set to `AF_INET` and indicates the protocol family the device is using. Linux allows a device to be using multiple protocol families at once, and maintains this information solely to look more like the standard BSD networking API.

The interface hardware type (type) field is taken from a table of physical media types. The values used by the ARP protocol (see RFC1700) are used for those media supporting ARP and additional values are assigned for other physical layers. New values are added when neccessary both to the kernel and to **net-tools** which is the package containing programs like **ifconfig** that need to be able to decode this field. The fields defined as of Linux pre2.0.5 are:
From RFC1700:

`ARPHRD_NETROM`
: NET/ROM(tm) devices.

`ARPHRD_ETHER`
: 10 and 100Mbit/second ethernet.

`ARPHRD_EETHER`
: Experimental Ethernet (not used).

`ARPHRD_AX25`
: AX.25 level 2 interfaces.

`ARPHRD_PRONET`
: PROnet token ring (not used).

`ARPHRD_CHAOS`
: ChaosNET (not used).

`ARPHRD_IEE802`
: 802.2 networks notably token ring.

`ARPHRD_ARCNET`
: ARCnet interfaces.

`ARPHRD_DLCI`
: Frame Relay DLCI.

Defined by Linux:

`ARPHRD_SLIP`
: Serial Line IP protocol

`ARPHRD_CSLIP`
: SLIP with VJ header compression

`ARPHRD_SLIP6`
: 6bit encoded SLIP

`ARPHRD_CSLIP6`
: 6bit encoded header compressed SLIP

`ARPHRD_ADAPT`
: SLIP interface in adaptive mode

`ARPHRD_PPP`
: PPP interfaces (async and sync)

`ARPHRD_TUNNEL`
: IPIP tunnels

`ARPHRD_TUNNEL6`

IPv6 over IP tunnels

`ARPHRD_FRAD`
: Frame Relay Access Device.

`ARPHRD_SKIP`
: SKIP encryption tunnel.

`ARPHRD_LOOPBACK`
: Loopback device.

`ARPHRD_LOCALTLK`
: Localtalk apple networking device.

`ARPHRD_METRICOM`
: Metricom Radio Network.

Those interfaces marked unused are defined types but without any current support on the existing net-tools. The Linux kernel provides additional generic support routines for devices using ethernet and token ring.

The `pa_addr` field is used to hold the IP address when the interface is up. Interfaces should start down with this variable clear. `pa_brdaddr` is used to hold the configured broadcast address, `pa_dstaddr` the target of a point to point link and `pa_mask` the IP netmask of the interface. All of these can be initialised to zero. The `pa_alen` field holds the length of an address (in our case an IP address), this should be initialised to 4.

Link Layer Variables

The `hard_header_len` is the number of bytes the device desires at the start of a network buffer it is passed. It does not have to be the number of bytes of physical header that will be added, although this is normal. A device can use this to provide itself a scratchpad at the start of each buffer.

In the 1.2.x series kernels, the `skb->data` pointer will point to the buffer start and you must avoid sending your scratchpad yourself. This also means for devices with variable length headers you will need to allocate `max_size+1` bytes and keep a length byte at the start so you know where the header really begins (the header should be contiguous with the data). Linux 1.3.x makes life much simpler and ensures you will have at least as much room as you asked free at the start of the buffer. It is up to you to use `skb_push()` appropriately as was discussed in the section on networking buffers.

The physical media addresses (if any) are maintained in `dev_addr` and `broadcast` respectively. These are byte arrays and addresses smaller than the size of the array are stored starting from the left. The `addr_len` field is used to hold the length of a hardware address. With many media there is no hardware address, and this should be set to zero. For some other interfaces the address must be set by a user program. The ifconfig tool permits the setting of an interface hardware address. In this case it need not be set initially, but the open code should take care not to allow a device to start transmitting without an address being set.

Flags

A set of flags are used to maintain the interface properties. Some of these are ``compatibility'' items and as such not directly useful. The flags are:

`IFF_UP`
: The interface is currently active. In Linux, the `IFF_RUNNING` and `IFF_UP` flags are basically handled as a pair. They exist as two items for compatibility reasons. When an interface is not marked as `IFF_UP` it may be removed. Unlike BSD, an interface that does not have `IFF_UP` set will never receive packets.

`IFF_BROADCAST`
: The interface has broadcast capability. There will be a valid IP address stored in the device addresses.

`IFF_DEBUG`
: Available to indicate debugging is desired. Not currently used.

`IFF_LOOPBACK`
: The loopback interface (lo) is the only interface that has this flag set. Setting it on other interfaces is neither defined nor a very good idea.

`IFF_POINTOPOINT`
: The interface is a point to point link (such as SLIP or PPP). There is no broadcast capability as such. The remote point to point address in the device structure is valid. A point to point link has no netmask or broadcast normally, but this can be enabled if needed.

`IFF_NOTRAILERS`
: More of a prehistoric than a historic compatibility flag. Not used.

`IFF_RUNNING`
: See `IFF_UP`

`IFF_NOARP`
: The interface does not perform ARP queries. Such an interface must have either a static table of address conversions or no need to perform mappings. The NetROM interface is a good example of this. Here all entries are hand configured as the NetROM protocol cannot do ARP queries.

`IFF_PROMISC`
: The interface if it is possible will hear all packets on the network. This is typically used for network monitoring although it may also be used for bridging. One or two interfaces like the AX.25 interfaces are always in promiscuous mode.

`IFF_ALLMULTI`
: Receive all multicast packets. An interface that cannot perform this operation but can receive all packets will go into promiscuous mode when asked to perform this task.

`IFF_MULTICAST`
: Indicate that the interface supports multicast IP traffic. This is not the same as supporting a physical multicast. AX.25 for example supports IP multicast using physical broadcast. Point to point protocols such as SLIP generally support IP multicast.

The Packet Queue

Packets are queued for an interface by the kernel protocol code. Within each device, `buffs[]` is an array of packet queues for each kernel priority level. These are maintained entirely by the kernel code, but must be initialised by the device itself on boot up. The intialisation code used is:

```
int ct=0;
while(ct<DEV_NUMBUFFS)
{
    skb_queue_head_init(&dev->buffs[ct]);
    ct++;
}
```

All other fields should be initialised to 0.

The device gets to select the queue length it wants by setting the field `dev->tx_queue_len` to the maximum number of frames the kernel should queue for the device. Typically this is around 100 for ethernet and 10 for serial lines. A device can modify this dynamically, although its effect will lag the change slightly.

Network Device Methods

Each network device has to provide a set of actual functions (methods) for the basic low level operations. It should also provide a set of support functions that interface the protocol layer to the protocol requirements of the link layer it is providing.

Setup

The init method is called when the device is initialised and registered with the system. It should perform any low level verification and checking needed, and return an error code if the device is not present, areas cannot be registered or it is otherwise unable to proceed. If the init method returns an error the `register_netdev()` call returns the error code and the device is not created.

Frame Transmission

All devices must provide a transmit function. It is possible for a device to exist that cannot transmit. In this case the device needs a transmit function that simply frees the buffer it is passed. The dummy device has exactly this functionality on transmit.

The `dev->hard_start_xmit()` function is called and provides the driver with its own device pointer and network buffer (an `sk_buff`) to transmit. If your device is unable to accept the buffer, it should return 1 and set `dev->tbusy` to a non-zero value. This will queue the buffer and it may be retried again later, although there is no guarantee that the buffer will be retried. If the protocol layer decides to free the buffer the driver has rejected, then it will not be offered back to the device. If the device knows the buffer cannot be transmitted in the near future, for example due to bad congestion, it can call `dev_kfree_skb()` to dump the buffer and return 0 indicating the buffer is processed.

If there is room the buffer should be processed. The buffer handed down already contains all the headers, including link layer headers, neccessary and need only be actually loaded into the hardware for transmission. In addition, the buffer is locked. This means that the device driver has absolute ownership of the buffer until it chooses to relinquish it. The contents of an `sk_buff` remain read-only, except that you are guaranteed that the next/previous pointers are free so you can use the `sk_buff` list primitives to build internal chains of buffers.

When the buffer has been loaded into the hardware, or in the case of some DMA driven devices, when the hardware has indicated transmission complete, the driver must release the buffer. This is done by calling `dev_kfree_skb(skb, FREE_WRITE)`. As soon as this call is made, the `sk_buff` in question may spontaneously disappear and the device driver thus should not reference it again.

Frame Headers

It is neccessary for the high level protocols to append low level headers to each frame before queueing it for transmission. It is also clearly undesirable that the protocol know in advance how to append low level headers for all possible frame types. Thus the protocol layer calls down to the device with a buffer that has at least `dev->hard_header_len` bytes free at the start of the buffer. It is then up to the network device to correctly call `skb_push()` and to put the header on the packet in its `dev->hard_header()` method. Devices with no link layer header, such as SLIP, may have this method specified as NULL.

The method is invoked giving the buffer concerned, the device's own pointers, its protocol identity, pointers to the source and destination hardware addresses, and the length of the packet to be sent. As the routine may be called before the protocol layers are fully assembled, it is vital that the method use the length parameter, **not** the buffer length.

The source address may be NULL to mean ``use the default address of this device'', and the destination may be NULL to mean ``unknown''. If as a result of an unknown destination the header may not be completed, the space should be allocated and any bytes that can be filled in should be filled in. This facility is currently only used by IP when ARP processing must take place. The function must then return the negative of the bytes of header added. If the header is completely built it must return the number of bytes of header added.

When a header cannot be completed the protocol layers will attempt to resolve the address neccessary. When this occurs, the `dev->rebuild_header()` method is called with the address at which the header is located, the device in question, the destination IP address, and the network buffer pointer. If the device is able to resolve the address by whatever means available (normally ARP), then it fills in the physical address and returns 1. If the header cannot be resolved, it returns 0 and the buffer will be retried the next time the protocol layer has reason to believe resolution will be possible.

Reception

There is no receive method in a network device, because it is the device that invokes processing of such events. With a typical device, an interrupt notifies the handler that a completed packet is ready for reception. The device allocates a buffer of suitable size with `dev_alloc_skb()` and places the bytes from the hardware into the buffer. Next, the device driver analyses the frame to decide the packet type. The driver sets `skb->dev` to the device that received the frame. It sets `skb->protocol` to the protocol the frame represents so that the frame can be given to the correct protocol layer. The link layer header pointer is stored in `skb->mac.raw` and the link layer header removed with `skb_pull()` so that the protocols need not be aware of it. Finally, to keep the link and protocol isolated, the device driver must set `skb->pkt_type` to one of the following:

`PACKET_BROADCAST`
: Link layer broadcast.

`PACKET_MULTICAST`
: Link layer multicast.

`PACKET_SELF`
: Frame to us.

`PACKET_OTHERHOST`
: Frame to another single host.

This last type is normally reported as a result of an interface running in promiscuous mode.

Finally, the device driver invokes `netif_rx()` to pass the buffer up to the protocol layer. The buffer is queued for processing by the networking protocols after the interrupt handler returns. Deferring the processing in this fashion dramatically reduces the time interrupts are disabled and improves overall responsiveness. Once `netif_rx()` is called, the buffer ceases to be property of the device driver and may not be altered or referred to again.

Flow control on received packets is applied at two levels by the protocols. Firstly a maximum amount of data may be outstanding for `netif_rx()` to process. Secondly each socket on the system has a queue which limits the amount of pending data. Thus all flow control is applied by the protocol layers. On the transmit side a per device variable `dev->tx_queue_len` is used as a queue length limiter. The size of the queue is normally 100 frames, which is enough that the queue will be kept well filled when sending a lot of data over fast links. On a slow link such as slip link, the queue is normally set to about 10 frames, as sending even 10 frames is several seconds of queued data.

One piece of magic that is done for reception with most existing device, and one you should implement if possible, is to reserve the neccessary bytes at the head of the buffer to land the IP header on a long word boundary. The existing ethernet drivers thus do:

```
skb=dev_alloc_skb(length+2);
if(skb==NULL)
    return;
skb_reserve(skb,2);
/* then 14 bytes of ethernet hardware header */
```

to align IP headers on a 16 byte boundary, which is also the start of a cache line and helps give performance improvments. On the Sparc or DEC Alpha these improvements are very noticable.

Optional Functionality

Each device has the option of providing additional functions and facilities to the protocol layers. Not implementing these functions will cause a degradation in service available via the interface but not prevent operation. These operations split into two categories--configuration and activation/shutdown.

Activation And Shutdown

When a device is activated (that is, the flag `IFF_UP` is set) the `dev->open()` method is invoked if the device has provided one. This permits the device to take any action such as enabling the interface that are needed when the interface is to be used. An error return from this function causes the device to stay down and causes the user request to activate the device to fail with the error returned by `dev->open()`

The second use of this function is with any device loaded as a module. Here it is neccessary to prevent a device being unloaded while it is open. Thus the `MOD_INC_USE_COUNT` macro must be used within the open method.

The `dev->close()` method is invoked when the device is configured down and should shut off the hardware in such a way as to minimise machine load (for example by disabling the interface or its ability to generate interrupts). It can also be used to allow a module device to be unloaded now that it is down. The rest of the kernel is structured in such a way that when a device is closed, all references to it by pointer are removed. This ensures that the device may safely be unloaded from a running system. The close method is not permitted to fail.

Configuration And Statistics

A set of functions provide the ability to query and to set operating parameters. The first and most basic of these is a `get_stats` routine which when called returns a struct `enet_statistics` block for the interface. This allows user programs such as ifconfig to see the loading on the interface and any problem frames logged. Not providing this will lead to no statistics being available.

The `dev->set_mac_address()` function is called whenever a superuser process issues an ioctl of type `SIOCSIFHWADDR` to change the physical address of a device. For many devices this is not meaningful and for others not supported. If so leave this functiom pointer as `NULL`. Some devices can only perform a physical address change if the interface is taken down. For these check `IFF_UP` and if set then return `-EBUSY`.

The `dev->set_config()` function is called by the `SIOCSIFMAP` function when a user enters a command like `ifconfig eth0 irq 11`. It passes an `ifmap` structure containing the desired I/O and other interface parameters. For most interfaces this is not useful and you can return NULL.

Finally, the `dev->do_ioctl()` call is invoked whenever an ioctl in the range `SIOCDEVPRIVATE` to `SIOCDEVPRIVATE+15` is used on your interface. All these ioctl calls take a struct `ifreq`. This is copied into kernel space before your handler is called and copied back at the end. For maximum flexibility any user may make these calls and it is up to your code to check for superuser status when appropriate. For example the PLIP driver uses these to set parallel port time out speeds to allow a user to tune the plip device for their machine.

Multicasting

Certain physical media types such as ethernet support multicast frames at the physical layer. A multicast frame is heard by a group, but not all, hosts on the network, rather than going from one host to another.

The capabilities of ethernet cards are fairly variable. Most fall into one of three categories:

1. No multicast filters. The card either receives all multicasts or none of them. Such cards can be a nuisance on a network with a lot of multicast traffic such as group video conferences.
2. Hash filters. A table is loaded onto the card giving a mask of entries that we wish to hear multicast for. This filters out some of the unwanted multicasts but not all.
3. Perfect filters. Most cards that support perfect filters combine this option with 1 or 2 above. This is done because the perfect filter often has a length limit of 8 or 16 entries.

It is especially important that ethernet interfaces are programmed to support multicasting. Several ethernet protocols (notably Appletalk and IP multicast) rely on ethernet multicasting. Fortunately, most of the work is done by the kernel for you (see net/core/dev_mcast.c).

The kernel support code maintains lists of physical addresses your interface should be allowing for multicast. The device driver may return frames matching more than the requested list of multicasts if it is not able to do perfect filtering.

Whenever the list of multicast addresses changes the device drivers `dev->set_multicast_list()` function is invoked. The driver can then reload its physical tables. Typically this looks something like:

```
if(dev->flags&IFF_PROMISC)
    SetToHearAllPackets();
else if(dev->flags&IFF_ALLMULTI)
    SetToHearAllMulticasts();
else
{
    if(dev->mc_count<16)
    {
        LoadAddressList(dev->mc_list);
        SetToHearList();
    }
    else
```

```
        SetToHearAllMulticasts();
}
```

There are a small number of cards that can only do unicast or promiscuous mode. In this case the driver, when presented with a request for multicasts has to go promiscuous. If this is done, the driver must itself also set the `IFF_PROMISC` flag in `dev->flags`.

In order to aid driver writer the multicast list is kept valid at all times. This simplifies many drivers, as a reset from error condition in a driver often has to reload the multicast address lists.

Ethernet Support Routines

Ethernet is probably the most common physical interface type that is handled. The kernel provides a set of general purpose ethernet support routines that such drivers can use.

`eth_header()` is the standard ethernet handler for the `dev->hard_header` routine, and can be used in any ethernet driver. Combined with `eth_rebuild_header()` for the rebuild routine it provides all the ARP lookup required to put ethernet headers on IP packets.

The `eth_type_trans()` routine expects to be fed a raw ethernet packet. It analyses the headers and sets `skb->pkt_type` and `skb->mac` itself as well as returning the suggested value for `skb->protocol`. This routine is normally called from the ethernet driver receive interrupt handler to classify packets.

`eth_copy_and_sum()`, the final ethernet support routine, is quite internally complex but offers significant performance improvements for memory mapped cards. It provides the support to copy and checksum data from the card into an `sk_buff` in a single pass. This single pass through memory almost eliminates the cost of checksum computation when used and can really help IP throughput.

> *Alan Cox has been working on Linux since version 0.95, when he installed it in order to do further work on the AberMUD game. He now manages the Linux Networking, SMP, and Linux/8086 projects and hasn't done any work on AberMUD since November 1993.*

Messages

2. Question on alloc_skb() *by Joern Wohlrab*
    1. Re: Question on alloc_skb() *by Erik Petersen*
1. Question on network interfaces *by Vijay Gupta*
    2. Re: Question on network interfaces *by Pedro Roque*
    1. Untitled
        1. Finding net info *by Alan Cox*

Question on alloc_skb()

Forum: Network Buffers And Memory Management
Keywords: network interface
Date: Mon, 31 Mar 1997 18:16:47 GMT
From: Joern Wohlrab <*unknown*>

```
Hi,
  As I read in that introduction a network device driver
has to alloc a sk_buff in it's ISR and this has to happen
atomically, isn't it? Well my experiences are that unfortunately
it often happens that alloc_skb returns NULL. So my idea was
I alloc a few sk_buff's (with GFP_KERNEL flag) in the device
drivers open function. The device driver would organize these
sk_buff's as ring. Else the device driver must forbid the
higher layer to free the sk_buff's. Is this possible just
by setting the lock flag inside sk_buff?
The only problem with this scheme is when the user process
doesn't read frequently from the socket the device driver
overrides unread sk_buff data again and the packet order would
be destroyed. But let's assume we don't care. So is this plan
possible at all?
  Thank you very much.
--
Joern Wohlrab
```

Messages

1. Re: Question on alloc_skb() *by Erik Petersen*

Re: Question on alloc_skb()

Forum: Network Buffers And Memory Management
Re: Question on alloc_skb() (Joern Wohlrab)
Keywords: network interface
Date: Tue, 08 Apr 1997 04:42:56 GMT
From: Erik Petersen <erik@spellcast.com>

Hmm.... I'm not sure why you would want to do this personally. If alloc_skb() returns NULL, there is no memory to allocate the block you want. Normally you would report the squeeze and drop the data.

When you pass the skb to netif_rx you are esentially saying "here you go". You cant expect to reclaim the buffer as it will eventually be freed.

If you must make a best effort to deliver the data regardless of the memory situation at the time the data is received (the interrupt handler), I would create an skb list when the driver loads. Then during the interrupt, try to alloc_skb and if it fails, stuff the data in one of the pre-alloced buffers. Then the next time an interrupt occurs, try to replenish the buffer pool.

If you get to a point where both the alloc_skb fails and the buffer pool is empty, you're pretty much screwed anyways.

This would solve short-term squeeze situations but if you are that tight for memory, you might want to just printk a message saying "Get more memory cheapskate" or words to that effect.

If you're smart about it, you could balance the buffer pool at interrupt time to ensure you have enough to do the job. If you're using a good number of the buffers continuously, you might want to dynamically increase the number of buffers in the pool. If you aren't, you could reduce the number dynamically.

Just a thought.

Question on network interfaces

Forum: Network Buffers And Memory Management
Keywords: network interface
Date: Sun, 19 May 1996 17:46:31 GMT
From: Vijay Gupta <vijay@crhc.uiuc.edu>

Hi,

```
        I am working on some networking code at the kernel level
using version 1.3.71. I had the following problems :
```

How to find :

(1) the interfaces which are available :

```
        (e.g. in the case of a router, you will have many
         interfaces, one interface for each of its IP addresses).

        Many times you have both a SLIP as well as an Ethernet
        interface to your router computer.
```

(2) the status of the interface :

```
        whether the interface is up or down.
```

There is a structure called "struct ifnet" which is used in include/linux/route.h. struct ifnet has information like the name of the interface (e.g. le0 or sl0), as well as the status of the interface (whether up or down).

But I could not find the definition of this structure anywhere in version 1.3.71.

Older versions of linux had the definition of this structure available (struct ifnet also occurs in BSD code). But now I am unable to find its definition or use. Is there a substitute for that structure ?

If there is no substitute, is it the case that the information about the available interfaces cannot be obtained ?

```
Thank you very much,
        Vijay Gupta
        (Email : vijay@crhc.uiuc.edu)
```

Messages

2. Re: Question on network interfaces *by* *Pedro Roque*
1. Untitled
    1. Finding net info *by* *Alan Cox*

Re: Question on network interfaces

Forum: Network Buffers And Memory Management
Re: Question on network interfaces (Vijay Gupta)
Keywords: network interface
Date: Wed, 28 Aug 1996 15:33:15 GMT
From: Pedro Roque <roque@di.fc.ul.pt>

```
If you want to scan the interface list from kernel space you can do something like:

{
        struct device *dev;

        for (dev = dev_base; dev != NULL; dev = dev->next)
        {
                /* your code here */
                /* example */

                if (dev->family == AF_INET)
                {
                        /* this is an inet device */
                }

                if (dev->type == ARPHRD_ETHER)
                {
                        /* this is ethernet */
                }
        }
}

./Pedro.
```

Untitled

Forum: Network Buffers And Memory Management
Re: Question on network interfaces (Vijay Gupta)
Keywords: network interface
Date: Tue, 21 May 1996 23:00:03 GMT
From: *<unknown>*

I more or less managed to get the answers to the above questions by winding through start_kernel -> init -> ifconfig -> ....

```
Thanks,
        Vijay
```

Messages

1. Finding net info *by Alan Cox*

Finding net info

Forum: Network Buffers And Memory Management
Re: Question on network interfaces (Vijay Gupta)
Keywords: network interface
Date: Thu, 30 May 1996 16:12:51 GMT
From: Alan Cox <alan.cox@linux.org>

For general scanning there is both the BSD ioctl (which is a pain as you must guess the largest size), or /proc/net/dev (just cat it). For the state of an interface you use a struct ifreq filled in and do SIOCGIFFLAGS and test IFF_RUNNING and IFF_UP

Translating Addresses in Kernel Space

From a message from Linus Torvalds to the linux-kernel mailing list of 27 Sep 1996, edited.

I'll take this opportunity to tell all device driver writers about the ugly secrets of portability. Things are actually worse than just physical and virtual addresses.

The aha1542 is a bus-master device, and [a patch posted to the linux-kernel list] makes the driver give the controller the physical address of the buffers, which is correct on x86, because all bus master devices see the physical memory mappings directly.

However, on many setups, there are actually *three* different ways of looking at memory addresses, and in this case we actually want the third, the so-called "bus address".

Essentially, the three ways of addressing memory are (this is "real memory", i.e. normal RAM; see later about other details):

- CPU untranslated. This is the "physical" address, ie physical address 0 is what the CPU sees when it drives zeroes on the memory bus.
- CPU translated address. This is the "virtual" address, and is completely internal to the CPU itself with the CPU doing the appropriate translations into "CPU untranslated".
- Bus address. This is the address of memory as seen by OTHER devices, not the CPU. Now, in theory there could be many different bus addresses, with each device seeing memory in some device-specific way, but happily most hardware designers aren't actually actively trying to make things any more complex than necessary, so you can assume that all external hardware sees the memory the same way.

Now, on normal PC's, the bus address is exactly the same as the physical address, and things are very simple indeed. However, they are that simple because the memory and the devices share the same address space, and that is not generally necessarily true on other PCI/ISA setups.

Now, just as an example, on the PReP (PowerPC Reference Platform), the CPU sees a memory map something like this (this is from memory):

0-2GB
: "real memory"

2GB-3GB
: "system IO" (ie inb/out type accesses on x86)

3GB-4GB
: "IO memory" (ie shared memory over the IO bus)

Now, that looks simple enough. However, when you look at the same thing from the viewpoint of the devices, you have the reverse, and the physical memory address 0 actually shows up as address 2GB for any IO master.

So when the CPU wants any bus master to write to physical memory 0, it has to give the master address `0x80000000` as the memory address.

So, for example, depending on how the kernel is actually mapped on the PPC, you can end up with a setup like this:

physical address:
    0
virtual address:
    0xC0000000
bus address:
    0x80000000

where all the addresses actually point to the same thing, it's just seen through different translations.

Similarly, on the alpha, the normal translation is

physical address:
    0
virtual address:
    0xfffffc0000000000
bus address:
    0x40000000

(but there are also alpha's where the physical address and the bus address are the same).

Anyway, the way to look up all these translations, you do:

```
        #include <asm/io.h>

        phys_addr = virt_to_phys(virt_addr);
        virt_addr = phys_to_virt(phys_addr);
         bus_addr = virt_to_bus(virt_addr);
        virt_addr = bus_to_virt(bus_addr);
```

Now, when do you need these?

You want the *virtual* address when you are actually going to access that pointer from the kernel. So you can have something like this (from the aha1542 driver):

```
        /*
         * this is the hardware "mailbox" we use to communicate with
         * the controller. The controller sees this directly.
         */
        struct mailbox {
                __u32 status;
                __u32 bufstart;
                __u32 buflen;
                ..
        } mbox;

                unsigned char * retbuffer;

                /* get the address from the controller */
                retbuffer = bus_to_virt(mbox.bufstart);
                switch (retbuffer[0]) {
                        case STATUS_OK:
                                ...
```

On the other hand, you want the bus address when you have a buffer that you want to give to the controller:

```
        /* ask the controller to read the sense status into "sense_buffer" */
        mbox.bufstart = virt_to_bus(&sense_buffer);
        mbox.buflen = sizeof(sense_buffer);
        mbox.status = 0;
        notify_controller(&mbox);
```

And you generally *never* want to use the physical address, because you can't use that from the CPU (the CPU only uses translated virtual addresses), and you can't use it from the bus master.

So why do we care about the physical address at all? We do need the physical address in some cases, it's just not very often in normal code. The physical address is needed if you use memory mappings, for example, because the `remap_page_range()` mm function wants the physical address of the memory to be remapped (the memory management layer doesn't know about devices outside the CPU, so it shouldn't need to know about "bus addresses" etc).

NOTE NOTE NOTE! The above is only one part of the whole equation. The above only talks about "real memory", i.e. CPU memory, i.e. RAM.

There is a completely different type of memory too, and that's the "shared memory" on the PCI or ISA bus. That's generally not RAM (although in the case of a video graphics card it can be normal DRAM that is just used for a frame buffer), but can be things like a packet buffer in a network card etc.

This memory is called "PCI memory" or "shared memory" or "IO memory" or whatever, and there is only one way to access it: the `readb`/`writeb` and related functions. You should never take the address of such memory, because there is really nothing you can do with such an address: it's not conceptually in the same memory space as "real memory" at all, so you cannot just dereference a pointer. (Sadly, on x86 it *is* in the same memory space, so on x86 it actually works to just deference a pointer, but it's not portable).

For such memory, you can do things like

Reading:

```
            /*
             * read first 32 bits from ISA memory at 0xC0000, aka
             * C000:0000 in DOS terms
             */
            unsigned int signature = readl(0xC0000);
```

Remapping and writing:

```
            /*
             * remap framebuffer PCI memory area at 0xFC000000,
             * size 1MB, so that we can access it: We can directly
             * access only the 640k-1MB area, so anything else
             * has to be remapped.
```

```
	 */
	char * baseptr = ioremap(0xFC000000, 1024*1024);

	/* write a 'A' to the offset 10 of the area */
	writeb('A',baseptr+10);

	/* unmap when we unload the driver */
	iounmap(baseptr);
```

Copying and clearing:

```
	/* get the 6-byte ethernet address at ISA address E000:0040 */
	memcpy_fromio(kernel_buffer, 0xE0040, 6);
	/* write a packet to the driver */
	memcpy_toio(0xE1000, skb->data, skb->len);
	/* clear the frame buffer */
	memset_io(0xA0000, 0, 0x10000);
```

Ok, that just about covers the basics of accessing IO portably. Questions? Comments? You may think that all the above is overly complex, but one day you might find yourself with a 500MHz alpha in front of you, and then you'll be happy that your driver works `;)`

Note that kernel versions 2.0.x (and earlier) mistakenly called `ioremap()` "`vremap()`". `ioremap()` is the proper name, but I didn't think straight when I wrote it originally. People who have to support both can do something like:

```
	/* support old naming sillyness */
	#if LINUX_VERSION_CODE < 0x020100
	#define ioremap vremap
	#define iounmap vfree
	#endif
```

at the top of their source files, and then they can use the right names even on 2.0.x systems.

And the above sounds worse than it really is. Most real drivers really don't do all that complex things (or rather: the complexity is not so much in the actual IO accesses as in error handling and timeouts etc). It's generally not hard to fix drivers, and in many cases the code actually looks better afterwards:

```
	unsigned long signature = *(unsigned int *) 0xC0000;
```

vs.

```
	unsigned long signature = readl(0xC0000);
```

I think the second version actually is more readable, no?

```
			Linus
```

Kernel-Level Exception Handling

From a message from Joerg Pommnitz to the linux-kernel mailing list of 11 Nov 1996, edited.

According to Linus Torvalds:

People interested in low-level scary stuff should take a look at the uaccess.h files for x86 or alpha, and be ready to spend some time just figuring out what it all does ;)

I am, and I did.

Kernel-level exception handling in Linux 2.1.8

When a process runs in kernel mode, it often has to access user mode memory whose address has been passed by an untrusted program. To protect itself, the kernel has to verify this address.

In older versions of Linux, this was done with the

```
int verify_area(int type, const void * addr, unsigned long size)
```

function.

This function verified, that the memory area starting at address `addr` and of size `size` was accessible for the operation specified in `type` (read or write). To do this, `verify_read` had to look up the virtual memory area (`vma`) that contained the address `addr`. In the normal case (correctly working program), this test was successful. It only failed for the (hopefully) rare, buggy program. In some kernel profiling tests, this normally unneeded verification used up a considerable amount of time.

To overcome this situation, Linus decided to let the virtual memory hardware present in every Linux capable CPU handle this test.

How does this work?

Whenever the kernel tries to access an address that is currently not accessible, the CPU generates a page fault exception and calls the page fault handler

```
void do_page_fault(struct pt_regs *regs, unsigned long error_code)
```

in arch/i386/mm/fault.c. The parameters on the stack are set up by the low level assembly glue in arch/i386/kernel/entry.S. The parameter `regs` is a pointer to the saved registers on the stack, `error_code` contains a reason code for the exception.

`do_page_fault` first obtains the unaccessible address from the CPU control register CR2. If the address is within the virtual address space of the process, the fault probably occured, because the page was not swapped in, write protected or something similiar. However, we are interested in the other case: the address is not valid, there is no `vma` that contains this address. In this case, the kernel jumps to the `bad_area` label.

There it uses the address of the instruction that caused the exception (i.e. `regs->eip`) to find an address where the excecution can continue (fixup). If this search is successful, the fault handler modifies the return address (again `regs->eip`) and returns. The execution will continue at the address in fixup.

Where does fixup point to?

Since we jump to the the contents of fixup, fixup obviously points to executable code. This code is hidden inside the user access macros. I have picked the `get_user` macro defined in include/asm/uaccess.h as an example. The definition is somewhat hard to follow, so lets peek at the code generated by the preprocessor and the compiler. I selected the `get_user` call in drivers/char/console.c for a detailed examination.

The original code in console.c line 1405:

```
        get_user(c, buf);
```

The preprocessor output (edited to become somewhat readable):

```
(
  {
    long __gu_err = - 14 , __gu_val = 0;
    const __typeof__(*( (  buf ) )) *__gu_addr = ((buf));
    if (((((0 + current_set[0])->tss.segment) == 0x18 )  ||
       (((sizeof(*(buf))) <= 0xC0000000UL) &&
       ((unsigned long)(__gu_addr ) <= 0xC0000000UL - (sizeof(*(buf)))))))
      do {
        __gu_err  = 0;
        switch ((sizeof(*(buf)))) {
          case 1:
            __asm__ __volatile__(
              "1:      mov" "b" " %2,%" "b" "1\n"
              "2:\n"
              ".section .fixup,\"ax\"\n"
              "3:      movl %3,%0\n"
              "        xor" "b" " %" "b" "1,%" "b" "1\n"
              "        jmp 2b\n"
              ".section __ex_table,\"a\"\n"
              "        .align 4\n"
              "        .long 1b,3b\n"
              ".text"        : "=r"(__gu_err), "=q" (__gu_val): "m"((*(struct
__large_struct *)
                            (   __gu_addr   )) ), "i"(- 14 ), "0"(  __gu_err  )) ;
              break;
          case 2:
            __asm__ __volatile__(
              "1:      mov" "w" " %2,%" "w" "1\n"
              "2:\n"
              ".section .fixup,\"ax\"\n"
              "3:      movl %3,%0\n"
              "        xor" "w" " %" "w" "1,%" "w" "1\n"
              "        jmp 2b\n"
              ".section __ex_table,\"a\"\n"
              "        .align 4\n"
              "        .long 1b,3b\n"
              ".text"        : "=r"(__gu_err), "=r" (__gu_val) : "m"((*(struct
__large_struct *)
                            (   __gu_addr   )) ), "i"(- 14 ), "0"(  __gu_err  ));
              break;
          case 4:
            __asm__ __volatile__(
              "1:      mov" "l" " %2,%" "" "1\n"
              "2:\n"
              ".section .fixup,\"ax\"\n"
              "3:      movl %3,%0\n"
              "        xor" "l" " %" "" "1,%" "" "1\n"
              "        jmp 2b\n"
              ".section __ex_table,\"a\"\n"
              "        .align 4\n"        "        .long 1b,3b\n"
              ".text"        : "=r"(__gu_err), "=r" (__gu_val) : "m"((*(struct
__large_struct *)
                             (   __gu_addr   )) ), "i"(- 14 ), "0"(__gu_err));
              break;
          default:
            (__gu_val) = __get_user_bad();
        }
      } while (0) ;
    ((c)) = (__typeof__(*((buf))))__gu_val;
    __gu_err;
  }
);
```

WOW! Black GCC/assembly magic. This is impossible to follow, so lets see what code gcc generates:

```
        xorl %edx,%edx
        movl current_set,%eax
        cmpl $24,788(%eax)
        je .L1424
        cmpl $-1073741825,64(%esp)
        ja .L1423
.L1424:
        movl %edx,%eax
        movl 64(%esp),%ebx
#APP
1:      movb (%ebx),%dl  /* this is the actual user access */
2:
.section .fixup,"ax"
3:      movl $-14,%eax
        xorb %dl,%dl
        jmp 2b
.section __ex_table,"a"
        .align 4
        .long 1b,3b
.text
#NO_APP
.L1423:
        movzbl %dl,%esi
```

The optimizer does a good job and gives us something we can actually understand. Can we? The actual user access is quite obvious. Thanks to the unified address space we can just access the address in user memory. But what does the `.section` stuff do?

To understand this we have to look at the final kernel:

```
$ objdump --section-headers vmlinux

vmlinux:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00098f40  c0100000  c0100000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .fixup        000016bc  c0198f40  c0198f40  00099f40  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .rodata       0000f127  c019a5fc  c019a5fc  0009b5fc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 __ex_table    000015c0  c01a9724  c01a9724  000aa724  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         0000ea58  c01abcf0  c01abcf0  000abcf0  2**4
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          00018e21  c01ba748  c01ba748  000ba748  2**2
                  ALLOC
  6 .comment      00000ec4  00000000  00000000  000ba748  2**0
                  CONTENTS, READONLY
  7 .note         00001068  00000ec4  00000ec4  000bb60c  2**0
                  CONTENTS, READONLY
```

There are obviously 2 non standard ELF sections in the generated object file. But first we want to find out what happened to our code in the final kernel executable:

```
$ objdump --disassemble --section=.text vmlinux

c017e785 <do_con_write+c1> xorl   %edx,%edx
c017e787 <do_con_write+c3> movl   0xc01c7bec,%eax
c017e78c <do_con_write+c8> cmpl   $0x18,0x314(%eax)
c017e793 <do_con_write+cf> je     c017e79f <do_con_write+db>
c017e795 <do_con_write+d1> cmpl   $0xbfffffff,0x40(%esp,1)
c017e79d <do_con_write+d9> ja     c017e7a7 <do_con_write+e3>
c017e79f <do_con_write+db> movl   %edx,%eax
c017e7a1 <do_con_write+dd> movl   0x40(%esp,1),%ebx
c017e7a5 <do_con_write+e1> movb   (%ebx),%dl
c017e7a7 <do_con_write+e3> movzbl %dl,%esi
```

The whole user memory access is reduced to 10 x86 machine instructions. The instructions bracketed in the `.section` directives are not longer in the normal execution path. They are located in a different section of the executable file:

```
$ objdump --disassemble --section=.fixup vmlinux

c0199ff5 <.fixup+10b5> movl   $0xfffffff2,%eax
c0199ffa <.fixup+10ba> xorb   %dl,%dl
c0199ffc <.fixup+10bc> jmp    c017e7a7 <do_con_write+e3>
```

And finally:

```
$ objdump --full-contents --section=__ex_table vmlinux

c01aa7c4 93c017c0 e09f19c0 97c017c0 99c017c0  ................
c01aa7d4 f6c217c0 e99f19c0 a5e717c0 f59f19c0  ................
c01aa7e4 080a18c0 01a019c0 0a0a18c0 04a019c0  ................
```

or in human readable byte order:

```
c01aa7c4 c017c093 c0199fe0 c017c097 c017c099  ................
c01aa7d4 c017c2f6 c0199fe9 c017e7a5 c0199ff5  ................
                           this is the interesting part!
c01aa7e4 c0180a08 c019a001 c0180a0a c019a004  ................
```

What happened? The assembly directives

```
.section .fixup,"ax"
.section __ex_table,"a"
```

told the assembler to move the following code to the specified sections in the ELF object file. So the instructions

```
3:      movl $-14,%eax
        xorb %dl,%dl
        jmp 2b
```

ended up in the `.fixup` section of the object file and the addresses

```
        .long 1b,3b
```

ended up in the `__ex_table` section of the object file. `1b` and `3b` are local labels. The local label `1b` (`1b` stands for next label 1 backward) is the address of the instruction that might fault. In our case, the address of the label 1b is `c017e7a5`:

the original assembly code:

```
     1:      movb (%ebx),%dl
```

and linked in vmlinux:

```
     c017e7a5 <do_con_write+e1> movb   (%ebx),%dl
```

The local label 3 (backwards again) is the address of the code to handle the fault, in our case the actual value is `c0199ff5`:

the original assembly code:
    3: movl $-14,%eax
and linked in vmlinux:

```
     c0199ff5 <.fixup+10b5> movl   $0xfffffff2,%eax
```

The assembly code

```
.section __ex_table,"a"
        .align 4
        .long 1b,3b
```

becomes the value pair

```
c01aa7d4 c017c2f6 c0199fe9 c017e7a5 c0199ff5  ................
```

```
                                   ^this is ^this is
                                       1b       3b
```

`c017e7a5`,`c0199ff5` in the exception table of the kernel.

In order for the function `search_exception_table` to find the exception table in the __`ex_table` section, it uses a linker feature: whenever the linker sees a section whose entire name is a valid C identifier, it creates the symbols `__start_`*`section`* and `__stop_`*`section`* delimiting the extents of the section. So `search_exception_table` brackets its search by `__start___ex_table` and `__stop___ex_table`

Exception handling in action

So, what actually happens if a fault from kernel mode with no suitable `vma` occurs?

1. access to invalid address:

   ```
   c017e7a5 <do_con_write+e1> movb   (%ebx),%dl
   ```

2. MMU generates exception
3. CPU calls `do_page_fault`
4. `do_page_fault` calls `search_exception_table (regs->eip == c017e7a5);`
5. `search_exception_table` looks up the address `c017e7a5` in the exception table (i.e. the contents of the ELF section __`ex_table` and returns the address of the associated fault handle code `c0199ff5`.
6. `do_page_fault` modifies its own return address to point to the fault handle code and returns.
7. execution continues in the fault handling code.
8. 8a) `EAX` becomes `-EFAULT` (== -14)
   8b) `DL` becomes zero (the value we "read" from user space)
   8c) execution continues at local label 2 (address of the instruction immediately after the faulting user access).

The steps 8a to 8c in a certain way emulate the faulting instruction.

That's it, mostly. If you look at our example, you might ask why we set `EAX` to `-EFAULT` in the exception handler code. Well, the `get_user` macro actually returns a value: 0, if the user access was successful, `-EFAULT` on failure. Our original code did not test this return value, however the inline assembly code in `get_user` tries to return `-EFAULT`. GCC selected `EAX` to return this value.

```
Joerg Pommnitz    | joerg@raleigh.ibm.com | Never attribute to malloc
Mobile/Wireless   | Dept UMRA             | that which can be adequately
Tel:(919)254-6397 | Office B502/E117      | explained by stupidity.
```

DMA to user space

Forum: Device Drivers
Date: Wed, 11 Jun 1997 09:23:28 GMT
From: Marcel Boosten <Marcel.Boosten@cern.ch>

```
Hello,

I'm developing a device driver for a PCI board meant for
high performance communication. Interaction with the board
is possible via DMA. In order to get optimal performance
I need to do DMA directly to user space.

QUESTION:
How do I implement DMA to user space?

SUBQUESTIONS:
In "The Linux Kernel", David A Rusling writes the following:
"Device drivers have to be careful when using DMA. First
of all the DMA controller knows nothing of virtual memory,
it only has access to the physical memory in the system.
Therefore the memory that is being DMA'd to or from must
be a contiguous block of physical memory. This means that
you cannot DMA directly into the virtual address space of
a process. YOU CAN HOWEVER LOCK THE PROCESSES PHYSICAL
PAGES INTO MEMORY, PREVENTING THEM FROM BEING SWAPPED OUT
TO THE SWAP DEVICE DURING A DMA OPERATION. Secondly, the
DMA controller cannot access the whole of physical memory.
The DMA channel's address register represents the first 16
bits of the DMA address, the next 8 bits come from the page
register. This means that DMA requests are limited to the
bottom 16 Mbytes of memory."
[see http://www.linuxhq.com/guides/TLK/node87.html]

Reading this the following subquestions arise:
   - How can one lock specific process pages?
   - How can one obtain the physical address of
     the pages involved?
   - How can one ensure that the pages involved
     are DMA-able (below 16Mb)?
   - Is it possible to obtain a continues block of
```

```
    physical memory in user space?

Greetings,
  Marcel
```

How a device driver can driver his device

Forum: Device Drivers
Keywords: device driver
Date: Sat, 31 May 1997 07:31:17 GMT
From: Kim yeonseop <javakys@hyowon.pusan.ac.kr>

```
Hi.

I am a beginner for device driver on linux.
I wrote , 'zero.c' which is introduced to
'http://www.redhat.com/~johnsonm/devices.html',
a sample driver to test on my system .
But I don't know how to drive 'zero device' by this device driver.

Please help me.

Thanks for your response.

Kim yeonseop : javakys@hyowon.pusan.ac.kr
```

Messages

1. Untitled NEW

Untitled

Forum: Device Drivers
Re: How a device driver can driver his device (Kim yeonseop)
Keywords: device driver
Date: Thu, 05 Jun 1997 02:08:25 GMT
From: <unknown>

What do you mean to "drive the driver"? more clearly.

memcpy error?

Forum: Device Drivers
Keywords: memcpy verify_area
Date: Wed, 21 May 1997 14:33:34 GMT
From: Edgar Vonk <edgar@it.et.tudelft.nl>

I am using memcpy in a device driver to copy data between to buffers in kernel space (one is a DMA buffer) and I keep getting segmentation faults I can't explain.

I changed the driver since and now it copies the DMA buffer directly into user space with memcpy_tofs (and verify_area) and this seems to work just fine.

Anyone know why? Does this have to do with the memcpy faults under heavy system load? I saw a discussion and a kernel patch about this somewhere.

thanks,

(running i586-linux-2.0.30-RedHat4.1)

Unable to handle kernel paging request - error

Forum: Device Drivers
Keywords: kernel paging
Date: Wed, 14 May 1997 16:15:40 GMT
From: Edgar Vonk <edgar@it.et.tudelft.nl>

Hai,

just a simple question. What does the "Unable to handle kernel paging request at virtual address ..." usually indicate?

Does this mean a memory allocation problem, or just a memory addressing problem. Also, why does it come back with a virtual address and not a physical one? Does this mean it is doing something in user space?

I am writing a device driver for a Data Acquisition Card, but haven't got a clue what the bug in my code is.

cheers,

(Running i586-Linux-2.0.30-RedHat4.1)

_syscallX() Macros

Forum: Device Drivers
Date: Wed, 26 Mar 1997 23:07:31 GMT
From: Tom Howley <*unknown*>

Is it possible to use _syscallX macros in loadable device drivers. I first of all have had problems with "errno: wrong version or undefined".It seems to be defined in linux/lib/errno.c. I want to be able to use the system calls signal, getitimer and setitimer in my driver Does anybody know how I can get a _syscall() macro to work in my loadable device driver??

Any advice would be much appreciated.

Tom.

MediaMagic Sound Card DSP-16. How to run in Linux.

Forum: Device Drivers
Keywords: MediaMagic DSP16
Date: Tue, 18 Mar 1997 04:25:37 GMT
From: Robert Hinson <oppie@afn.org>

```
I am looking for a way to run the MediaMagic Sound Card DSP-16 under Linux RedHat
4.0?
I would very much appreciate it.  Or how to set it up with the current drivers.  I
know
it is SoundBlaster and SoundBlaster Pro Compatible, but I don't know how to make it
work.
I would like some help.  My e-mail address is oppie@afn.org  since I don't read this.
```

What does mark_bh() do?

Forum: Device Drivers
Keywords: network drivers interrupt mark_bh
Date: Wed, 12 Mar 1997 01:42:49 GMT
From: Erik Petersen <erik@spellcast.com>

Can someone expain when and how I use mark_bh(). I am assuming from general knowledge that it mark the end of the interrupt service routine and allows a context switch in following code.

Here is why I want to know. I have a network driver in which it would be advantagous to be able to sleep during code initiated by an interrupt. For example a piece of data is received by the device which is passed to a kernel daemon via a character device inode and a select call. I then want to wait for the daemon to respond or timeout.

The question is, if I call mark_bh(NET_BH) IMMEDIATE_BH?? before I sleep, can I sleep or do I Aiee...Killing Interrupt handler, Idle task may not sleep?

mark_bh doesn't seem to be explained anywhere but is used by many net drivers for reasons I don't understand. Is there somewhere I can look for this information?

My only obvious alternative at this point is to create a request queue of some sort and respond to activity on the character device. The problem is that I can't really continue transferring data until I get a response from the daemon.

Any thoughts?

Erik Petersen.

Messages

1. Untitled *by Praveen Dwivedi*

Untitled

Forum: Device Drivers
Re: What does mark_bh() do? (Erik Petersen)
Keywords: network drivers interrupt mark_bh
Date: Fri, 14 Mar 1997 08:28:12 GMT
From: Praveen Dwivedi <pkd@sequent.com>

```
I am not an expert on Linux kernel but here is what my
hacking wisdom says.

mark_bh marks the bottom half of some hardware interrupts.
An example would be timer interrupt which comes
100 times a second. Generally what happens is that you
do minimum stuff in actual handler and call mark_bh()
which takes care of updating lots of time related system
stuff. The reason why this is the preferred way to do
things is because you want to have actual interrupt handler
as small as possible so as to avoid losing further interrupts.

Look at the code in do_timer. It may help.

 -pkd
```

3D Acceleration

Forum: Device Drivers
Keywords: 3D acceleration driver
Date: Sat, 08 Mar 1997 18:04:25 GMT
From: <jamesbat@innotts.co.uk>

How would I go about making a driver for the Apocalypse 3D please Email reply

Device Drivers: /dev/radio...

Forum: Device Drivers
Keywords: device /dev radio
Date: Fri, 07 Mar 1997 00:56:50 GMT
From: Matthew Kirkwood <weejock@ferret.lmh.ox.ac.uk>

Hi,

I intend to write (when my radio card arrives in a couple of days) a driver for /dev/radio.

I have already obtained reasonable information for this, which is all fair enough, but I have not yet seen anything along the lines of "/dev/* device creating for the inept...". Should I create a document explaining this? (/dev/radio, as I envisage it, would be a mostly ioctl based thing, depending upon hardware support....)

Thanks, and keep up the hacking, Matthew.

Does anybody know why kernel wakes my driver up without apparant reasons?

Forum: Device Drivers
Keywords: wake_up interrupt time_out
Date: Wed, 26 Feb 1997 17:02:31 GMT
From: David van Leeuwen <david@tm.tno.nl>

Hi, i've written a device driver for a cdrom device. It's old. I know. But i keep getting compaints that it doesn't work reliably.

It used to work OK in the old 1.3.fourties. Since more modern kernel version, it tended to break more often. Read errors...

I spent days tracking down the bug, it appeared that the driver was woken without an interrupt occurring, or my own time-out causing the wake-up. I was stymied.

Now i posted a message similar to this to the kernel list half a year ago. But i wasn't capable of reading the list (sorry) because i use my e-mail address at work. Apparently, there was some short reaction that my go_to_sleep routine should do something like

```
        while(!my_interrupt_woke_me)
                sleep_on(&wait)
```

Why is this? Why does the kernel wake me up if i didn't ask for it (i.e., no interrupt occured and no time-out occurred)

I found out that the sleep_on() could immediately wakeup (i.e., not go to sleep) for many times in a row. I had to hack around by trying to go to sleep up to 100 times, but i am not charmed by the hack.

Does it have to do with the (new?) macros DEVICE_TIMEOUT and TIMEOUT_VALUE that i've _not_ defined (because i wrote it in the KHG 0.5 days...).

Thanks,

---david (david@tm.tno.nl)

Getting a DMA buffer aligned with 64k boundaries

Forum: Device Drivers
Date: Sun, 17 Nov 1996 01:25:45 GMT
From: Juan de La Figuera Bayon <juan@hobbes.fmc.uam.es>

I'm writing a device driver for a Data Translation DT2821 adquisition card. It includes DMA (and I have already worked with it under MSDOS). The polled modes for DA and AD conversion already work. But for the DMA, I need to ask for a buffer which can be up to 128k in size (ok, I usually ask for less than 256 words in my aplication). And it should be aligned with 64k boundaries. I suppose it is something pretty obvious, but it is my first try at device driver programming under Linux. Any help would be appreciated.

Hardware Interface I/O Access

Forum: Device Drivers
Keywords: I/O
Date: Mon, 07 Oct 1996 12:36:40 GMT
From: Terry Moore <tmoore@solbrn.dseg.ti.com>

```
I need to write a driver using inb() and outb().
I am struggling with compiling a simple test program
to test these fuctions.
(1.) If I use gcc -o -DMODULE -D__KERNEL__ -c myfile.c
     I do not understand how to use the resulting file
     created -DMODULE.
(2.) If I use gcc -o tst tst.c the following fail appears.
     undefined reference to __inbc
     undefeined reference to __inb
```

What I expected was a executable program to run from the command line.

Thanks Terry M. tmoore@solbrn.dseg.ti.com

Messages

1. You are somewhat confused... *by Michael K. Johnson*

You are somewhat confused...

Forum: Device Drivers
Re: Hardware Interface I/O Access (Terry Moore)
Keywords: I/O
Date: Mon, 14 Oct 1996 22:16:16 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

You've got two things mixed up--user level drivers and kernel loadable modules. An executable program is what you want, not a module, so don't define `MODULE`. Just compile your executable with `-O` and the undefined references should go away.

Is Anybody know something about SIS 496 IDE chipset?

Forum: Device Drivers
Date: Fri, 27 Sep 1996 13:13:21 GMT
From: Alexander <avenco@online.ru>

I use SIS 496 (E)IDE controler chipset. Linux 2.0.21 doesn't support it. Is anybody know about it? I need technical informationd about the chipset for writig driver. e-mail : avenco@online.ru

alex

Messages

Vertical Retrace Interrupt - I need to use it

Forum: Device Drivers
Date: Wed, 04 Sep 1996 06:39:27 GMT
From: Brynn Rogers <brynn@wwa.com>

I am writing an application that provides new images to the screen every vertical refresh. (Think of it as an animation)

As I understand it, I need to write a device driver to hook the vertical retrace interrupt (whatever interrupt your graphics card generates), and to install a new colormap so the next image is cleanly flipped in. (I don't need many colors, but I need lots of images).

I have been devouring all information (and donuts) I can get my hands on, and still am a little bit clueless as to how I should go about this. What I am really confused about is this: Should I have a device that my animation program opens and then uses ioctls to talk to, Just have the driver wake my process and signal it, or Something much better that somebody will clue me in on.

The driver only needs to know a few things, like which image planes are ready and the ID's? of the colormaps to use for which planes, and which screen or GC or whatever it needs.

Brynn

Messages

1. Your choice... *by Michael K. Johnson*

Your choice...

Forum: Device Drivers
Re: Vertical Retrace Interrupt - I need to use it (Brynn Rogers)
Date: Sun, 29 Sep 1996 20:44:18 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

> *What I am really confused about is this: Should I have a device that my animation program opens and then uses ioctls to talk to, Just have the driver wake my process and signal it, or Something much better that somebody will clue me in on.*

You are quite right that you need a device driver. If you can, I recommend avoiding using ioctls; if you can use the `write()` method to take data from the application and the `read()` method to give data back to the application (remember that those names are user-space-centric), I would recommend that you do it that way. It doesn't sound to me like a case in which `ioctl()`'s would be the cleanest solution.

help working with skb structures

Forum: Device Drivers
Date: Thu, 29 Aug 1996 15:44:32 GMT
From: arkane <cat@iol.unh.edu>

I am working on interfacing directly with the networking device drivers on my linux box. I have tracked down the functions for transmitting (dev_queue_xmit()) packets down to the driver level. What I need to do is bypass the socket interface without destroying it ... So that I can transmit my own packets or my own design down to the wire (I am using this for my job of testing new networking hardware -- RMON probes mostly) so I need to be able to create both good and bad packets with most any kind of data contained inside as RMON-2 will be able to pick apart a packet and identify its contents.

We can build the packets, but we can't get them to the wire through standard means. I think that this can be accomplished with the dev_queue_xmit() function. Question is: in the sk_buff structure what do I need to set up specifically so that dev_queue_xmit() and the driver will simply pass my data to the hardware without building the standard headers required by ethernet and other network types? I'll worry about that, and if I make a mistake I will clean up the mess. Any help is appreciated.

TIA cat@iol.unh.edu

Interrupt Sharing ?

Forum: Device Drivers
Keywords: Interrupt sharing, PCI, Plug%0Aamp;Play
Date: Tue, 11 Jun 1996 16:09:00 GMT
From: Frieder Löffler <floeff@mathematik.uni-stuttgart.de>

I wonder if interrupt sharing is an issue for the Linux kernel. I currently have a machine with 2 PCI Plug&Play devices that choose the same irq (an HP-Vectra onboard SCSI controller and a HP J2585 100-VG-AnyLan card).

It seems that there is no way to use such configurations at the moment?

Frieder

Messages

1. Interrupt sharing-possible *by Vladimir Myslik*
-> Interrupt sharing - How to do with Network Drivers? *by Frieder Löffler*
-> Interrupt sharing 101 *by Christophe Beauregard*

Interrupt sharing-possible

Forum: Device Drivers
Re: Interrupt Sharing ? (Frieder Löffler)
Keywords: Interrupt sharing, PCI, Plug%0Aamp;Play
Date: Thu, 11 Jul 1996 02:24:57 GMT
From: Vladimir Myslik <xmyslik@cslab.felk.cvut.cz>

Linux kernel has support for shared interrupt usage. It has a list of routines (func.) that are called when an HW intr arises. On the interrupt arrival, the routines in the list are circularily called in the order in which the devices ISRs were hooked onto this chain.

So, if your SCSI generates int#11 and your ethernet card the same irq, and the bus really notices CPU about them, linux should have no problems.

However, the ISA and IMHO PCI devices have problems with sharing one IRQ line per several physical cards (devices). The devices should had been designed with open collector or with 3-state IRQ lines with transition to IRQ active only during the interrupt generation(log. 0/1), instead of sitting on the irq line.

So, a user wanting to find out whether it's possible to share one irq line, should set both the cards to it, make either of them generate interrupt (packet arrival,seek on disk) and look at the /proc/interrupts statistics, whether the appropriate number incremented or not.

Got all from usenet&kernel sources, don't blame me.

Messages

1. Interrupt sharing - How to do with Network Drivers? *by Frieder Löffler*
-> Interrupt sharing 101 *by Christophe Beauregard*

Interrupt sharing - How to do with Network Drivers?

Forum: Device Drivers
Re: Interrupt Sharing ? (Frieder Löffler)
Keywords: Interrupt sharing, PCI, Plug%0Aamp;Play
Date: Thu, 11 Jul 1996 09:02:56 GMT
From: Frieder Löffler <floeff@mathematik.uni-stuttgart.de>

Hi,

you are right - as I noticed in the AM53C974 SCSI driver, some drivers seem to be designed to share interrupts. But I cannot see at the moment how I can implement interrupt sharing in the networking drivers. Maybe someone could explain how this can be done - for example by adding some lines of code to skeleton.c ?

Right now, I can't see how I am supposed to register the interrupt handler routine for the second driver.

Thanks, Frieder

Messages

1. Interrupt sharing 101 *by Christophe Beauregard*

Interrupt sharing 101

Forum: Device Drivers
Re: Interrupt Sharing ? (Frieder Löffler)
Re: Interrupt sharing - How to do with Network Drivers? (Frieder Löffler)
Keywords: Interrupt sharing, PCI, Plug%0Aamp;Play
Date: Wed, 28 Aug 1996 16:48:01 GMT
From: Christophe Beauregard <chrisb@truespectra.com>

I guess this would be a handy thing to have in the knowledge base...

The key thing to sharing an interrupt is to make sure that you have separate context information for each instance of the driver. That is, no static global variables. For most network drivers you just use the ``struct device* dev'' for the context.

Pass this to request_irq() as the last argument:

```
request_irq( irq, interrupt_handler, SA_SHIRQ, "MyDevice",
             dev );
```

Note that the SA_INTERRUPT flag is significant here, since you can't share an IRQ if one driver uses fast interrupts and the other uses slow interrupts. This is a bug, IMHO, since long chains of interrupt handlers may alter the timing such that processing is no longer ``fast''. A better behaviour would be to just implicitly change to slow interrupts when more than one device is on the IRQ (and change back when the device is released down to one fast handler, of course).

Then your interrupt handler looks something like this:

```
static void interrupt_handler( int irq, void* dev_id, ...) {
    struct device* dev = (struct device*) dev_id;

    if( dev == NULL ) {
        ASSERT( 0 ); /* stupid programmer error.  Either
                        we passed a NULL dev to request_irq
                        or someone screwed up irq.c */
    }

    /* query the device to see if it caused the interrupt */
    if( !(inb(something)&something_else) ) {
        /* nope, not us - normally we'd call this a spurious
        interrupt, but it might belong to another device. */
        return;
    }

    /* now, using the dev structure, service the interrupt */
    ...

    /* tell the hardware device we're done (IMPORTANT)
    If this isn't done, the device will continue to hold
    the IRQ line high, and we go into a nasty interrupt
    loop.  Some devices might do this implicitly in the
    interrupt processing (i.e. by emptying a buffer) */
    outb(something, something);
}
```

Because you have a separate ``struct device*'' for each instance of the card, multiple cards can share the same IRQ. Of course, they can also share the IRQ with other card, assuming they all Do The Right Thing.

You can usually modify an existing device to do shared IRQs by simply finding the part of the code where it spews out a spurious interrupt message and replacing that with a `return' statement, adding SA_SHIRQ to the request_irq call, and removing references to irq2dev_map[]. I've had no problems doing this for drivers including drivers/char/psaux.c, drivers/net/tulip.c, drivers/scsi/aicxxx7.c and most of the MCA drivers.

c.

Device Driver notification of "Linux going down"

Forum: Device Drivers
Keywords: device drivers shutdown modules init watchdog
Date: Tue, 06 Aug 1996 20:21:06 GMT
From: Stan Troeh <stan@forthrt.com>

We have written a character device driver for FORTHRIGHT's PC WATCHDOG SYSTEM. We find, however, in developing a generic "watch" application (tells the hardware that Linux is still healthy) that we can't detect when Linux is intentionally shutting down. If Linux succeeds in going down and back up within the default 2 minute window, everything is transparent and no problem occurs. However we find that at times when the tolerance is set tighter or the user delays making a LILO selection, etc., that the hardware performs a physical PC Reset while Linux is doing coming up (after a shutdown -r).

Is there a "shutting down" call made to the drivers? We have not found one, but have found a place where it could be added in ReadItab() or InitMain() of init.c. But that doesn't seem closely related to device driver module management.

Suggestions for better ways to package the driver are welcome. We would also be willing to work on a "generic" solution (such as a device driver __halt() routine) if there is interest in this approach.

Messages

1. Through application which has opened the device *by Michael K. Johnson*
2. Device Driver notification of "Linux going down" *by Marko Kohtala*

Through application which has opened the device

Forum: Device Drivers
Re: Device Driver notification of "Linux going down" (Stan Troeh)
Keywords: device drivers shutdown modules init watchdog
Date: Wed, 14 Aug 1996 04:55:38 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

In order to shut down a device, have a user-level application have it opened, and when it is sent SIGTERM by init (or, presumably, any other process), close the device or alert it of the shutdown in some other way.

Device Driver notification of "Linux going down"

Forum: Device Drivers
Re: Device Driver notification of "Linux going down" (Stan Troeh)
Keywords: device drivers shutdown modules init watchdog notifier
Date: Mon, 03 Mar 1997 09:33:14 GMT
From: Marko Kohtala <Marko.Kohtala@ntc.nokia.com>

In 2.1.x kernels there is a boot_notifier_list. See in include/linux/notifier.h and kernel/sys.c.

Is waitv honored?

Forum: Device Drivers
Keywords: waitv VT
Date: Sun, 07 Jul 1996 02:18:18 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

The `vt_mode` structure in /usr/include/linux/vt.h has a member called `waitv` that doesn't seem to be used. That is, drivers/char/vt.c examines and sets it, and drivers/char/tty_io.c resets it when the terminal is reset, but nothing else seems to be done with it.

I'm guessing that it exists because the SVR4 VT code has a structure member of the same name, and that the only reason it is set and reset is for compatibility with apps written for SVR4. Am I right?

PCI Driver

Forum: Device Drivers
Keywords: A PCI Driver ???
Date: Wed, 12 Jun 1996 17:04:44 GMT
From: Flavia Donno <flavia@galileo.pi.infn.it>

Probably this is not the right place for this question, but please ... Answer to me! Has anyone written a PCI driver for Lynux ? Any example ? Documentation ?

Thank you in advance.

```
        Flavia
```

Messages

1. There is linux-2.0/drivers/pci/pci.c *by Hasdi*

There is linux-2.0/drivers/pci/pci.c

Forum: Device Drivers
Re: PCI Driver (Flavia Donno)
Keywords: A PCI Driver ???
Date: Thu, 13 Jun 1996 19:38:02 GMT
From: Hasdi <hasdi@engin.umich.edu>

The subject says it all.

I don't know why pci.c is the only file in the pci directory. I thought there are lots of pci drivers. Is there something about pci that every kernel should know about?

Re: Network Device Drivers

Forum: Device Drivers
Keywords: network driver prototype functions
Date: Wed, 22 May 1996 16:23:09 GMT
From: Paul Gortmaker <gpg109@rsphy1.anu.edu.au>

```
> I don't know anything about this topic. The kernel source
> includes a skeleton.c file that can get you started.
> Someone has promised to write this section, so check back
> sometime...

Hrrm, who was that? (just curious...)

Somebody asked me about a year or so ago as to what the basics
of a net driver would look like.  I haven't seen Alan's article
in linux journal, so this may be useless in comparison.
Regardless, here it is anyway.

Paul.

------------------------------
1) Probe:
        called at boot to check for existence of card. Best if it
        can check un-obtrsively by reading from memory, etc. Can
        also read from i/o ports. Writing to i/o ports in a probe
        is *not* allowed as it may kill another device.
        Some device initialization is usually done here (allocating
        i/o space, IRQs,filling in the dev->??? fields etc.)

2) Interrupt handler:
        Called by the kernel when the card posts an interrupt.
        This has the job of determining why the card posted
        an interrupt, and acting accordingly. Usual interrupt
        conditions are data to be rec'd, transmit completed,
        error conditions being reported.

3) Transmit function
        Linked to dev->hard_start_xmit() and is called by the
        kernel when there is some data that the kernel wants
        to put out over the device. This puts the data onto
        the card and triggers the transmit.

4) Receive function
        Called by the interrupt handler when the card reports
        that there is data on the card. It pulls the data off
        the card, packages it into a sk_buff and lets the
        kernel know the data is there for it by doing a
        netif_rx(sk_buff)

5) Open function
        linked to dev->open and called by the networking layers
        when somebody does "ifconfig <device_name> up" -- this
        puts the device on line and enables it for Rx/Tx of
        data.

Someday, perhaps I will have the time to write a proper
document on the subject....   Naaaaahhhhhh.
```

Messages

1. Re: Network Device Drivers *by Neal Tucker*
    1. network driver info *by Neal Tucker*
    -> Network Driver Desprately Needed *by Paul Atkinson*
2. Transmit function *by Joerg Schorr*
    1. Re: Transmit function *by Paul Gortmaker*
    -> Skbuff *by Joerg Schorr*

Re: Network Device Drivers

Forum: Device Drivers
Re: Re: Network Device Drivers (Paul Gortmaker)
Keywords: network driver functions
Date: Thu, 30 May 1996 10:42:09 GMT
From: Neal Tucker <ntucker@adobe.com>

Paul Gortmaker says:

```
>Somebody asked me about a year or so ago as to what the basics
>of a net driver would look like.
>
>1) Probe:
>        called at boot to check for existence of card. Best if it
>        can check un-obtrsively by reading from memory, etc. Can
>        also read from i/o ports. Writing to i/o ports in a probe
>        is *not* allowed as it may kill another device.
>        Some device initialization is usually done here (allocating
>        i/o space, IRQs,filling in the dev->??? fields etc.)
```

You must be the guy that wrote that part of the ethernet HOWTO. :-)

I've just recently been looking at the network device driver interface, and I read your stuff and this part confused me, since all the code I was looking at (dummy, loopback, slip..) refers to this as "init", rather than "probe". (which may sound a bit nit picky, but there were other routines called "probe" that I studied for a while, thinking they were the important ones (they turned out to be used for module initialization only) :-).

But on to my real reason for writing... One thing that I think would be helpful to people trying to write a network driver for the first time is a description of how this is all hooked into the kernel. I've found plenty of examples of what the actual driver code needs to do, (lots of some_driver.c files, including skeleton.c, which is usually what people point to), but no explanation of how to get it called.

Basically what it comes down to is an explanation of Space.c, which doesn't do very much, but is a bit funny looking to a first-timer. Now that I understand it, it seems a bit obvious, but back when I was going mad trying to figure out why my driver didn't execute, it would have been really nice to have it all spelled out.

So once it's done, I will submit a description. If you'd like, check out a start at http://fester.axis.net/~linux/454.html. Make sure to let me and/or the rest of the world what you think.

-Neal Tucker

Messages

1. network driver info *by Neal Tucker*
-> Network Driver Desprately Needed *by Paul Atkinson*

network driver info

Forum: Device Drivers
Re: Re: Network Device Drivers (Paul Gortmaker)
Re: Re: Network Device Drivers (Neal Tucker)
Keywords: network driver functions
Date: Sat, 15 Jun 1996 03:33:21 GMT
From: Neal Tucker <ntucker@adobe.com>

Earlier, I posted a pointer to a bit of info on network device drivers, and the site that the web page is on is going away, so I am including what was there here...

How a Network Device Gets Added to the Kernel

There is a global variable called `dev_base` which points to a linked list of "device" structures. Each record represents a network device, and contains a pointer to the device driver's initialization function. The initialization function is the first code from the driver to ever get executed, and is responsible for setting up the hooks to the other driver code.

At boot time, the function `device_setup` (*drivers/block/genhd.c*) calls a function called `net_dev_init` (*net/core/dev.c*) which walks through the linked list pointed to by `dev_base`, calling each device's `init` function. If the `init` indicates failure (by returning a nonzero result), `net_dev_init` removes the device from the linked list and continues on.

This brings up the question of how the devices get added to the linked list of devices before any of their code is executed. That is accomplished by a clever piece of C preprocessor work in *drivers/net/Space.c*. This file has the static declarations for each device's "device" struct, including the pointer to the next device in the list. How can we define these links statically without knowing which devices are going to be included? Here's how it's done (from *drivers/net/Space.c*):

```
#define NEXT_DEV        NULL

#if defined(CONFIG_SLIP)
static struct device slip_dev =
{
  device name and some other info goes here
  ...
  NEXT_DEV,     /* <- link to previously listed */
                /*    device struct (NULL here) */
```

```
      slip_init,   /* <- pointer to init function */
    };

    #undef NEXT_DEV
    #define NEXT_DEV (&slip_dev)
    #endif

    #if defined(CONFIG_PPP)
    static struct device ppp_dev =
    {
      device name and some other info goes here
      ...
      NEXT_DEV,    /* <- link to previously listed   */
                   /*    device struct, which is now *
                   /*    defined as &slip_dev        */
      ppp_init,    /* <- pointer to init function */
    };

    #undef NEXT_DEV
    #define NEXT_DEV (&ppp_dev)
    #endif

    struct device loopback_dev =
    {
      device name and some other info goes here
      ...
      NEXT_DEV,         /* <- link to previously listed   */
                        /*    device struct, which is now */
                        /*    defined as &ppp_dev         */
      loopback_init,    /* <- pointer to init function */
    };

    /* And finally, the head of the list, which points   */
    /* to the most recently defined device struct,       */
    /* loopback_dev.  This (dev_base) is the pointer the */
    /* kernel uses to access all the devices.            */

    struct device *dev_base = &loopback_dev;
```

There is a constant, `NEXT_DEV`, defined to always point at the last device record declared. When each device record gets declared, it puts the value of `NEXT_DEV` in itself as the "next" pointer and then *redefines* `NEXT_DEV` to point to itself. This is how the linked list is built. Note that `NEXT_DEV` starts out `NULL` so that the first device structure is the end of the list, and at the end, the global `dev_base`, which is the head of the list, gets the value of the last device structure.

Ethernet devices

Ethernet devices are a bit of a special case in how they get called at initialization time, probably due to the fact that there are so many different types of ethernet devices that we'd like to be able to refer to them by just calling them ethernet devices (ie "eth0", "eth1", etc), rather than calling them by name (ie "NE2000", "3C509", etc).

In the linked list mentioned above, there is a single entry for all ethernet devices, whose initialization function is set to the function `ethif_probe` (also defined in *drivers/net/Space.c*). This function simply calls each ethernet device's `init` function until it finds one that succeeds. This is done with a huge expression made up of the ANDed results of the calls to the initialization functions (note that with the ethernet devices, the init function is conventionally called *`xxx`*`_probe`). Here is an abridged version of that function:

```
    static int ethif_probe(struct device *dev)
    {
        u_long base_addr = dev->base_addr;

        if ((base_addr == 0xffe0)  ||  (base_addr == 1))
            return 1;

        if (1                  /* note start of expression here */
    #ifdef CONFIG_DGRS
            && dgrs_probe(dev)
    #endif
    #ifdef CONFIG_VORTEX
            && tc59x_probe(dev)
    #endif
    #ifdef CONFIG_NE2000
            && ne_probe(dev)
    #endif
            && 1 ) {         /* end of expression here         */
            return 1;
        }
        return 0;
    }
```

The result is that the if statement bails out as false if any of the `probe` calls returns zero (success), and only one ethernet card is initialized and used, no matter how many drivers you have installed. For the drivers that aren't installed, the `#ifdef` removes the code completely, and the expression gets a bit smaller. The implications of this scheme are that supporting multiple ethernet cards is now a special case, and requires providing command line parameters to the kernel which cause `ethif_probe` to be executed multiple times.

Messages

1. Network Driver Desprately Needed *by Paul Atkinson*

Network Driver Desprately Needed

Forum: Device Drivers
Re: Re: Network Device Drivers (Paul Gortmaker)
Re: Re: Network Device Drivers (Neal Tucker)
Re: network driver info (Neal Tucker)
Keywords: device drivers network compaq tlan thunderlan netflex
Date: Tue, 06 May 1997 20:56:36 GMT
From: Paul Atkinson <patkinson@aerotek.co.uk>

I have looked everywhere for a Compaq Netflex 100BaseT network card device driver/patch and have come up with nothing :(I wouldn't know where to start to make my own (I have a hard enough time recompiling the kernel!). If anyone would like to fill a void in Linux Hardware Compatibility it would be very much appreciated. The card is based on a T1 ThunderLAN chip.

Many thanks

Paul.

Transmit function

Forum: Device Drivers
Re: Re: Network Device Drivers (Paul Gortmaker)
Keywords: network driver prototype functions
Date: Fri, 31 May 1996 20:55:37 GMT
From: Joerg Schorr <jschorr@studi.epfl.ch>

```
> 3) Transmit function
>         Linked to dev->hard_start_xmit() and is called by the
>         kernel when there is some data that the kernel wants
>         to put out over the device. This puts the data onto
>         the card and triggers the transmit.

Well, i'm having to some work with network on linux, and i also
noticed this part for transmit; but the PC i am working on, uses
a  WD80x3 card (using the wd.c driver), and as it seems the transmit function
is wd_block_output; but where are between the dev->hard_start_xmit
and the wd_block_ouptut??
I haven't it out for the moment.
```

Messages

1. Re: Transmit function *by Paul Gortmaker*
-> Skbuff *by Joerg Schorr*

Re: Transmit function

Forum: Device Drivers
Re: Re: Network Device Drivers (Paul Gortmaker)
Re: Transmit function (Joerg Schorr)
Keywords: network driver prototype functions
Date: Fri, 31 May 1996 23:55:18 GMT
From: Paul Gortmaker <*unknown*>

The wd driver is not a complete driver by itself. It uses the code in 8390.c to do most of the work. The function ei_transmit() in 8390.c is what is linked to dev->hard_start_xmit(), and then ei_transmit will call ei_block_output() which in this case is pointing at wd_block_output().

Paul.

Messages

1. Skbuff *by Joerg Schorr*

Skbuff

Forum: Device Drivers
Re: Re: Network Device Drivers (Paul Gortmaker)
Re: Transmit function (Joerg Schorr)
Re: Re: Transmit function (Paul Gortmaker)
Keywords: network driver prototype functions
Date: Thu, 06 Jun 1996 19:39:48 GMT
From: Joerg Schorr <jschorr@studi.epfl.ch>

```
In the wd_block_output (in wd.c) function, there is a moment
where the buf (which is skb->data) is copied to the shared
memory of the ethercard (if I understood it right). But I
didn't found out when the message and headers (ip and udp for
the case I am interested in) are copied in skb->data??

Also: what is exactly in skb->data?? Is there more than the
message and the headers?? Also the rest of the skbuffer??
```

Filesystems

There has been very little documentation so far regarding writing filesystems for Linux. Let's change that...

A tour of the Linux VFS
: Before you can consider writing a filesystem for Linux, you need to have at least a vague understanding of how the Linux Virtual Filesystem Switch operates. The basic principles are fairly simple.

Design and Implementation of the Second Extended Filesystem
: The three main authors and designers of ext2fs have written an excellent paper on it, and have contributed it to the KHG. This paper was first published in the Proceedings of the First Dutch International Symposium on Linux, ISBN 90-367-0385-9.

Filesystem Tutorial
: It may be a long time before I get around to writing a tutorial on how to write a Linux filesystem; after all, I've never done it. Would someone else like to help?

Messages

12. Need /proc info *by Kai Xu* NEW
11. Where to find libext2 sources? *by Mark Salter*
    1. Nevermind... *by Mark Salter*
10. New File System *by Vamsi Krishna*
9. Partition? *by Wilfredo Lugo Beauchamp*
    1. Please be more specific.... *by Theodore Ts'o*
8. Need documentation on userfs implementation *by Natchu Vishnu Priya*
7. ext2fs tools *by Wilfredo Lugo Beauchamp*
    1. Where to find e2fsprogs *by Theodore Ts'o*
6. libext2fs documentation *by James Beckett*
    1. libext2fs documentation *by Theodore Ts'o*
5. proc filesystem *by Praveen Krishnan*
    1. man proc *by Michael K. Johnson*
4. Need NFS documentation *by Ermelindo Mauriello*
3. Even more ext2 documentation! *by Michael K. Johnson*
2. More ext2 documentation *by Michael K. Johnson*

1. Ext2 paper *by Theodore Ts'o*
    1. Done. *by Michael K. Johnson*

A tour of the Linux VFS

I'm not an expert on this topic. I've never written a filesystem from scratch; I've only worked on the proc filesystem, and I didn't do much real filesystem hacking there, only extensions to what was already there.

So if you see any mistakes or ommissions here (there have ***got*** *to be ommissions in a piece this short on a topic this large), please respond, in order to let me fix them and let other people know about them.*

In Linux, all files are accessed through the Virtual Filesystem Switch, or VFS. This is a layer of code which implements generic filesystem actions and vectors requests to the correct specific code to handle the request. Two main types of code modules take advantage of the VFS services, device drivers and filesystems. Because device drivers are covered elsewhere in the KHG, we won't cover them explicitly here. This tour will focus on filesystems. Because the VFS doesn't exist in a vacuum, we'll show its relationship with the favorite Linux filesystem, the ext2 filesystem.

One warning: without a decent understanding of the system calls that have to do with files, you are not likely to be able to make heads or tails of filesystems. Most of the VFS and most of the code in a normal Linux filesystem is pretty directly related to completing normal system calls, and you will not be able to understand how the rest of the system works without understanding the system calls on which it is based.

Where to find the code

The source code for the VFS is in the fs/ subdirectory of the Linux kernel source, along with a few other related pieces, such as the buffer cache and code to deal with each executable file format. Each specific filesystem is kept in a lower subdirectory; for example, the ext2 filesystem source code is kept in fs/ext2/.

This table gives the names of the files in the fs/ subdirectory and explains the basic purpose of each one. The middle column, labeled **system**, is supposed to show to which major subsystem the file is (mainly) dedicated. EXE means that it is used for recognizing and loading executable files. DEV means that is for device driver support. BUF means buffer cache. VFS means that it is a part of the VFS, and delegates some functionality to filesystem-specific code. VFSg means that this code is completely generic and never delegates part of its operation to specific filesystem code (that I noticed, anyway) and which you shouldn't have to worry about while writing a filesystem.

Filename	**system**	**Purpose**
binfmt_aout.c	EXE	Recognize and execute old-style a.out executables.
binfmt_elf.c	EXE	Recognize and execute new ELF executables
binfmt_java.c	EXE	Recognize and execute Java apps and applets
binfmt_script.c	EXE	Recognize and execute #!-style scripts
block_dev.c	DEV	Generic read(), write(), and fsync() functions for block devices.
buffer.c	BUF	The buffer cache, which caches blocks read from block devices.
dcache.c	VFS	The directory cache, which caches directory name lookups.
devices.c	DEV	Generic device support functions, such as registries.
dquot.c	VFS	Generic disk quota support.
exec.c	VFSg	Generic executable support. Calls functions in the binfmt_* files.
fcntl.c	VFSg	fcntl() handling.
fifo.c	VFSg	fifo handling.
file_table.c	VFSg	Dynamically-extensible list of open files on the system.
filesystems.c	VFS	All compiled-in filesystems are initialized from here by calling `init_*name*_fs()`.

inode.c	VFSg	Dynamically-extensible list of open inodes on the system.
ioctl.c	VFS	First-stage handling for ioctl's; passes handling to the filesystem or device driver if necessary.
locks.c	VFSg	Support for fcntl() locking, flock() locking, and manadatory locking.
namei.c	VFS	Fills in the inode, given a pathname. Implements several name-related system calls.
noquot.c	VFS	No quotas: optimization to avoid `#ifdef`'s in dquot.c
open.c	VFS	**Lots** of system calls including (surprise) open(), close(), and vhangup().
pipe.c	VFSg	Pipes.
read_write.c	VFS	read(), write(), readv(), writev(), lseek().
readdir.c	VFS	Several different interfaces for reading directories.
select.c	VFS	The guts of the select() system call
stat.c	VFS	stat() and readlink() support.
super.c	VFS	Superblock support, filesystem registry, mount()/umount().

Attaching a filesystem to the kernel

If you look at the code in any filesystem for `init_`*`name`*`_fs()`, you will find that it probably contains about one line of code. For instance, in the ext2fs, it looks like this (from fs/ext2/super.c):

```
int init_ext2_fs(void)
{
        return register_filesystem(&ext2_fs_type);
}
```

All it does is register the filesystem with the registry kept in fs/super.c. `ext2_fs_type` is a pretty simple structure:

```
static struct file_system_type ext2_fs_type = {
        ext2_read_super, "ext2", 1, NULL
};
```

The `ext2_read_super` entry is a pointer to a function which allows a filesystem to be mounted (among other things; more later). `"ext2"` is the name of the filesystem type, which is used (when you type `mount ... -t ext2`) to determine which filesystem to use to mount a device. The `1` says that it needs a device to be mounted on (unlike the proc filesyste or a network filesystem), and the `NULL` is required to fill up space that will be used to keep a linked list of filesystem types in the filesystem registry, kept in (look it up in the table!) fs/super.c.

It's possible for a filesystem to support more than one type of filesystem. For instance, in fs/sysv/inode.c, three possible filesystem types are supported by one filesystem, with this code:

```
static struct file_system_type sysv_fs_type[3] = {
        {sysv_read_super, "xenix", 1, NULL},
        {sysv_read_super, "sysv", 1, NULL},
        {sysv_read_super, "coherent", 1, NULL}
};

int init_sysv_fs(void)
{
        int i;
        int ouch;

        for (i = 0; i < 3; i++) {
                if ((ouch = register_filesystem(&sysv_fs_type[i])) != 0)
                        return ouch;
        }
        return ouch;
}
```

Connecting the filesystem to a disk

The rest of the communication between the filesystem code and the kernel doesn't happen until a device bearing that type of file system is mounted. When you mount a device containing an ext2 file system, `ext2_read_super()` is called. If it succeeds in reading the superblock and is able to mount the filesystem, it fills in the `super_block` structure with information that includes a pointer to a structure called `super_operations`, which contains pointers to functions which do common operations related to superblocks; in this case, pointers to functions specific to ext2.

A superblock is the block that defines an entire filesystem on a device. It is sometimes mythical, as in the case of the DOS filesystem--that is, the filesystem may or may not actually have a block on disk that is the real superblock. If not, it has to make something up. Operations that pertain to the filesystem as a whole (as opposed to individual files) are considered superblock operations. The `super_operations` structure contains pointers to functions which manipulate inodes, the superblock, and which refer to or change the status of the filesystem as a whole (`statfs()` and `remount()`).

You have probably noticed that there are a lot of pointers, and especially pointers to functions, here. The good news is that all the messy pointer work is done; that's the VFS's job. All the author for the filesystem needs to do is fill in (usually static) structures with pointers to functions, and pass pointers to those structures back to the VFS so it can get at the filesystem and the files.

For example, the super_operations structure looks like this (from `<linux/fs.h>`):

```
struct super_operations {
        void (*read_inode) (struct inode *);
        int (*notify_change) (struct inode *, struct iattr *);
        void (*write_inode) (struct inode *);
        void (*put_inode) (struct inode *);
        void (*put_super) (struct super_block *);
        void (*write_super) (struct super_block *);
        void (*statfs) (struct super_block *, struct statfs *, int);
        int (*remount_fs) (struct super_block *, int *, char *);
};
```

That's the VFS part. Here's the much simpler declaration of the ext2 instance of that structure, in fs/ext2/super.c:

```
static struct super_operations ext2_sops = {
        ext2_read_inode,
        NULL,
        ext2_write_inode,
        ext2_put_inode,
        ext2_put_super,
        ext2_write_super,
        ext2_statfs,
        ext2_remount
};
```

First, notice that an unneeded entry has simply been set to `NULL`. That's pretty normal Linux behavior; whenever there is a sensible default behavior of a function pointer, and that sensible default is what you want, you are almost sure to be able to provide a `NULL` pointer and get the default painlessly. Second, notice how simple and clean the declaration is. All the painful stuff like `sb->s_op->write_super(sb);` s hidden in the VFS implementation.

The details of how the filesystem actually reads and writes the blocks, including the superblock, from and to the disk will be covered in a different section. There will actually be (I hope) two descriptions--a simple, functional one in a section on how to write filesystems, and a more detailed one in a tour through the buffer cache. For now, assume that it is done by magic...

Mounting a filesystem

When a filesystem is mounted (which file is in charge of mounting a filesystem? Look at the table above, and find that it is fs/super.c. You might want to follow along in fs/super.c), `do_umount()` calls read_super, which ends up calling (in the case of the ext2 filesystem), `ext2_read_super()`, which returns the superblock. That superblock includes a pointer to that structure of pointers to functions that we see in the definition of `ext2_sops` above. It also includes a lot of other data; you can look at the definition of `struct super_block` in include/linux/fs.h if you like.

Finding a file

Once a filesystem is mounted, it is possible to access files on that filesystem. There are two main steps here: looking up the name to find what inode it points to, and then accessing the inode.

When the VFS is looking at a name, it includes a path. Unless the filename is absolute (it starts with a `/` character), it is relative to the current directory of the process that made the system call that included a path. It uses filesystem-specific code to look up files on the filesystems specified. It takes the path name one component (filename components are separated with `/` characters) at a time, and looks it up. If it is a directory, then the next component is looked up in the directory returned by the previous lookup. Every component which is looked up, whether it is a file or a directory, returns an **inode number** which uniquely identifies it, and by which its contents are accessed.

If the file turns out to be a symbolic link to another file, then the VFS starts over with the new name which is retrieved from the symbolic link. In order to prevent infinite recursion, there's a limit on the **depth** of symlinks; the kernel will only follow so many symlinks in a row before giving up.

When the VFS and the filesystem together have resolved a name into an inode number (that's the `namei()` function in namei.c), then the inode can be accessed. The `iget()` function finds **and returns** the inode specified by an inode number. The `iput()` function is later used to release access to the inode. It is kind of like `malloc()` and `free()`, except that more than one process may hold an inode open at once, and a reference count is maintained to know when it's free and when it's not.

The integer file handle which is passed back to the application code is an offset into a file table for that process. That file table slot holds the inode number that was looked up with the `namei()` function until the file is closed or the process terminates. So whenever a process does anything to a ``file'' using a file handle, it is really manipulating the inode in question.

inode Operations

That inode number and `inode` structure have to come from somewhere, and the VFS can't make them up on it's own. They have to come from the filesystem. So how does the VFS look up the name in the filesystem and get an `inode` back?

It starts at the beginning of the path name and looks up the inode of the first directory in the path. Then it uses that inode to look up the next directory in the path. When it reachs the end, it has found the inode of the file or directory it is trying to look up. But since it needs an `inode` to get started, how **does** it get started with the first lookup? There is an inode pointer kept in the superblock called `s_mounted` which points at an inode structure for the filesystem. This inode is allocated when the filesystem is mounted and de-allocated when the filesystem is unmounted. Normally, as in the ext2 filesystem, the `s_mounted` inode is the inode of the root directory of the filesystem. From there, all the other inodes can be looked up.

Each inode includes a pointer to a structure of pointers to functions. Sound familiar? This is the `inode_operations` structure. One of the elements of that structure is called `lookup()`, and it is used to look up another inode on the same filesystem. In general, a filesystem has only one `lookup()` function that is the same in every inode on the filesystem, but it is possible to have several different `lookup()` functions and assign them as appropriate for the filesystem; the proc filesystem does this because different directories in the proc filesystem have different purposes. The `inode_operations` structure looks like this (defined, like most everything we are looking at, in `<linux/fs.h>`):

```
struct inode_operations {
```

```
            struct file_operations * default_file_ops;
            int (*create) (struct inode *,const char *,int,int,struct inode **);
            int (*lookup) (struct inode *,const char *,int,struct inode **);
            int (*link) (struct inode *,struct inode *,const char *,int);
            int (*unlink) (struct inode *,const char *,int);
            int (*symlink) (struct inode *,const char *,int,const char *);
            int (*mkdir) (struct inode *,const char *,int,int);
            int (*rmdir) (struct inode *,const char *,int);
            int (*mknod) (struct inode *,const char *,int,int,int);
            int (*rename) (struct inode *,const char *,int,struct inode *,const char
*,int);
            int (*readlink) (struct inode *,char *,int);
            int (*follow_link) (struct inode *,struct inode *,int,int,struct inode
**);
            int (*readpage) (struct inode *, struct page *);
            int (*writepage) (struct inode *, struct page *);
            int (*bmap) (struct inode *,int);
            void (*truncate) (struct inode *);
            int (*permission) (struct inode *, int);
            int (*smap) (struct inode *,int);
    };
```

Most of these functions map directly to system calls.

In the ext2 filesystem, directories, files, and symlinks have different `inode_operations` (this is normal). The file fs/ext2/dir.c contains `ext2_dir_inode_operations`, the file fs/ext2/file.c contains `ext2_file_inode_operations`, and the file fs/ext2/symlink.c contains `ext2_symlink_inode_operations`.

There are many system calls related to files (and directories) which aren't accounted for in the `inode_operations` structure; those are found in the `file_operations` structure. The `file_operations` structure is the same one used when writing device drivers and contains operations that work specifically on files, rather than inodes:

```
    struct file_operations {
            int (*lseek) (struct inode *, struct file *, off_t, int);
            int (*read) (struct inode *, struct file *, char *, int);
            int (*write) (struct inode *, struct file *, const char *, int);
            int (*readdir) (struct inode *, struct file *, void *, filldir_t);
            int (*select) (struct inode *, struct file *, int, select_table *);
            int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
            int (*mmap) (struct inode *, struct file *, struct vm_area_struct *);
            int (*open) (struct inode *, struct file *);
            void (*release) (struct inode *, struct file *);
            int (*fsync) (struct inode *, struct file *);
            int (*fasync) (struct inode *, struct file *, int);
            int (*check_media_change) (kdev_t dev);
            int (*revalidate) (kdev_t dev);
    };
```

There are also a few functions which aren't directly related to system calls--and where they don't apply, they can simply be set to `NULL`.

Summary

The role of the VFS is:

- Keep track of available filesystem types.

- Associate (and disassociate) devices with instances of the appropriate filesystem.
- Do any reasonable generic processing for operations involving files.
- When filesystem-specific operations become necessary, vector them to the filesystem in charge of the file, directory, or inode in question.

The interaction between the VFS and specific filesystem types occurs through two main data structures, the `super_block` structure and the `inode` structure, and their associated data structures, including `super_operations`, `inode_operations`, `file_operations`, and others, which are kept in the include file `<linux/fs.h>`.

Therefore, the role of a specific filesystem code is to provide a superblock for each filesystem mounted and a unique inode for each file on the filesystem, and to provide code to carry out actions specific to filesystems and files that are requested by system calls and sorted out by the VFS.

Messages

1. A couple of comments and corrections *by Jeremy Fitzhardinge*

A couple of comments and corrections

Forum: A tour of the Linux VFS
Keywords: comments, correction, implementation details
Date: Thu, 23 May 1996 03:20:42 GMT
From: Jeremy Fitzhardinge <jeremy@zip.com.au>

```
A minor point: binfmt_java only deals with Java class files,
not JavaScript (which is unrelated to Java in all but name).

In general it looks OK, but it doesn't really talk about the
distinction between file operation and inode operations (or
the distinction between struct file and struct inode).

Inodes represent entities on disks: there's one (and only one)
for each file, directory, symlink, device and FIFO on the
filesystem.

When a user process does an open() syscall, it does a couple
of things:
        - it looks up the inode for the pathname
        - it puts the inode into the inode cache
        - it allocates a new file structure
        - it allocates a filedescriptor slot, and points it
          it at the file structure

The number of the file descriptor slot is what is returned.

The distinction is subtle but important.  Inodes have no
notion of "current offset": that's in the file structure,
so one inode can have many file structures, each at a
different offset and mode (RO/WO/RW/append).  If you use
the dup() syscall, you can have multiple file descriptors
pointing to the same file structure (so they share their
offsets and open modes).

What this means for a filesystem is that you rarely need
to implement the open() file operation, unless you actually
care about specific file open and close operations.  However,
you must always implement iget() operations.
```

```
I guess there's lots more that can go here:
        - interaction with the VM system and page cache
        - coping with dynamic filesystems (filesystems who's
          contents change of their own accord)
        - structure of a Unix filesystem (inodes/names/links)
        - what order to do things in, and how the various
          VFS calls interact
        - documenting the args of the VFS interface functions

I know this is supposed to be documentation, but the kernel
source is a good place to look...

[Plug mode] A good way of finding out about how Linux
filesystems work is Userfs, which allows you to write user
processes which implement filesystems.  It has a readme
which talks in some detail about how to implement filesystems,
which is somewhat applicable to writing kernel resident
filesystems too.  It's available at
tsx-11.mit.edu:pub/linux/ALPHA/userfs.  The structure of
the userfs kernel module is also a pretty good guide for
a skeletal filesystem.

        J
```

Design and Implementation of the Second Extended Filesystem

Rémy Card, Laboratoire MASI--Institut Blaise Pascal, E-Mail: card@masi.ibp.fr, and Theodore Ts'o, Massachussets Institute of Technology, E-Mail: tytso@mit.edu, and Stephen Tweedie, University of Edinburgh, E-Mail: sct@dcs.ed.ac.uk

Introduction

Linux is a Unix-like operating system, which runs on PC-386 computers. It was implemented first as extension to the Minix operating system [Tanenbaum 1987] and its first versions included support for the Minix filesystem only. The Minix filesystem contains two serious limitations: block addresses are stored in 16 bit integers, thus the maximal filesystem size is restricted to 64 mega bytes, and directories contain fixed-size entries and the maximal file name is 14 characters.

We have designed and implemented two new filesystems that are included in the standard Linux kernel. These filesystems, called ``Extended File System'' (Ext fs) and ``Second Extended File System'' (Ext2 fs) raise the limitations and add new features.

In this paper, we describe the history of Linux filesystems. We briefly introduce the fundamental concepts implemented in Unix filesystems. We present the implementation of the Virtual File System layer in Linux and we detail the Second Extended File System kernel code and user mode tools. Last, we present performance measurements made on Linux and BSD filesystems and we conclude with the current status of Ext2fs and the future directions.

History of Linux filesystems

In its very early days, Linux was cross-developed under the Minix operating system. It was easier to share disks between the two systems than to design a new filesystem, so Linus Torvalds decided to implement support for the Minix filesystem in Linux. The Minix filesystem was an efficient and relatively bug-free piece of software.

However, the restrictions in the design of the Minix filesystem were too limiting, so people started thinking and working on the implementation of new filesystems in Linux.

In order to ease the addition of new filesystems into the Linux kernel, a Virtual File System (VFS) layer was developed. The VFS layer was initially written by Chris Provenzano, and later rewritten by Linus Torvalds before it was integrated into the Linux kernel. It is described in The Virtual File System.

After the integration of the VFS in the kernel, a new filesystem, called the ``Extended File System'' was implemented in April 1992 and added to Linux 0.96c. This new filesystem removed the two big Minix limitations: its maximal size was 2 giga bytes and the maximal file name size was 255 characters. It was an improvement over the Minix filesystem but some problems were still present in it. There was no support for the separate access, inode modification, and data modification timestamps. The filesystem used linked lists to keep track of free blocks and inodes and this produced bad performances: as the filesystem was used, the lists became unsorted and the filesystem became fragmented.

As a response to these problems, two new filesytems were released in Alpha version in January 1993: the Xia filesystem and the Second Extended File System. The Xia filesystem was heavily based on the Minix filesystem kernel code and only added a few improvements over this filesystem. Basically, it provided long file names, support for bigger partitions and support for the three timestamps. On the other hand, Ext2fs was based on the Extfs code with many reorganizations and many improvements. It had been designed with evolution in mind and contained space for future improvements. It will be described with more details in The Second Extended File System

When the two new filesystems were first released, they provided essentially the same features. Due to its minimal design, Xia fs was more stable than Ext2fs. As the filesystems were used more widely, bugs were fixed in Ext2fs and lots of improvements and new features were integrated. Ext2fs is now very stable and has become the de-facto standard Linux filesystem.

This table contains a summary of the features provided by the different filesystems:

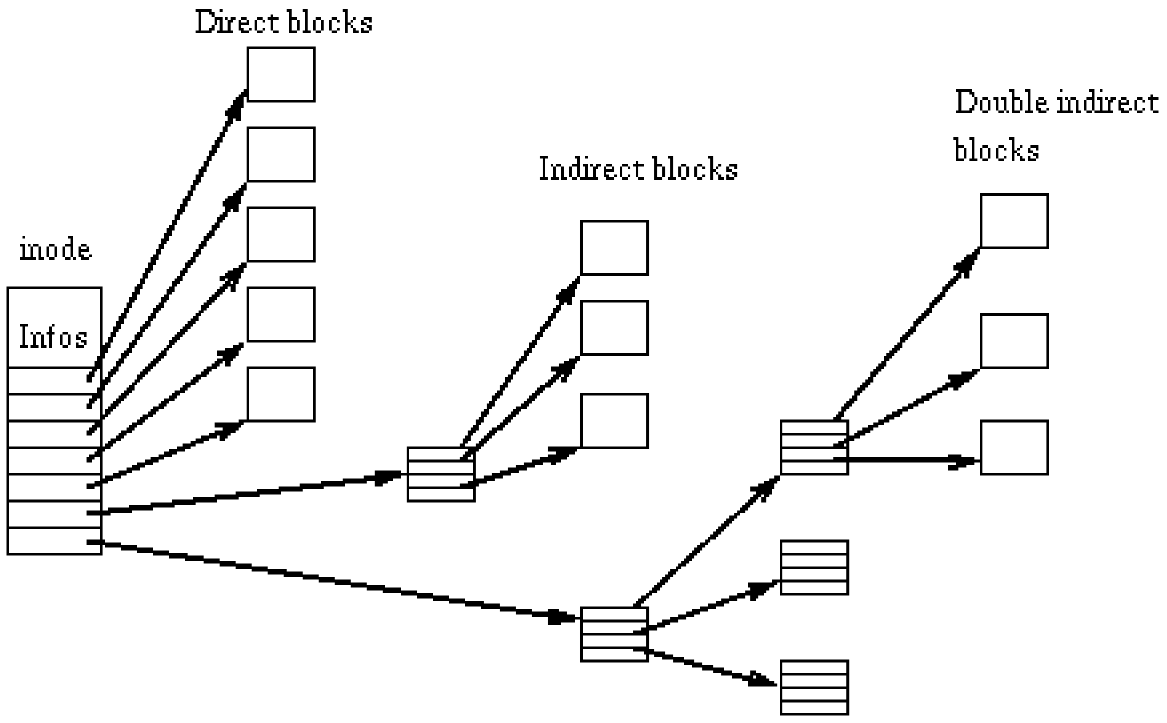
	Minix FS	**Ext FS**	**Ext2 FS**	**Xia FS**
Max FS size	64 MB	2 GB	4 TB	2 GB
Max file size	64 MB	2 GB	2 GB	64 MB
Max file name	16/30 c	255 c	255 c	248 c
3 times support	No	No	Yes	Yes
Extensible	No	No	Yes	No
Var. block size	No	No	Yes	No
Maintained	Yes	No	Yes	?

Basic File System Concepts

Every Linux filesystem implements a basic set of common concepts derivated from the Unix operating system [Bach 1986] files are represented by inodes, directories are simply files containing a list of entries and devices can be accessed by requesting I/O on special files.

Inodes

Each file is represented by a structure, called an inode. Each inode contains the description of the file: file type, access rights, owners, timestamps, size, pointers to data blocks. The addresses of data blocks allocated to a file are stored in its inode. When a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table and reads or writes the physical block. This figure represents the structure of an inode:

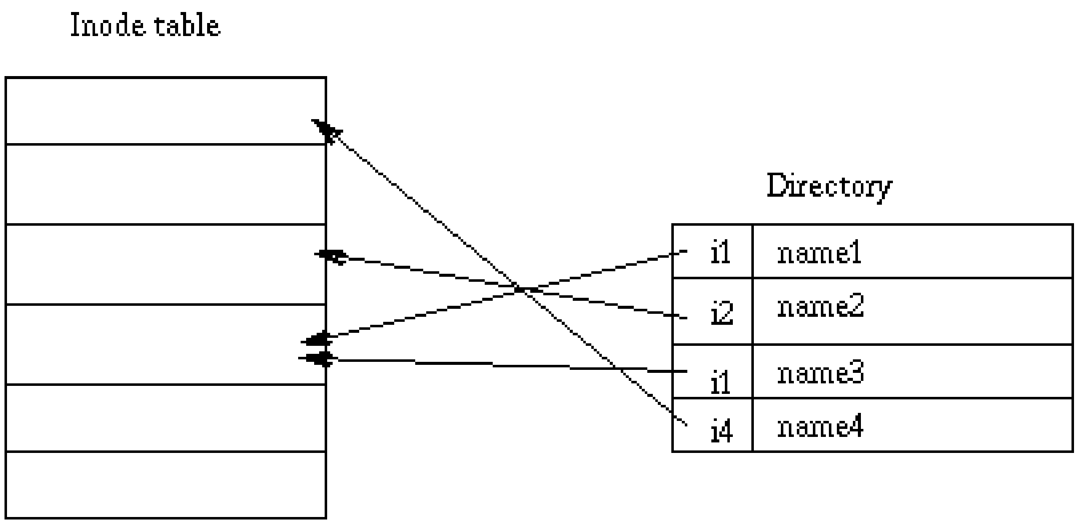


Directories

Directories are structured in a hierarchical tree. Each directory can contain files and subdirectories.

Directories are implemented as a special type of files. Actually, a directory is a file containing a list of entries. Each entry contains an inode number and a file name. When a process uses a pathname, the kernel code searchs in the directories to find the corresponding inode number. After the name has been converted to an inode number, the inode is loaded into memory and is used by subsequent requests.

This figure represents a directory:



Links

Unix filesystems implement the concept of link. Several names can be associated with a inode. The inode contains a field containing the number associated with the file. Adding a link simply consists in creating a directory entry, where the inode number points to the inode, and in incrementing the links count in the inode. When a link is deleted, i.e. when one uses the `rm` command to remove a filename, the kernel decrements the links count and deallocates the inode if this count becomes zero.

This type of link is called a hard link and can only be used within a single filesystem: it is impossible to create cross-filesystem hard links. Moreover, hard links can only point on files: a directory hard link cannot be created to prevent the apparition of a cycle in the directory tree.

Another kind of links exists in most Unix filesystems. Symbolic links are simply files which contain a filename. When the kernel encounters a symbolic link during a pathname to inode conversion, it replaces the name of the link by its contents, i.e. the name of the target file, and restarts the pathname interpretation. Since a symbolic link does not point to an inode, it is possible to create cross-filesystems symbolic links. Symbolic links can point to any type of file, even on nonexistent files. Symbolic links are very useful because they don't have the limitations associated to hard links. However, they use some disk space, allocated for their inode and their data blocks, and cause an overhead in the pathname to inode conversion because the kernel has to restart the name interpretation when it encounters a symbolic link.

Device special files

In Unix-like operating systems, devices can be accessed via special files. A device special file does not use any space on the filesystem. It is only an access point to the device driver.

Two types of special files exist: character and block special files. The former allows I/O operations in character mode while the later requires data to be written in block mode via the buffer cache functions. When an I/O request is made on a special file, it is forwarded to a (pseudo) device driver. A special file is referenced by a major number, which identifies the device type, and a minor number, which identifies the unit.

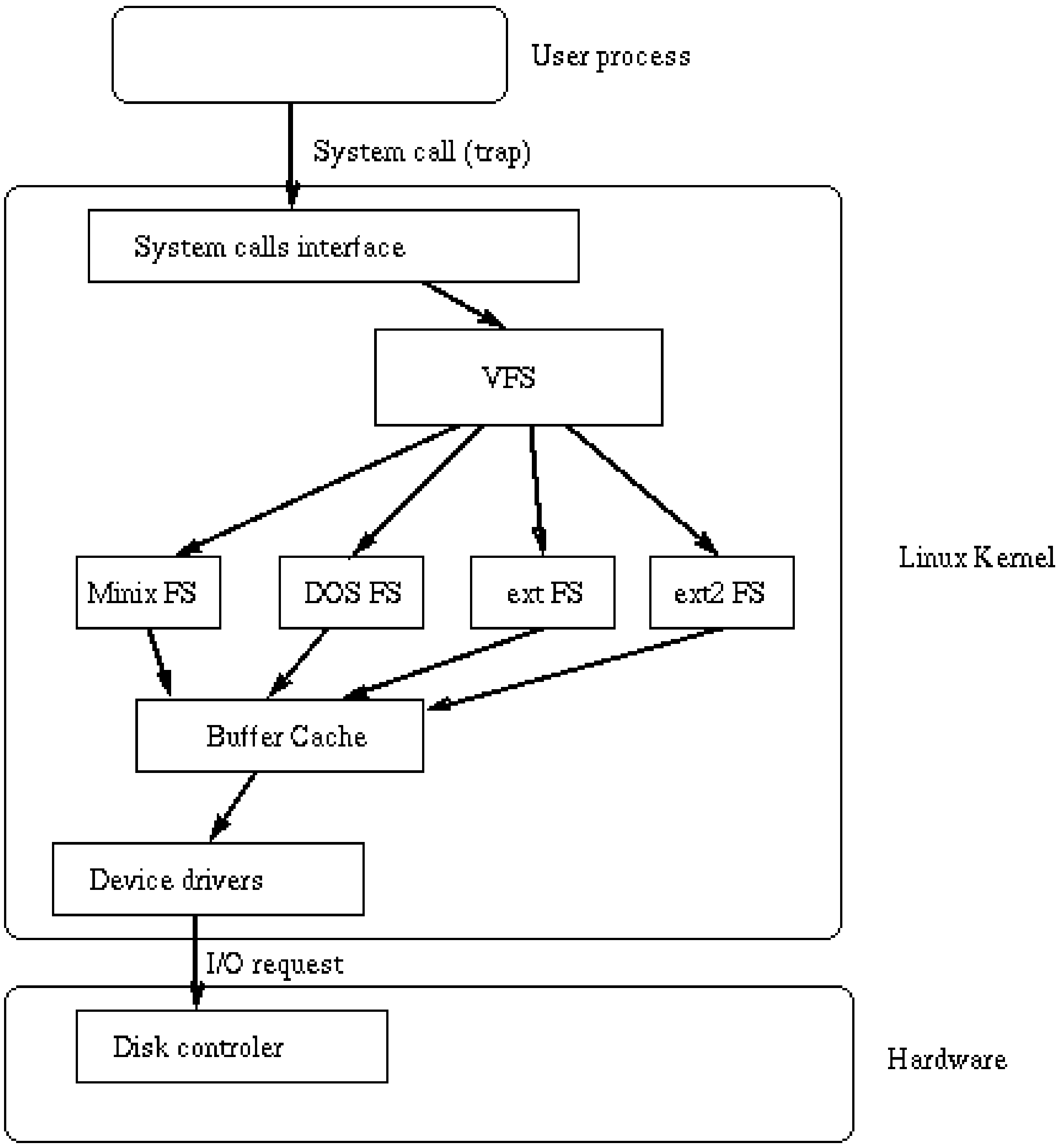
The Virtual File System

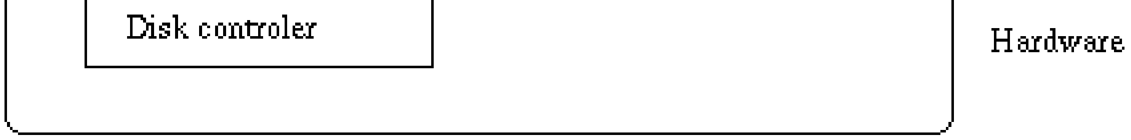
Principle

The Linux kernel contains a Virtual File System layer which is used during system calls acting on files. The VFS is an indirection layer which handles the file oriented system calls and calls the necessary functions in the physical filesystem code to do the I/O.

This indirection mechanism is frequently used in Unix-like operating systems to ease the integration and the use of several filesystem types [Kleiman 1986, Seltzer *et al.* 1993].

When a process issues a file oriented system call, the kernel calls a function contained in the VFS. This function handles the structure independent manipulations and redirects the call to a function contained in the physical filesystem code, which is responsible for handling the structure dependent operations. Filesystem code uses the buffer cache functions to request I/O on devices. This scheme is illustrated in this figure:





The VFS structure

The VFS defines a set of functions that every filesystem has to implement. This interface is made up of a set of operations associated to three kinds of objects: filesystems, inodes, and open files.

The VFS knows about filesystem types supported in the kernel. It uses a table defined during the kernel configuration. Each entry in this table describes a filesystem type: it contains the name of the filesystem type and a pointer on a function called during the mount operation. When a filesystem is to be mounted, the appropriate mount function is called. This function is responsible for reading the superblock from the disk, initializing its internal variables, and returning a mounted filesystem descriptor to the VFS. After the filesystem is mounted, the VFS functions can use this descriptor to access the physical filesystem routines.

A mounted filesystem descriptor contains several kinds of data: informations that are common to every filesystem types, pointers to functions provided by the physical filesystem kernel code, and private data maintained by the physical filesystem code. The function pointers contained in the filesystem descriptors allow the VFS to access the filesystem internal routines.

Two other types of descriptors are used by the VFS: an inode descriptor and an open file descriptor. Each descriptor contains informations related to files in use and a set of operations provided by the physical filesystem code. While the inode descriptor contains pointers to functions that can be used to act on any file (e.g. `create`, `unlink`), the file descriptors contains pointer to functions which can only act on open files (e.g. `read`, `write`).

The Second Extended File System

Motivations

The Second Extended File System has been designed and implemented to fix some problems present in the first Extended File System. Our goal was to provide a powerful filesystem, which implements Unix file semantics and offers advanced features.

Of course, we wanted to Ext2fs to have excellent performance. We also wanted to provide a very robust filesystem in order to reduce the risk of data loss in intensive use. Last, but not least, Ext2fs had to include provision for extensions to allow users to benefit from new features without reformatting their filesystem.

``Standard'' Ext2fs features

The Ext2fs supports standard Unix file types: regular files, directories, device special files and symbolic links.

Ext2fs is able to manage filesystems created on really big partitions. While the original kernel code restricted the maximal filesystem size to 2 GB, recent work in the VFS layer have raised this limit to 4 TB. Thus, it is now possible to use big disks without the need of creating many partitions.

Ext2fs provides long file names. It uses variable length directory entries. The maximal file name size is 255 characters. This limit could be extended to 1012 if needed.

Ext2fs reserves some blocks for the super user (`root`). Normally, 5% of the blocks are reserved. This allows the administrator to recover easily from situations where user processes fill up filesystems.

``Advanced'' Ext2fs features

In addition to the standard Unix features, Ext2fs supports some extensions which are not usually present in Unix filesystems.

File attributes allow the users to modify the kernel behavior when acting on a set of files. One can set attributes on a file or on a directory. In the later case, new files created in the directory inherit these attributes.

BSD or System V Release 4 semantics can be selected at mount time. A mount option allows the administrator to choose the file creation semantics. On a filesystem mounted with BSD semantics, files are created with the same group id as their parent directory. System V semantics are a bit more complex: if a directory has the setgid bit set, new files inherit the group id of the directory and subdirectories inherit the group id and the setgid bit; in the other case, files and subdirectories are created with the primary group id of the calling process.

BSD-like synchronous updates can be used in Ext2fs. A mount option allows the administrator to request that metadata (inodes, bitmap blocks, indirect blocks and directory blocks) be written synchronously on the disk when they are modified. This can be useful to maintain a strict metadata consistency but this leads to poor performances. Actually, this feature is not normally used, since in addition to the performance loss associated with using synchronous updates of the metadata, it can cause corruption in the user data which will not be flagged by the filesystem checker.

Ext2fs allows the administrator to choose the logical block size when creating the filesystem. Block sizes can typically be 1024, 2048 and 4096 bytes. Using big block sizes can speed up I/O since fewer I/O requests, and thus fewer disk head seeks, need to be done to access a file. On the other hand, big blocks waste more disk space: on the average, the last block allocated to a file is only half full, so as blocks get bigger, more space is wasted in the last block of each file. In addition, most of the advantages of larger block sizes are obtained by Ext2 filesystem's preallocation techniques (see section Performance optimizations.

Ext2fs implements fast symbolic links. A fast symbolic link does not use any data block on the filesystem. The target name is not stored in a data block but in the inode itself. This policy can save some disk space (no data block needs to be allocated) and speeds up link operations (there is no need to read a data block when accessing such a link). Of course, the space available in the inode is limited so not every link can be implemented as a fast symbolic link. The maximal size of the target name in a fast symbolic link is 60 characters. We plan to extend this scheme to small files in the near future.

Ext2fs keeps track of the filesystem state. A special field in the superblock is used by the kernel code to indicate the status of the file system. When a filesystem is mounted in read/write mode, its state is set to ``Not Clean''. When it is unmounted or remounted in read-only mode, its state is reset to ``Clean''. At boot time, the filesystem checker uses this information to decide if a filesystem must be checked. The kernel code also records errors in this field. When an inconsistency is detected by the kernel code, the filesystem is marked as ``Erroneous''. The filesystem checker tests this to force the check of the filesystem regardless of its apparently clean state.

Always skipping filesystem checks may sometimes be dangerous, so Ext2fs provides two ways to force checks at regular intervals. A mount counter is maintained in the superblock. Each time the filesystem is mounted in read/write mode, this counter is incremented. When it reaches a maximal value (also recorded in the superblock), the filesystem checker forces the check even if the filesystem is ``Clean''. A last check time and a maximal check interval are also maintained in the superblock. These two fields allow the administrator to request periodical checks. When the maximal check interval has been reached, the checker ignores the filesystem state and forces a filesystem check. Ext2fs offers tools to tune the filesystem behavior. The `tune2fs` program can be used to modify:

- the error behavior. When an inconsistency is detected by the kernel code, the filesystem is marked as ``Erroneous'' and one of the three following actions can be done: continue normal execution, remount the filesystem in read-only mode to avoid corrupting the filesystem, make the kernel panic and reboot to run the filesystem checker.
- the maximal mount count.
- the maximal check interval.
- the number of logical blocks reserved for the super user.

Mount options can also be used to change the kernel error behavior.

An attribute allows the users to request secure deletion on files. When such a file is deleted, random data is written in the disk blocks previously allocated to the file. This prevents malicious people from gaining access to the previous content of the file by using a disk editor.

Last, new types of files inspired from the 4.4 BSD filesystem have recently been added to Ext2fs. Immutable files can only be read: nobody can write or delete them. This can be used to protect sensitive configuration files. Append-only files can be opened in write mode but data is always appended at the end of the file. Like immutable files, they cannot be deleted or renamed. This is especially useful for log files which can only grow.

Physical Structure

The physical structure of Ext2 filesystems has been strongly influenced by the layout of the BSD filesystem [McKusick *et al.* 1984]. A filesystem is made up of block groups. Block groups are analogous to BSD FFS's cylinder groups. However, block groups are not tied to the physical layout of the blocks on the disk, since modern drives tend to be optimized for sequential access and hide their physical geometry to the operating system.

The physical structure of a filesystem is represented in this table:

Boot Sector	Block Group 1	Block Group 2	... ...	Block Group N

Each block group contains a redundant copy of crucial filesystem control informations (superblock and the filesystem descriptors) and also contains a part of the filesystem (a block bitmap, an inode bitmap, a piece of the inode table, and data blocks). The structure of a block group is represented in this table:

Super Block	FS descriptors	Block Bitmap	Inode Bitmap	Inode Table	Data Blocks

Using block groups is a big win in terms of reliability: since the control structures are replicated in each block group, it is easy to recover from a filesystem where the superblock has been corrupted. This structure also helps to get good performances: by reducing the distance between the inode table and the data blocks, it is possible to reduce the disk head seeks during I/O on files.

In Ext2fs, directories are managed as linked lists of variable length entries. Each entry contains the inode number, the entry length, the file name and its length. By using variable length entries, it is possible to implement long file names without wasting disk space in directories. The structure of a directory entry is shown in this table:

inode number	entry length	name length	filename

As an example, The next table represents the structure of a directory containing three files: `file1`, `long_file_name`, and `f2`:

i1	16	05	`file1`
i2	40	14	`long_file_name`
i3	12	02	`f2`

Performance optimizations

The Ext2fs kernel code contains many performance optimizations, which tend to improve I/O speed when reading and writing files.

Ext2fs takes advantage of the buffer cache management by performing readaheads: when a block has to be read, the kernel code requests the I/O on several contiguous blocks. This way, it tries to ensure that the next block to read will already be loaded into the buffer cache. Readaheads are normally performed during sequential reads on files and Ext2fs extends them to directory reads, either explicit reads (`readdir(2)` calls) or implicit ones (`namei` kernel directory lookup).

Ext2fs also contains many allocation optimizations. Block groups are used to cluster together related inodes and data: the kernel code always tries to allocate data blocks for a file in the same group as its inode. This is intended to reduce the disk head seeks made when the kernel reads an inode and its data blocks.

When writing data to a file, Ext2fs preallocates up to 8 adjacent blocks when allocating a new block. Preallocation hit rates are around 75% even on very full filesystems. This preallocation achieves good write performances under heavy load. It also allows contiguous blocks to be allocated to files, thus it speeds up the future sequential reads.

These two allocation optimizations produce a very good locality of:

- related files through block groups
- related blocks through the 8 bits clustering of block allocations.

The Ext2fs library

To allow user mode programs to manipulate the control structures of an Ext2 filesystem, the libext2fs library was developed. This library provides routines which can be used to examine and modify the data of an Ext2 filesystem, by accessing the filesystem directly through the physical device.

The Ext2fs library was designed to allow maximal code reuse through the use of software abstraction techniques. For example, several different iterators are provided. A program can simply pass in a function to `ext2fs_block_interate()`, which will be called for each block in an inode. Another iterator function allows an user-provided function to be called for each file in a directory.

Many of the Ext2fs utilities (`mke2fs`, `e2fsck`, `tune2fs`, `dumpe2fs`, and `debugfs`) use the Ext2fs library. This greatly simplifies the maintainance of these utilities, since any changes to reflect new features in the Ext2 filesystem format need only be made in one place--in the Ext2fs library. This code reuse also results in smaller binaries, since the Ext2fs library can be built as a shared library image.

Because the interfaces of the Ext2fs library are so abstract and general, new programs which require direct access to the Ext2fs filesystem can very easily be written. For example, the Ext2fs library was used during the port of the 4.4BSD dump and restore backup utilities. Very few changes were needed to adapt these tools to Linux: only a few filesystem dependent functions had to be replaced by calls to the Ext2fs library.

The Ext2fs library provides access to several classes of operations. The first class are the filesystem-oriented operations. A program can open and close a filesystem, read and write the bitmaps, and create a new filesystem on the disk. Functions are also available to manipulate the filesystem's bad blocks list.

The second class of operations affect directories. A caller of the Ext2fs library can create and expand directories, as well as add and remove directory entries. Functions are also provided to both resolve a pathname to an inode number, and to determine a pathname of an inode given its inode number.

The final class of operations are oriented around inodes. It is possible to scan the inode table, read and write inodes, and scan through all of the blocks in an inode. Allocation and deallocation routines are also available and allow user mode programs to allocate and free blocks and inodes.

The Ext2fs tools

Powerful management tools have been developed for Ext2fs. These utilities are used to create, modify, and correct any inconsistencies in Ext2 filesystems. The `mke2fs` program is used to initialize a partition to contain an empty Ext2 filesystem.

The `tune2fs` program can be used to modify the filesystem parameters. As explained in section ``Advanced'' Ext2fs features, it can change the error behavior, the maximal mount count, the maximal check interval, and the number of logical blocks reserved for the super user.

The most interesting tool is probably the filesystem checker. `E2fsck` is intended to repair filesystem inconsistencies after an unclean shutdown of the system. The original version of `e2fsck` was based on Linus Torvald's fsck program for the Minix filesystem. However, the current version of `e2fsck` was rewritten from scratch, using the Ext2fs library, and is much faster and can correct more filesystem inconsistencies than the original version.

The `e2fsck` program is designed to run as quickly as possible. Since filesystem checkers tend to be disk bound, this was done by optimizing the algorithms used by `e2fsck` so that filesystem structures are not repeatedly accessed from the disk. In addition, the order in which inodes and directories are checked are sorted by block number to reduce the amount of time in disk seeks. Many of these ideas were originally explored by [Bina and Emrath 1989] although they have since been further refined by the authors.

In pass 1, `e2fsck` iterates over all of the inodes in the filesystem and performs checks over each inode as an unconnected object in the filesystem. That is, these checks do not require any cross-checks to other filesystem objects. Examples of such checks include making sure the file mode is legal, and that all of the blocks in the inode are valid block numbers. During pass 1, bitmaps indicating which blocks and inodes are in use are compiled.

If `e2fsck` notices data blocks which are claimed by more than one inode, it invokes passes 1B through 1D to resolve these conflicts, either by cloning the shared blocks so that each inode has its own copy of the shared block, or by deallocating one or more of the inodes.

Pass 1 takes the longest time to execute, since all of the inodes have to be read into memory and checked. To reduce the I/O time necessary in future passes, critical filesystem information is cached in memory. The most important example of this technique is the location on disk of all of the directory blocks on the filesystem. This obviates the need to re-read the directory inodes structures during pass 2 to obtain this information.

Pass 2 checks directories as unconnected objects. Since directory entries do not span disk blocks, each directory block can be checked individually without reference to other directory blocks. This allows `e2fsck` to sort all of the directory blocks by block number, and check directory blocks in ascending order, thus decreasing disk seek time. The directory blocks are checked to make sure that the directory entries are valid, and contain references to inode numbers which are in use (as determined by pass 1).

For the first directory block in each directory inode, the `.' and `..' entries are checked to make sure they exist, and that the inode number for the `.' entry matches the current directory. (The inode number for the `..' entry is not checked until pass 3.)

Pass 2 also caches information concerning the parent directory in which each directory is linked. (If a directory is referenced by more than one directory, the second reference of the directory is treated as an illegal hard link, and it is removed).

It is noteworthy to note that at the end of pass 2, nearly all of the disk I/O which `e2fsck` needs to perform is complete. Information required by passes 3, 4 and 5 are cached in memory; hence, the remaining passes of `e2fsck` are largely CPU bound, and take less than 5-10% of the total running time of `e2fsck`.

In pass 3, the directory connectivity is checked. `E2fsck` traces the path of each directory back to the root, using information that was cached during pass 2. At this time, the `..' entry for each directory is also checked to make sure it is valid. Any directories which can not be traced back to the root are linked to the `/lost+found` directory.

In pass 4, `e2fsck` checks the reference counts for all inodes, by iterating over all the inodes and comparing the link counts (which were cached in pass 1) against internal counters computed during passes 2 and 3. Any undeleted files with a zero link count is also linked to the `/lost+found` directory during this pass.

Finally, in pass 5, `e2fsck` checks the validity of the filesystem summary information. It compares the block and inode bitmaps which were constructed during the previous passes against the actual bitmaps on the filesystem, and corrects the on-disk copies if necessary.

The filesystem debugger is another useful tool. `Debugfs` is a powerful program which can be used to examine and change the state of a filesystem. Basically, it provides an interactive interface to the Ext2fs library: commands typed by the user are translated into calls to the library routines.

`Debugfs` can be used to examine the internal structures of a filesystem, manually repair a corrupted filesystem, or create test cases for `e2fsck`. Unfortunately, this program can be dangerous if it is used by people who do not know what they are doing; it is very easy to destroy a filesystem with this tool. For this reason, `debugfs` opens filesytems for read-only access by default. The user must explicitly specify the `-w` flag in order to use `debugfs` to open a filesystem for read/wite access.

Performance Measurements

Description of the benchmarks

We have run benchmarks to measure filesystem performances. Benchmarks have been made on a middle-end PC, based on a i486DX2 processor, using 16 MB of memory and two 420 MB IDE disks. The tests were run on Ext2 fs and Xia fs (Linux 1.1.62) and on the BSD Fast filesystem in asynchronous and synchronous mode (FreeBSD 2.0 Alpha--based on the 4.4BSD Lite distribution).

We have run two different benchmarks. The Bonnie benchmark tests I/O speed on a big file--the file size was set to 60 MB during the tests. It writes data to the file using character based I/O, rewrites the contents of the whole file, writes data using block based I/O, reads the file using character I/O and block I/O, and seeks into the file. The Andrew Benchmark was developed at Carneggie Mellon University and has been used at the University of Berkeley to benchmark BSD FFS and LFS. It runs in five phases: it creates a directory hierarchy, makes a copy of the data, recursively examine the status of every file, examine every byte of every file, and compile several of the files.

Results of the Bonnie benchmark

The results of the Bonnie benchmark are presented in this table:

	Char Write (KB/s)	**Block Write (KB/s)**	**Rewrite (KB/s)**	**Char Read (KB/s)**	**Block Read (KB/s)**
BSD Async	710	684	401	721	888
BSD Sync	699	677	400	710	878
Ext2 fs	452	1237	536	397	1033
Xia fs	440	704	380	366	895

The results are very good in block oriented I/O: Ext2 fs outperforms other filesystems. This is clearly a benefit of the optimizations included in the allocation routines. Writes are fast because data is written in cluster mode. Reads are fast because contiguous blocks have been allocated to the file. Thus there is no head seek between two reads and the readahead optimizations can be fully used.

On the other hand, performance is better in the FreeBSD operating system in character oriented I/O. This is probably due to the fact that FreeBSD and Linux do not use the same stdio routines in their respective C libraries. It seems that FreeBSD has a more optimized character I/O library and its performance is better.

Results of the Andrew benchmark

The results of the Andrew benchmark are presented in this table:

	P1 Create (ms)	**P2 Copy (ms)**	**P3 Stat (ms)**	**P4 Grep (ms)**	**P5 Compile (ms)**
BSD Async	2203	7391	6319	17466	75314
BSD Sync	2330	7732	6317	17499	75681
Ext2 fs	790	4791	7235	11685	63210
Xia fs	934	5402	8400	12912	66997

The results of the two first passes show that Linux benefits from its asynchronous metadata I/O. In passes 1 and 2, directories and files are created and BSD synchronously writes inodes and directory entries. There is an anomaly, though: even in asynchronous mode, the performance under BSD is poor. We suspect that the asynchronous support under FreeBSD is not fully implemented.

In pass 3, the Linux and BSD times are very similar. This is a big progress against the same benchmark run six months ago. While BSD used to outperform Linux by a factor of 3 in this test, the addition of a file name cache in the VFS has fixed this performance problem.

In passes 4 and 5, Linux is faster than FreeBSD mainly because it uses an unified buffer cache management. The buffer cache space can grow when needed and use more memory than the one in FreeBSD, which uses a fixed size buffer cache. Comparison of the Ext2fs and Xiafs results shows that the optimizations included in Ext2fs are really useful: the performance gain between Ext2fs and Xiafs is around 5-10%.

Conclusion

The Second Extended File System is probably the most widely used filesystem in the Linux community. It provides standard Unix file semantics and advanced features. Moreover, thanks to the optimizations included in the kernel code, it is robust and offers excellent performance.

Since Ext2fs has been designed with evolution in mind, it contains hooks that can be used to add new features. Some people are working on extensions to the current filesystem: access control lists conforming to the Posix semantics [IEEE 1992], undelete, and on-the-fly file compression.

Ext2fs was first developed and integrated in the Linux kernel and is now actively being ported to other operating systems. An Ext2fs server running on top of the GNU Hurd has been implemented. People are also working on an Ext2fs port in the LITES server, running on top of the Mach microkernel [Accetta *et al.* 1986], and in the VSTa operating system. Last, but not least, Ext2fs is an important part of the Masix operating system [Card *et al.* 1993], currently under development by one of the authors.

Acknowledgments

The Ext2fs kernel code and tools have been written mostly by the authors of this paper. Some other people have also contributed to the development of Ext2fs either by suggesting new features or by sending patches. We want to thank these contributors for their help.

References

[Accetta *et al.* 1986] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A New Kernel Foundation For UNIX Development. In *Proceedings of the USENIX 1986 Summer Conference*, June 1986.

[Bach 1986] M. Bach. *The Design of the UNIX Operating System*. Prentice Hall, 1986.

[Bina and Emrath 1989] E. Bina and P. Emrath. A Faster fsck for BSD Unix. In *Proceedings of the USENIX Winter Conference*, January 1989.

[Card *et al.* 1993] R. Card, E. Commelin, S. Dayras, and F. Mével. The MASIX Multi-Server Operating System. In *OSF Workshop on Microkernel Technology for Distributed Systems*, June 1993.

[IEEE 1992] *SECURITY INTERFACE for the Portable Operating System Interface for Computer Environments - Draft 13*. Institute of Electrical and Electronics Engineers, Inc, 1992.

[Kleiman 1986] S. Kleiman. Vnodes: An Architecture for Multiple File System Types in Sun UNIX. In *Proceedings of the Summer USENIX Conference*, pages 260--269, June 1986.

[McKusick *et al.* 1984] M. McKusick, W. Joy, S. Leffler, and R. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181--197, August 1984.

[Seltzer *et al.* 1993] M. Seltzer, K. Bostic, M. McKusick, and C. Staelin. An Implementation of a Log-Structured File System for UNIX. In *Proceedings of the USENIX Winter Conference,* January 1993.

[Tanenbaum 1987] A. Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, 1987.

Need /proc info

Forum: Filesystems
Keywords: linux filesystem /proc
Date: Wed, 04 Jun 1997 15:37:10 GMT
From: Kai Xu <kai@caip.rutgers.edu>

Hi,

Where can I find the detailed write-ups about the /proc file system in Linux. I want to know how it works and how it is implemented(not just the source code, I have it already)

Please reply to kai@caip.rutgers.edu , if you can.

Thanks.

Kai Xu

Where to find libext2 sources?

Forum: Filesystems
Date: Fri, 09 May 1997 14:27:59 GMT
From: Mark Salter <marks@qms.com>

```
The subject says it all.
```

Messages

1. Nevermind... *by Mark Salter*

Nevermind...

Forum: Filesystems
Re: Where to find libext2 sources? (Mark Salter)
Date: Fri, 09 May 1997 19:34:50 GMT
From: Mark Salter <*unknown*>

```
I found it. I didn't realize it was part of e2fsprogs.
```

New File System

Forum: Filesystems

Keywords: linux file system file_write memcpy fragment
Date: Fri, 18 Apr 1997 05:15:59 GMT
From: Vamsi Krishna <vamsi@cslab.uky.edu>

```
Hi,
        I am implementing a new file system, in Linux. I have
borrowed lot of code from Minix for this purpose. I had a
problem while implementing the function for file_write. I had
to modify this function a lot for supporting fragments. I
tried to use all these functions ( memcpy_tofs, memcpy_fromfs,
memcpy, bcopy, memmove) to copy the contents of the last
block into one or fragments. When I used memcpy_tofs or
memcpy_fromfs, I got segmentation faults. And when I used
the other functions, it only used to copy the first fragment
properly and not the remaining ones. Could anyone please help
me in this regard.
```

Thank you Vamsi

Partition?

Forum: Filesystems

Keywords: Partition
Date: Wed, 26 Mar 1997 03:05:49 GMT

From: Wilfredo Lugo Beauchamp <ak47@amadeus.upr.clu.edu>

```
        Hi,  I'm working in some Linux ext2 filesystem
applications and I need to read an inode.  The problem is
that the function iget(.....) needs the superblock.  Is it
possible get the superblock without the partition info?  Are
there any functions that return the current partition?
                Thanks for your attention
                        Wilfredo Lugo
```

Messages

1. Please be more specific.... *by Theodore Ts'o*

Please be more specific....

Forum: Filesystems
Re: Partition? (Wilfredo Lugo Beauchamp)
Keywords: ext2 filesystem
Date: Wed, 26 Mar 1997 04:28:53 GMT
From: Theodore Ts'o <tytso@mit.edu>

It's not clear from your question whether you are trying to write a user mode application, or trying to write kernel code. Parts of your question imply that you're writing user-mode code, but iget() is a kernel routine which isn't available to user-mode programs.

Why don't you be a bit more specific about what you're trying to do, and perhaps we can help you out. Assuming that you're writing a user-mode application, are you trying to read the filesystem directly using the device file, and using direct I/O to the device? Or are you trying to get some information from a filesystem that is already mounted?

Why do you need to read an inode? What are you trying to do with it?

Need documentation on userfs implementation

Forum: Filesystems
Keywords: userfs ftpfs
Date: Thu, 13 Feb 1997 10:08:41 GMT
From: Natchu Vishnu Priya <vishnu@cs.iitm.ernet.in>

I need documentation on userfs, in linux.. I seem to be able to find only an alpha version of userfs available and the ftpfs in that does not work.

A list of function, etc.. on the lines of those on the vfs, might be helpful.

-vishnu

ext2fs tools

Forum: Filesystems
Keywords: ext2fs tools
Date: Wed, 05 Feb 1997 00:42:12 GMT
From: Wilfredo Lugo Beauchamp <ak47@amadeus.upr.clu.edu>

```
        Hi, I would like to know where I can find the set of
tools e2fsprogs.  I'm working on an undelete for Linux for
my Operating Systems course.  I read a document prepared by
Mr. Linus Tauro of the University of California and he used
these tools.

                Wilfredo Lugo
```

Messages

1. Where to find e2fsprogs *by Theodore Ts'o*

Where to find e2fsprogs

Forum: Filesystems
Re: ext2fs tools (Wilfredo Lugo Beauchamp)
Keywords: ext2fs tools
Date: Wed, 05 Feb 1997 15:46:52 GMT
From: Theodore Ts'o <tytso@mit.edu>

The website for for the e2fsprogs package can be found at

`http://web.mit.edu/tytso/www/linux/e2fsprogs.html`

You can also ftp it from tsx-11.mit.edu, in the /pub/linux/packages/ext2fs directory.

libext2fs documentation

Forum: Filesystems
Keywords: libext2fs filesystem
Date: Wed, 15 Jan 1997 18:07:22 GMT
From: James Beckett <jmb@isltd.insignia.com>

After a repartition and (win95) reformat I find I didn't save away all the data I wanted from an ext2 fs, so I've spent a morning grovelling through the source and figuring out the structure. (I think I can get the data back, only the first block group got overwritten by format)

Now I find that libext2fs exists.. is there any documentation on how to use it, and how much does it depend on the filesystem being intact? Can it be told to use a backup superblock? I discovered that mount(8) can be given an option to do so, but the utilities (e2fsck, debugfs etc) don't seem to, so is it some limitation of libext2fs?

Messages

1. libext2fs documentation *by Theodore Ts'o*

libext2fs documentation

Forum: Filesystems
Re: libext2fs documentation (James Beckett)
Keywords: libext2fs filesystem
Date: Wed, 22 Jan 1997 23:20:07 GMT
From: Theodore Ts'o <tytso@mit.edu>

No, there currently isnt any documentation on the libext2fs library. The library is relatively well structured internally, and so most people who have looked at it haven't had *too* much trouble figuring it out.

It would be nice to have some documentation on it, though, and I am soliciting volunteers who would be willing to do a first pass documentation on it; I'm definitely willing to work with someone who is interested in doing that sort of tech writing.

As for your question, you can absolutely tell it to use a backup superblock, just take a look at the source code for the function signature for the ext2fs_open() function. One of the arguments is "superblock", and that's the block number for the superblock. You're right that debugfs currently doesn't have a method for opening the filesystem with a backup superblock. E2fsck most certainly does have a way to do this, though, and it's documented in the man page. Try using "e2fsck -b 8193".

proc filesystem

Forum: Filesystems
Keywords: proc filesystem
Date: Fri, 18 Oct 1996 21:18:40 GMT
From: Praveen Krishnan <praveen@kurinji.iitm.ernet.in>

Hello,

Isn't there any documentation on the /proc filesystem ? There was a chapter on it in the earlier versions of KHG but i dont see it here. Maybe i've missed it. if so ,could someone please direct me to where i would be able to get some programming related info on the /proc filesystem.

Thanx a lot

- praveen

Messages

1. man proc *by Michael K. Johnson*

man proc

Forum: Filesystems
Re: proc filesystem (Praveen Krishnan)
Keywords: proc filesystem
Date: Thu, 24 Oct 1996 23:20:10 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

man proc

The proc chapter was removed because the man page was more complete and more up-to-date.

Need NFS documentation

Forum: Filesystems
Keywords: NFS
Date: Fri, 27 Sep 1996 11:22:14 GMT
From: Ermelindo Mauriello <ermmau@ikonos.dia.unisa.it>

I'm working on a Transparent Cryptographic Filesystem for Linux based on the NFS concept. Documentation about this filesystem can be found ad mikonos.dia.unisa.it or www.globenet.it. Actually this filesystem is out of the kernel but the project is to push it into the system using nfs module. So I need documentation about NFS module. Where can I find it ? Thanks in advance to all will help me. ermmau@mikonos.dia.unisa.it

Even more ext2 documentation!

Forum: Filesystems
Keywords: info
Date: Wed, 12 Jun 1996 16:51:45 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Even more documentation on ext2fs is available. The ext2ed (available via ftp from tsx-11.mit.edu in /pub/linux/packages/ext2fs) contains a set of detailed papers on ext2fs, including an overview, a design document, and a users guide for ext2ed.

Also, an Analysis of the Ext2fs structure is available.

More ext2 documentation

Forum: Filesystems
Keywords: ext2fs
Date: Mon, 03 Jun 1996 22:15:17 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Remy Card recently announced that he has made postscript versions of the slides which he prepared for the 3rd International Linux Conference in Berlin available for ftp at tsx-11.mit.edu:/pub/linux/packages/ext2fs/slides/berlin96

One talk was on quota management for ext2fs, and the other was on the implementation of POSIX.6 ACL's for ext2fs. Four sets of slides are available: two-up slides on quotas, one-up slides on quotas, two-up slides on ACL's, and one-up slides on ACL's.

Ext2 paper

Forum: Filesystems
Keywords: ext2 filesystem
Date: Wed, 29 May 1996 21:02:45 GMT
From: Theodore Ts'o <tytso@mit.edu>

At one point, the ext2 paper which Remy, Stephen and I wrote was supposed to be going into the KHG. It was written for the Amsterdam Linux conference 1-2 years ago, but we got copyright clearance so that it could be included in the KHG. However, it seems that it never did get included into the KHG. Does anyone know what happened with that? I no longer have a copy of the original TeX, but Remy (as the primary author) should. I think it would be a very valuable addition to the KHG.

Messages

1. Done. *by Michael K. Johnson*

Done.

Forum: Filesystems
Re: Ext2 paper (Theodore Ts'o)
Keywords: ext2 filesystem
Date: Wed, 12 Jun 1996 16:11:41 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

See above! Thanks, Ted et. al.

Linux Memory Management

---

1. The Players

The TLB.

This is more of a virtual entity than a strict model as far as the Linux flush architecture is concerned. The only characteristics it has is:

1. It keeps track of process/kernel mappings in some way, whether in software or hardware.
2. Architecture specific code may need to be notified when the kernel has changed a process/kernel mapping.

The cache.

This entity is essentially "memory state" as the flush architecture views it. In general it has the following properties:

1. It will always hold copies of data which will be viewed as uptodate by the local processor.
2. Its proper functioning may be related to the TLB and process/kernel page mappings in some way, that is to say they may depend upon each other.
3. It may, in a virtually cached configuration, cause aliasing problems if one physical page is mapped at the same time to two virtual pages, and due to to the bits of an address used to index the cache line, the same piece of data can end up residing in the cache twice, allowing inconsistancies to result.
4. Devices and DMA may or may not be able to see the most up to date copy of a piece of data which resides in the cache of the local processor.
5. Currently, it is assumed that coherence in a multiprocessor environment is maintained by the cache/memory subsystem. That is to say, when one processor requests a datum on the memory bus and another processor has a more uptodate copy, by whatever means the requestor will get the uptodate copy owned by the other processor.

(NOTE: SMP architectures without hardware cache coherence mechanisms are indeed possible, the current flush architecture does not handle this currently. If at at some point a Linux port to some system where this is an issue occurrs, I will add the necessary hooks. But it will not be pretty.)

2. What the flush architecture cares about

1. At all times the memory management hardware's view of a set of process/kernel mappings will be consistant with that of the kernel page tables.
2. If the memory management kernel code makes a modification to a user process page, by modifying the data via the kernel-space alias of the underlying physical page, the user thread of control will see the right data before it is allowed to continue execution, regardless of the cache architecture and/or semantics.
3. In general, when address space state is changed (on the generic kernel memory management code's behalf **only**) the appropriate flush architecture hook will be called describing that state change in full.

3. What the flush architecture does not care about

1. DMA/Driver coherency. This includes DMA mappings (in the sense of MMU mappings) and cache/DMA datum consistency. These sorts of issues have no buisness in the flush architecture, see below how they should be handled.
2. Split Instruction/Data cache consistancy with respect to modifications made to the process instruction space performed by the signal dispatch code. Again see below on how this should be handled in another way.

4. The interfaces for the flush architecture and how to implement them

In general all of the routines described below will be called with the following sequence:

```
        flush_cache_foo(...);
        modify_address_space();
        flush_tlb_foo(...);
```

The logic here is:

1. It may be illegal in a given architecture for a piece of cache data to exist when no mapping for that data exists, therefore the flush must occur before the change is made.
2. It is possible for a given MMU/TLB architecture to perform a hardware table walk of the kernel page tables. Therefore the TLB flush is done after the page tables have been changed so that afterwards the hardware can only load in the new copy of the page table information to the TLB.

`void flush_cache_all(void);`
`void flush_tlb_all(void);`

These routines are to notify the architecture specific code that a change has been made to the kernel address space mappings, which means that the mappings of every process has effectively changed.

An implementation shall:

1. Eliminate all cache entries which are valid at this point in time when `flush_cache_all` is invoked. This applies to virtual cache architectures. If the cache is write-back in nature, this routine shall commit the cache data to memory before invalidating each entry. For physical caches, no action need be performed since physical mappings have no bearing on address space translations.
2. For `flush_tlb_all`, all TLB mappings for the kernel address space should be made consistant with the OS page tables by whatever means necessary. Note that with an architecture that possesses the notion of "MMU/TLB contexts" it may be necessary to perform this synchronization in every "active" MMU/TLB context.

`void flush_cache_mm(struct mm_struct *mm);`
`void flush_tlb_mm(struct mm_struct *mm);`

These routines notify the system that the entire address space described by the `mm_struct` passed is changing. Please take note of two things in particular:

1. The `mm_struct` is the unit of mmu/tlb real estate as far as the flush architecture is concerned. In particular, an `mm_struct` may map to one or many tasks or none!
2. This "address space" change is considered to be occurring in user space only. It is therefore safe for code to avoid flushing kernel tlb/cache entries if that is possible for efficiency.

An implementation shall:

1. For `flush_cache_mm`, whatever entries could exist in a virtual cache for the address space described by `mm_struct` are to be invalidated.
2. For `flush_tlb_mm`, the tlb/mmu hardware is to be placed in a state where it will see the (now current) kernel page table entries for the address space described by the `mm_struct`.

```
flush_cache_range(struct mm_struct *mm, unsigned long start,
                  unsigned long end);
flush_tlb_range(struct mm_struct *mm, unsigned long start,
                unsigned long end);
```

A change to a particular range of user addresses in the address space described by the `mm_struct` passed is occurring. The two notes above for `flush_*_mm()` concerning the `mm_struct` passed apply here as well.

An implementation shall:

1. For `flush_cache_range`, on a virtually cached system, all cache entries which are valid for the range start to end in the address space described by the `mm_struct` are to be invalidated.
2. For `flush_tlb_range`, whatever actions necessary to cause the MMU/TLB hardware to not contain stale translations are to be performed. This means that whatever translations are in the kernel page tables in the range start to end in the address space described by the `mm_struct` are to be what the memory mangement hardware will see from this point forward, by whatever means.

`void flush_cache_page(struct vm_area_struct *vma, unsigned long address);`
`void flush_tlb_page(struct vm_area_struct *vma, unsigned long address);`

A change to a single page at `address` within user space to the address space described by the `vm_area_struct` passed is occurring. An implementation, if need be, can get at the assosciated `mm_struct` for this address space via `vma->vm_mm`. The VMA is passed for convenience so that an implementation can inspect `vma->vm_flags`. This way in an implementation where the instruction and data spaces are not unified, one can check to see if `VM_EXEC` is set in `vma->vm_flags` to possibly avoid flushing the instruction space, for example.

The two notes above for `flush_*_mm()` concerning the `mm_struct` (passed indirectly via `vma->vm_mm`) apply here as well.

An implementation shall:

1. For `flush_cache_range`, on a virtually cached system, all cache entries which are valid for the page at `address` in the address space described by the VMA are to be invalidated.
2. For `flush_tlb_range`, whatever actions necessary to cause the MMU/TLB hardware to not contain stale translations are to be performed. This means that whatever translations are in the kernel page tables for the page at `address` in the address space described by the VMA passed are to be what the memory mangement hardware will see from this point forward, by whatever means.

`void flush_page_to_ram(unsigned long page);`

This is the ugly duckling. But its semantics are necessary on so many architectures that I needed to add it to the flush architecture for Linux.

Briefly, when (as one example) the kernel services a COW fault, it uses the aliased mappings of all physical memory in kernel space to perform the copy of the page in question to a new page. This presents a problem for virtually indexed caches which are write-back in nature. In this case, the kernel touches two physical pages in kernel space. The code sequence being described here essentially looks like:

```
do_wp_page()
{
        [ ... ]
                copy_cow_page(old_page,new_page);
                flush_page_to_ram(old_page);
                flush_page_to_ram(new_page);
                flush_cache_page(vma, address);
                modify_address_space();
                free_page(old_page);
                flush_tlb_page(vma, address);
        [ ... ]
}
```

(Some of the actual code has been simplified for example purposes.)

Consider a virtually indexed cache which is write-back. At the point in time at which the copy of the page occurs to the kernel space aliases, it is possible for the user space view of the original page to be in the caches (at the user's address, ie. where the fault is occurring). The page copy can bring this data (for the old page) into the caches. It will also place the data (at the new kernel aliased mapping of the page) being copied to into the cache, and for write back caches this data will be dirty or modified in the cache.

In such a case main memory will not see the most recent copy of the data. The caches are stupid, so for the new page we are giving to the user, without forcing the cached data at the kernel alias to main memory the process will see the old contents of the page (ie. whatever garbage was there before the copy done by COW processing above).

A concrete example of what was just described:

Consider a process which shares a page, read-only with another task (or many) at virtual address 0x2000 in user space. And for example purposes let us say that this virtual address maps to physical page 0x14000.

```
        Virtual Pages
task 1  --------------
        | 0x00000000 |
        --------------
        | 0x00001000 |                  Physical Pages
        --------------                  --------------
        | 0x00002000 | --\              | 0x00000000 |
        --------------    \             --------------
                           \            | ...        |
task 2  --------------      \           --------------
        | 0x00000000 |      |---->      | 0x00014000 |
        --------------      /           --------------
        | 0x00001000 |     /            | ...        |
        --------------    /             --------------
        | 0x00002000 | --/
        --------------
```

If task 2 tries to write to the read-only page at address 0x2000 we will get a fault and eventually end up at the code fragment shown above in `do_wp_page()`.

The kernel will get a new page for task2, let us say this is physical page 0x26000, and let us also say that the kernel alias mappings for physical pages 0x14000 and 0x26000 can reside in the two unique cache lines at the same time based upon the line indexing scheme of this cache.

The page contents get copied from the kernel mappings for physical page 0x14000 to the ones for physical page 0x26000.

At this point in time, on a write-back virtually indexed cache architecture we have a potential inconsistancy. The new data copied into physical page 0x26000 is not necessary in main memory at this point, in fact it could be all in the cache only at the kernel alias of the physical address. Also, the (non-modified, ie. clean) data for the original (old) page is in the cache at the kernel alias for physical page 0x14000, this can produce an inconsistancy later on, so to be safe it is best to be eliminate the cached copies of this data as well.

Let us say we did not write back the data for the page at 0x26000 and we let it just stay there. We would return to task 2 (who has this new page now mapped in at virtual address 0x2000), he would complete his write, then he would read some other piece of data in this new page (i.e. expecting the contents that existed there beforehand). At this point in time if the data is left in the cache at the kernel alias for the new physical page, the user will get whatever was in main memory before the copy for his read. This can lead to disasterous results.

Therefore an architecture shall:

On virtually indexed cache architectures, do whatever is necessary to make main memory consistant with the cached copy of the kernel space page passed.

NOTE: It is actually necessary for this routine to invalidate lines in a virtual cache which is not write-back in nature. To see why this is really necessary, replay the above example with task 1 and 2, but this time `fork()` yet another task 3 before the COW faults occur, consider the contents of the caches in both kernel and user space if the following sequence occurrs in exact succession:

1. task 1 reads some the page at 0x2000
2. task 2 COW faults the page at 0x2000
3. task 2 performs his writes to the new page at 0x2000
4. task 3 COW faults the page at 0x2000

Even on a non-writeback virtually indexed cache, task 3 can see inconsistant data after the COW fault if `flush_page_to_ram` does not invalidate the kernel aliased physical page from the cache.

```
void update_mmu_cache(struct vm_area_struct *vma,
                      unsigned long address, pte_t pte);
```

Although not strictly part of the flush architecture, on certain architectures some critical operations and checks need to be performed here for things to work out properly and for the system to remain consistant.

In particular, for virtually indexed caches this routine must check to see that the new mapping being added by the current page fault does not add an "bad alias" to user space.

A "bad alias" is defined as two or more mappings (at least one of which is writable) to two or more virtual pages which all translate to the same exact physical page, and due to the indexing algorithm of the cache can also reside in unique and mutually exclusive cache lines.

If such a "bad alias" is detected an implementation needs to resolve this inconsistancy some how, one solution is to walk through all of the mappings and change the page tables to make these pages as "non-cacheable" if the hardware allows such a thing.

The checks for this are very simple, all an implementation needs to do essentially is:

```
if((vma->vm_flags & (VM_WRITE|VM_SHARED)) == (VM_WRITE|VM_SHARED))
        check_for_potential_bad_aliases();
```

So for the common case (shared writable mappings are extremely rare) only one comparison is needed for systems with virtually indexed caches.

5. Implications for SMP

Depending upon the architecture certain amends may be needed to allow the flush architecture to work on an SMP system.

The main concern is whether one of the above flush operations cause the entire system to be globally see the flush, or the flush is only guarenteed to be seen by the local processor.

In the latter case a cross calling mechanism is needed. The current two SMP systems supported under Linux (Intel and Sparc) use inter-processor interrupts to "broadcast" the flush operation and cause it to run locally on all processors if necessary.

As an example, on sun4m Sparc systems all processers in the system must execute the flush request to guarentee consistancy across the entire system. However, on sun4d Sparc machines, TLB flushes performed on the local processor are broadcast over the system bus by the hardware and therefore a cross call is not necessary.

6. Implications for context based MMU/CACHE architectures

The entire idea behind the concept of MMU and cache context facilities is to allow many address spaces to share the cache/mmu resources on the cpu.

To take full advantage of such a facility, and still maintain coherency as described above, requires some extra consideration from the implementor.

The issues involved will vary greatly from one implementation to another, at least this has been the experience of the author. But in particular some of the issues are likely to be:

1. The relationship of kernel space mappings to user space ones, as far as contexts are concerned. On some systems kernel mappings have a "global" attribute, in that the hardware does not concern itself with context information when a translation is made which has this attribute. Therefore one flush (in any context) of a kernel cache/mmu mapping could be sufficient.
   However it is possible in other implementations for the kernel to share the context key assosciated with a particular address space. It may be necessary in such a case to walk into all contexts which are currently valid and perform the complete flush in each one for a kernel address space flush.
2. The cost of per-context flushes can become a key issue, especially with respect to the TLB. For example, if a tlb flush is needed on a large range of addresses (or an entire address space) it may be more prudent to allocate and assign a new mmu context to this process for the sake of efficiency.

7. How to handle what the flush architecture does not do, with examples

The flush architecture just described make no amends for device/DMA coherency with cached data. It also has no provisions for any mapping strategies necessary for DMA and devices should that be necessary on a certain machine Linux is ported to. Such issues are none of the flush architectures buisness.

Such issues are most cleanly dealt with at the device driver level. The author is convinced of this after his experiance with a common set of Sparc device drivers which needed to all function correctly on more than a handfull of cache/mmu and bus architecrures in the **same** kernel.

In fact this implementation is more efficient because the driver knows exactly when DMA needs to see consistant data or when DMA is going to create an inconsistancy which must be resolved. Any attempt to reach this level of efficiency via hooks added to the generic kernel memory management code would be complex and if anything very unclean.

As an example, consider on the Sparc how DMA buffers are handled. When a device driver must perform DMA to/from either a single buffer or a scatter list of many buffers it uses a set of abstract routines:

```
char *(*mmu_get_scsi_one)(char *, unsigned long, struct linux_sbus *sbus);
void  (*mmu_get_scsi_sgl)(struct mmu_sglist *, int, struct linux_sbus *sbus);
void  (*mmu_release_scsi_one)(char *, unsigned long, struct linux_sbus *sbus);
void  (*mmu_release_scsi_sgl)(struct mmu_sglist *, int, struct linux_sbus *sbus);
void  (*mmu_map_dma_area)(unsigned long addr, int len);
```

Essentially the `mmu_get_*` routines are passed a pointer or a set pointers and size specifications to areas in kernel space for which DMA will occur, they return a DMA capable address (i.e. one which can be loaded into the DMA controller for the transfer). When the driver is done with the DMA and the transfer has completed the `mmu_release_*` routines must be called with the DMA'able address(es) so that the resources can be freed (if necessary) and cache flushes can be performed (if necessary).

The final routine is there for drivers which need to have a block of DMA memory for a long period of time, for example a networking driver would use this for a pool transmit and receive buffers.

The final argument is a Sparc specific entity which allows the machine level code to perform the mapping if DMA mappings are setup on a per-BUS basis.

8. Open issues

There seems to be some very stupid cache architectures out there which want to cause trouble when an alias is placed into the cache (even a safe one where none of the aliased cache entries are writable!). Of note is the MIPS R4000 which will give an exception when such a situation occurs, these can occur when COW processing is happing in the current implementation. On most chips which do something stupid like this, the exception handler can flush the entries in the cache being complained about and all is well. The author is mostly concerned about the cost of these exceptions during COW processing and the effects this will have for system performance. Perhaps a new flush is neccessary, which would be performed before the page copy in COW fault processing, which essentially is to flush a user space page if not doing so would cause the trouble just described.

There has been heated talk lately about adding page flipping facilities for very intelligent networking hardware. It may be necessary to extend the flush architecture to provide the interfaces and facilities necessary for these changes to the networking code.

And by all means, the flush architecture is always subject to improvements and changes to handle new issues or new hardware which presents a problem that was to this point unknown.

```
David S. Miller
davem@caip.rutgers.edu
```

Linux Memory Management Overview

[Note: This overview of Linux's Memory Management is several years old. Linux's MM has gone through a nearly complete rewrite since this was written. However, if you can't understand the Linux MM code, reading this and understanding that this documents the predecessor to the current MM code *may* help you out.]

The Linux memory manager implements demand paging with a copy-on-write strategy relying on the 386's paging support. A process acquires its page tables from its parent (during a `fork()`) with the entries marked as read-only or swapped. Then, if the process tries to write to that memory space, and the page is a copy-on-write page, it is copied, and the page is marked read-write. An `exec()` results in the reading in of a page or so from the executable. The process then faults in any other pages it needs.

Each process has a page directory which means it can access 1 KB of page tables pointing to 1 MB of 4 KB pages which is 4 GB of memory. A process' page directory is initialized during a fork by `copy_page_tables()`. The idle process has its page directory initialized during the initialization sequence.

Each user process has a local descriptor table that contains a code segment and data-stack segment. These user segments extend from 0 to 3 GB (0xc0000000). In user space, linear addresses and logical addresses are identical.

On the 80386, linear address run from 0GB to 4GB. A linear address points to a particular memory location within this space. A linear address is **not** a physical address--it is a virtual address. A logical address consists of a selector and an offset. The selector points to a segment and the offset tells how far into that segment the address is located)

The kernel code and data segments are priveleged segments defined in the global descriptor table and extend from 3 GB to 4 GB. The swapper page directory (`swapper_page_dir` is set up so that logical addresses and physical addresses are identical in kernel space.

The space above 3 GB appears in a process' page directory as pointers to kernel page tables. This space is invisible to the process in user mode but the mapping becomes relevant when privileged mode is entered, for example, to handle a system call. Supervisor mode is entered within the context of the current process so address translation occurs with respect to the process' page directory but using kernel segments. This is identically the mapping produced by using the `swapper_pg_dir` and kernel segments as both page directories use the same page tables in this space. Only `task[0]` (the idle task, sometimes called the swapper task for historical reasons, even though it has nothing to do with swapping in the Linux implementation) uses the `swapper_pg_dir` directly.

- The user process' `segment_base` = 0x00, `page_dir` private to the process.
- user process makes a system call: `segment_base`=0xc0000000 `page_dir` = same user

page_dir.

- swapper_pg_dir contains a mapping for all physical pages from 0xc0000000 to 0xc0000000 + end_mem, so the first 768 entries in swapper_pg_dir are 0's, and then there are 4 or more that point to kernel page tables.
- The user page directories have the same entries as swapper_pg_dir above 768. The first 768 entries map the user space.

The upshot is that whenever the linear address is above 0xc0000000 everything uses the same kernel page tables.

The user stack sits at the top of the user data segment and grows down. The kernel stack is not a pretty data structure or segment that I can point to with a ``yon lies the kernel stack." A kernel_stack_frame (a page) is associated with each newly created process and is used whenever the kernel operates within the context of that process. Bad things would happen if the kernel stack were to grow below its current stack frame. **[Where is the kernel stack put? I know that there is one for every process, but where is it stored when it's not being used?]**

User pages can be stolen or swapped. A user page is one that is mapped below 3 GB in a user page table. This region does not contain page directories or page tables. Only dirty pages are swapped.

Minor alterations are needed in some places (tests for process memory limits comes to mind) to provide support for programmer defined segments.

[There is now a modify_ldt() system call used by dosemu, Wine, TWIN, and Wabi to create arbitrary segments.]

Physical memory

Here is a map of physical memory before any user processes are executed. The column on the left gives the **starting** address of the item, numbers in *italics* are approximate. The column in the middle names the item(s). The column on the far right gives the relevant routine or variable name or explains the entry.

0x110000	FREE	memory_end or high_memory
	mem_map	mem_init()
	inode_table	inode_init()
	device data	device_init()*
0x100000	more pg_tables	paging_init()
0x0A0000	RESERVED	
0x060000	FREE	
	low_memory_start	
0x006000	kernel code + data	

	floppy_track_buffer	
	bad_pg_table bad_page	used by page_fault_handlers to kill processes gracefully when out of memory.
0x002000	pg0	the first kernel page table.
0x001000	swapper_pg_dir	the kernel page directory.
0x000000	null page	

*device-inits that acquire memory are(main.c): profil_buffer, con_init, psaux_init, rd_init, scsi_dev_init.

Note that all memory not marked as FREE is RESERVED (mem_init). RESERVED pages belong to the kernel and are **never** freed or swapped.

A user process' view of memory

0xc0000000	The invisible kernel	reserved
	initial stack	
	room for stack growth	4 pages
0x60000000	shared libraries	
brk	unused	
	malloc memory	
end_data	uninitialized data	
end_code	initialized data	
0x00000000	text	

Both the code segment and data segment extend all the way from 0x00 to 3 GB. Currently the page fault handler do_wp_page checks to ensure that a process does not write to its code space. However, by catching the SEGV signal, it is possible to write to code space, causing a copy-on-write to occur. The handler do_no_page ensures that any new pages the process acquires belong to either the executable, a shared library, the stack, or lie within the brk value.

A user process can reset its brk value by calling sbrk(). This is what malloc() does when it needs to. The text and data portions are allocated on separate pages unless one chooses the -N compiler option. Shared library load addresses are currently taken from the shared image itself. The address is between 1.5 GB and 3 GB, except in special cases.

User process Memory Allocation

	swappable	**shareable**
a few code pages	Y	Y

a few data pages	Y	N?
stack	Y	N
`pg_dir`	N	N
code/data `page_table`	N	N
stack `page_table`	N	N
`task_struct`	N	N
`kernel_stack_frame`	N	N
shlib `page_table`	N	N
a few shlib pages	Y	Y?

[What do the question marks mean? Do they mean that they might go either way, or that you are not sure?]

The stack, shlibs and data are too far removed from each other to be spanned by one page table. All kernel `page_table`s are shared by all processes so they are not in the list. Only dirty pages are swapped. Clean pages are stolen so the process can read them back in from the executable if it likes. Mostly only clean pages are shared. A dirty page ends up shared across a fork until the parent or child chooses to write to it again.

Memory Management data in the process table

Here is a summary of some of the data kept in the process table which is used for memory managment:

Process memory limits
: `ulong start_code, end_code, end_data, brk, start_stack;`

Page fault counting
: `ulong min_flt, maj_flt, cmin_flt, cmaj_flt`

Local descriptor table
: `struct desc_struct ldt[32]` is the local descriptor table for task.

`rss`
: number of resident pages.

`swappable`
: if 0, then process's pages will not be swapped.

`kernel_stack_page`
: pointer to page allocated in fork.

`saved_kernel_stack`
: V86 mode stuff

`struct tss`

- Stack segments

  `esp0`
  : kernel stack pointer (`kernel_stack_page`)

  `ss0`
  : kernel stack segment (0x10)

esp1
: = ss1 = esp2 = ss2 = 0
  unused privelege levels.

- Segment selectors: ds = es = fs = gs = ss = 0x17, cs = 0x0f
  All point to segments in the current `ldt[]`.
- `cr3`: points to the page directory for this process.
- `ldt`: `_LDT(n)` selector for current task's LDT.

Memory initialization

In `start_kernel()` (main.c) there are 3 variables related to memory initialization:

`memory_start`	starts out at 1 MB. Updated by device initialization.
`memory_end`	end of physical memory: 8 MB, 16 MB, or whatever.
`low_memory_start`	end of the kernel code and data that is loaded initially.

Each device init typically takes `memory_start` and returns an updated value if it allocates space at `memory_start` (by simply grabbing it). `paging_init()` initializes the page tables in the {\tt swapper_pg_dir} (starting at 0xc0000000) to cover all of the physical memory from `memory_start` to `memory_end`. Actually the first 4 MB is done in `startup_32` (head.S). `memory_start` is incremented if any new `page_table`s are added. The first page is zeroed to trap null pointer references in the kernel.

In `sched_init()` the `ldt` and `tss` descriptors for `task[0]` are set in the GDT, and loaded into the TR and LDTR (the only time it's done explicitly). A trap gate (0x80) is set up for `system_call()`. The nested task flag is turned off in preparation for entering user mode. The timer is turned on. The `task_struct` for `task[0]` appears in its entirety in `<linux/sched.h>`.

`mem_map` is then constructed by `mem_init()` to reflect the current usage of physical pages. This is the state reflected in the physical memory map of the previous section.

Then Linux moves into user mode with an `iret` after pushing the current `ss`, `esp`, etc. Of course the user segments for `task[0]` are mapped right over the kernel segments so execution continues exactly where it left off.

`task[0]`:

pg_dir
: = `swapper_pg_dir` which means the the only addresses mapped are in the range 3 GB to 3 GB + `high_memory`.

LDT[1]
: = user code, base=0xc0000000, size = 640K

LDT[2]
: = user data, base=0xc0000000, size = 640K

The first `exec()` sets the LDT entries for `task[1]` to the user values of base = 0x0, limit = TASK_SIZE = 0xc0000000. Thereafter, no process sees the kernel segments while in user mode.

Processes and the Memory Manager

Memory-related work done by `fork()`:

- Memory allocation
  - 1 page for the `task_struct`.
  - 1 page for the kernel stack.
  - 1 for the `pg_dir` and some for `pg_table`s (copy_page_tables)
- Other changes
  - `ss0` set to kernel stack segment (0x10) to be sure?
  - `esp0` set to top of the newly allocated `kernel_stack_page`
  - `cr3` set by `copy_page_tables()` to point to newly allocated page directory.
  - `ldt = _LDT(task_nr)` creates new ldt descriptor.
  - descriptors set in gdt for new tss and `ldt[]`.
  - The remaining registers are inherited from parent.

The processes end up sharing their code and data segments (although they have separate local desctriptor tables, the entries point to the same segments). The stack and data pages will be copied when the parent or child writes to them (copy-on-write).

Memory-related work done by `exec()`:

- memory allocation
  - 1 page for exec header entire file for omagic
  - 1 page or more for stack (MAX_ARG_PAGES)
- `clear_page_tables()` used to remove old pages.
- `change_ldt()` sets the descriptors in the new `LDT[]`
- `ldt[1]` = code base=0x00, limit=TASK_SIZE
- `ldt[2]` = data base=0x00, limit=TASK_SIZE
  These segments are DPL=3, P=1, S=1, G=1. type=a (code) or 2 (data)
- Up to `MAX_ARG_PAGES` dirty pages of argv and envp are allocated and stashed at the top of the data segment for the newly created user stack.
- Set the instruction pointer of the caller `eip = ex.a_entry`
- Set the stack pointer of the caller to the stack just created (esp = stack pointer) These will be popped off the stack when the caller resumes.
- update memory limits
  ```
  end_code = ex.a_text
  end_data = end_code + ex.a_data
  brk = end_data + ex.a_bss
  ```

Interrupts and traps are handled within the context of the current task. In particular, the page directory of the current process is used in address translation. The segments, however, are kernel segments so that all linear addresses point into kernel memory. For example, assume a user process invokes a system call and the kernel wants to access a variable at address 0x01. The linear address is 0xc0000001 (using kernel segments) and the physical address is 0x01. The later is because the process' page directory maps this range exactly as `page_pg_dir`.

The kernel space (0xc0000000 + `high_memory`) is mapped by the kernel page tables which are themselves part of the RESERVED memory. They are therefore shared by all processes. During a fork `copy_page_tables()` treats RESERVED page tables differently. It sets pointers in the process page directories to point to kernel page tables and does not actually allocate new page tables as it does normally. As an example the `kernel_stack_page` (which sits somewhere in the kernel space) does not need an associated `page_table` allocated in the process' `pg_dir` to map it.

The interrupt instruction sets the stack pointer and stack segment from the privilege 0 values saved in the tss of the current task. Note that the kernel stack is a really fragmented object--it's not a single object, but rather a bunch of stack frames each allocated when a process is created, and released when it exits. The kernel stack should never grow so rapidly within a process context that it extends below the current frame.

Acquiring and Freeing Memory: Paging Policy

[Note: swapping has also been massively changed in recent kernels, with the ``kswap'' changes.]

When any kernel routine wants memory it ends up calling `get_free_page()`. This is at a lower level than `kmalloc()` (in fact `kmalloc()` uses `get_free_page()` when it needs more memory).

`get_free_page()` takes one parameter, a priority. Possible values are `GFP_BUFFER`, `GFP_KERNEL`, `GFP_NFS`, and `GFP_ATOMIC`. It takes a page off of the `free_page_list`, updates `mem_map`, zeroes the page and returns the physical address of the page (note that `kmalloc()` returns a physical address. The logic of the mm depends on the identity map between logical and physical addresses).

That itself is simple enough. The problem, of course, is that the `free_page_list` may be empty. If you did not request an atomic operation, at this stage, you enter into the realm of page stealing which we'll go into in a moment. As a last resort (and for atomic requests) a page is torn off from the `secondary_page_list` (as you may have guessed, when pages are freed, the `secondary_page_list` gets filled up first).

The actual manipulation of the `page_lists` and `mem_map` occurs in this mysterious macro called `REMOVE_FROM_MEM_QUEUE()` which you probably never want to look into. Suffice it to say that interrupts are disabled. **[I think that this should be explained here. It is not *that* hard...]**

Now back to the page stealing bit. `get_free_page()` calls `try_to_free_page()` which repeatedly calls `shrink_buffers()` and `swap_out()` in that order until it is successful in freeing a page. The priority is increased on each successive iteration so that these two routines run through their page stealing loops more often.

Here's one run through `swap_out()`:

- Run through the process table and get a swappable task, say, *Q*.
- Find a user page table (not RESERVED) in *Q*'s space.
- For each *page* in the table `try_to_swap_out(`*`page`*`)`.
- Quit when a page is freed.

Note that `swap_out()` (called by `try_to_free_page()`) maintains static variables so it may resume the search where it left off on the previous call.

`try_to_swap_out()` scans the page tables of all user processes and enforces the stealing policy:

1. Do not fiddle with RESERVED pages.
2. Age the page if it is marked accessed (1 bit).
3. Don't tamper with recently acquired pages (`last_free_pages[]`).
4. Leave dirty pages with `map_counts` > 1 alone.
5. Decrement the `map_count` of clean pages.
6. Free clean pages if they are unmapped.
7. Swap dirty pages with a `map_count` of 1.

Of these actions, 6 and 7 will stop the process as they result in the actual freeing of a physical page. Action 5 results in one of the processes losing an unshared clean page that was not accessed recently (decrement `Q->rss`) which is not all that bad, but the cumulative effects of a few iterations can slow down a process considerably. At present, there are 6 iterations, so a page shared by 6 processes can get stolen if it is clean.

Page table entries are updated and the TLB invalidated.

The actual work of freeing the page is done by `free_page()`, the complement of `get_free_page()`. It ignores RESERVED pages, updates `mem_map`, then frees the page and updates the `page_lists` if it is unmapped. For swapping (in 6 above), `write_swap_page()` gets called and does nothing remarkable from the memory management perspective.

The details of `shrink_buffers()` would take us too far afield. Essentially it looks for free buffers, then writes out dirty buffers, then goes at busy buffers and calls `free_page()` when its able to free all the buffers on a page.

Note that page directories and page tables along with RESERVED pages do not get swapped, stolen or aged. They are mapped in the process page directory through reserved page tables. They are freed only on exit from the process.

The page fault handlers

When a process is created via fork, it starts out with a page directory and a page or so of the executable. So the page fault handler is the source of most of a processes' memory.

The page fault handler `do_page_fault()` retrieves the faulting address from the register cr2. The error code (retrieved in sys_call.S) differentiates user/supervisor access and the reason for the fault--write protection or a missing page. The former is handled by `do_wp_page()` and the latter by `do_no_page()`.

If the faulting address is greater than TASK_SIZE the process receives a SIGKILL. **[Why this check? This can only happen in kernel mode because of segment level protection.]**

These routines have some subtleties as they can get called from an interrupt. You can't assume that it is the ``current'' task that is executing.

`do_no_page()` handles three possible situations:

1. The page is swapped.
2. The page belongs to the executable or a shared library.
3. The page is missing--a data page that has not been allocated.

In all cases `get_empty_pgtable()` is called first to ensure the existence of a page table that covers the faulting address. In case 3 `get_empty_page()` is called to provide a page at the required address and in case of the swapped page, `swap_in()` is called.

In case 2, the handler calls `share_page()` to see if the page is shareable with some other process. If that fails it reads in the page from the executable or library (It repeats the call to `share_page()` in case another process did the same meanwhile). Any portion of the page beyond the brk value is zeroed.

A page read in from the disk is counted as a major fault (`maj_flt`). This happens with a `swap_in()` or when it is read from the executable or a library. Other cases are deemed minor faults (`min_flt`).

When a shareable page is found, it is write-protected. A process that writes to a shared page will then have to go through `do_wp_page()` which does the copy-on-write.

`do_wp_page()` does the following:

- Send SIGSEGV if any user process is writing to current `code_space`.
- If the old page is not shared then just unprotect it.
  Else `get_free_page()` and `copy_page()`. The page acquires the dirty flag from the old page. Decrement the map count of the old page.

Paging

Paging is swapping on a page basis rather than by entire processes. We will use swapping here to refer to paging, since Linux only pages, and does not swap, and people are more used to the word ``swap'' than ``page." Kernel pages are never swapped. Clean pages are also not written to swap. They are freed and reloaded when required. The swapper maintains a single bit of aging info in the `PAGE_ACCESSED` bit of the page table entries. **[What are the maintainance details? How is it used?]**

Linux supports multiple swap files or devices which may be turned on or off by the swapon and swapoff system calls. Each swapfile or device is described by a `struct swap_info_struct` (swap.c).

```
static struct swap_info_struct {
      unsigned long flags;
      struct inode * swap_file;
      unsigned int swap_device;
      unsigned char * swap_map;
      char * swap_lockmap;
      int lowest_bit;
      int highest_bit;
} swap_info[MAX_SWAPFILES];
```

The flags field (`SWP_USED` or `SWP_WRITEOK`) is used to control access to the swap files. When `SWP_WRITEOK` is off space will not be allocated in that file. This is used by swapoff when it tries to unuse a file. When swapon adds a new swap file it sets `SWP_USED`. A static variable `nr_swapfiles` stores the number of currently active swap files. The fields `lowest_bit` and `highest_bit` bound the free region in the swap file and are used to speed up the search for free swap space.

The user program mkswap initializes a swap device or file. The first page contains a signature (`SWAP-SPACE') in the last 10 bytes, and holds a bitmap. Initially 0's in the bitmap signal bad pages. A `1' in the bitmap means the corresponding page is free. This page is never allocated so the initialization needs to be done just once.

The syscall `swapon()` is called by the user program swapon typically from /etc/rc. A couple of pages of memory are allocated for `swap_map` and `swap_lockmap`.

`swap_map` holds a byte for each page in the swapfile. It is initialized from the bitmap to contain a 0 for available pages and 128 for unusable pages. It is used to maintain a count of swap requests on each page in the swap file. `swap_lockmap` holds a bit for each page that is used to ensure mutual exclusion when reading or writing swap files.

When a page of memory is to be swapped out an index to the swap location is obtained by a call to `get_swap_page()`. This index is then stored in bits 1-31 of the page table entry so the swapped page may be located by the page fault handler, `do_no_page()` when needed.

The upper 7 bits of the index give the swapfile (or device) and the lower 24 bits give the page number on that device. That makes as many as 128 swapfiles, each with room for about 64 GB, but the space overhead due to the `swap_map` would be large. Instead the swapfile size is limited to 16 MB, because the `swap_map` then takes 1 page.

The function `swap_duplicate()` is used by `copy_page_tables()` to let a child process inherit swapped pages during a fork. It just increments the count maintained in `swap_map` for that page. Each process will swap in a separate copy of the page when it accesses it.

`swap_free()` decrements the count maintained in `swap_map`. When the count drops to 0 the page can be reallocated by `get_swap_page()`. It is called each time a swapped page is read into memory (`swap_in()`) or when a page is to be discarded (`free_one_table()`, etc.).

Messages

80386 Memory Management

A logical address specified in an instruction is first translated to a linear address by the segmenting hardware. This linear address is then translated to a physical address by the paging unit.

Paging on the 386

There are two levels of indirection in address translation by the paging unit. A **page directory** contains pointers to 1024 page tables. Each **page table** contains pointers to 1024 pages. The register CR3 contains the physical base address of the page directory and is stored as part of the TSS in the `task_struct` and is therefore loaded on each task switch.

A 32-bit Linear address is divided as follows:

31 22	**21 ...... 12**	**11 ...... 0**
DIR	TABLE	OFFSET

Physical address is then computed (in hardware) as:

CR3 + DIR	points to the table_base.
table_base + TABLE	points to the page_base.
physical_address =	page_base + OFFSET

Page directories (page tables) are page aligned so the lower 12 bits are used to store useful information about the page table (page) pointed to by the entry.

Format for Page directory and Page table entries:

31 12	**11 .. 9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADDRESS	OS	0	0	D	A	0	0	U/S	R/W	P

D	1 means page is dirty (undefined for page directory entry).
R/W	0 means readonly for user.
U/S	1 means user page.
P	1 means page is present in memory.
A	1 means page has been accessed (set to 0 by aging).
OS	bits can be used for LRU etc, and are defined by the OS.

The corresponding definitions for Linux are in .

When a page is swapped, bits 1-31 of the page table entry are used to mark where a page is stored in swap (bit 0 must be 0).

Paging is enabled by setting the highest bit in CR0. **[in head.S?]** At each stage of the address translation access permissions are verified and pages not present in memory and protection violations result in page faults. The fault handler (in memory.c) then either brings in a new page or unwriteprotects a page or does whatever needs to be done.

Page Fault handling Information

- The register CR2 contains the linear address that caused the last page fault.
- Page Fault Error Code (16 bits):

bit	cleared	set
0	page not present	page level protection
1	fault due to read	fault due to write
2	supervisor mode	user mode

  The rest are undefined. These are extracted in sys_call.S.

The Translation Lookaside Buffer (TLB) is a hardware cache for physical addresses of the most recently used virtual addresses. When a virtual address is translated the 386 first looks in the TLB to see if the information it needs is available. If not, it has to make a couple of memory references to get at the page directory and then the page table before it can actually get at the page. Three physical memory references for address translation for every logical memory reference would kill the system, hence the TLB.

The TLB is flushed if CR3 loaded or by task switch that changes CR0. It is explicitly flushed in Linux by calling `invalidate()` which just reloads CR3.

Segments in the 80386

Segment registers are used in address translation to generate a linear address from a logical (virtual) address.
```
linear_address = segment_base + logical_address
```
The linear address is then translated into a physical address by the paging hardware.

Each segment in the system is described by a 8 byte segment descriptor which contains all pertinent information (base, limit, type, privilege).

The segments are:

Regular segments

- ❍ code and data segments

System segments

- ❍ (TSS) task state segments
- ❍ (LDT) local descriptor tables

Characteristics of system segments

- System segments are task specific.
- There is a Task State Segment (TSS) associated with each task in the system. It contains the `tss_struct` (sched.h). The size of the segment is that of the `tss_struct` excluding the `i387_union` (232 bytes). It contains all the information necessary to restart the task.
- The LDT's contain regular segment descriptors that are private to a task. In Linux there is one LDT per task. There is room for 32 descriptors in the linux `task_struct`. The normal LDT generated by Linux has a size of 24 bytes, hence room for only 3 entries as above. Its contents are:
  LDT[0]
  Null (mandatory)
  LDT[1]
  user code segment descriptor.
  LDT[2]
  user data/stack segment descriptor.
- The user segments all have base=0x00 so that the linear address is the same as the logical address.

To keep track of all these segments, the 386 uses a global descriptor table (GDT) that is setup in memory by the system (located by the GDT register). The GDT contains a segment descriptors for each task state segment, each local descriptor tablet and also regular segments. The Linux GDT contains just two normal segment entries:

- GDT[0] is the null descriptor.
- GDT[1] is the kernel code segment descriptor.
- GDT[2] is the kernel data/stack segment descriptor.

The rest of the GDT is filled with TSS and LDT system descriptors:

- GDT[3] ???
- GDT[4] = TSS0, GDT[5] = LDT0,
- GDT[6] = TSS1, GDT[7] = LDT1
- ... etc. ...

LDT[*n*] != LDT*n*

LDT[*n*] = the *n*th descriptor in the LDT of the current task.

LDT*n* = a descriptor in the GDT for the LDT of the *n*th task.

The kernel segments have base 0xc0000000 which is where the kernel lives in the linear view. Before a segment can be used, the contents of the descriptor for that segment must be loaded into the segment register. The 386 has a complex set of criteria regarding access to segments so you can't simply load a descriptor into a segment register. Also these segment registers have programmer invisible portions. The visible portion is what is usually called a segment register: cs, ds, es, fs, gs, and ss.

The programmer loads one of these registers with a 16-bit value called a selector. The selector uniquely identifies a segment descriptor in one of the tables. Access is validated and the corresponding descriptor loaded by the hardware.

Currently Linux largely ignores the (overly?) complex segment level protection afforded by the 386. It is biased towards the paging hardware and the associated page level protection. The segment level rules that apply to user processes are

1. A process cannot directly access the kernel data or code segments
2. There is always limit checking but given that every user segment goes from 0x00 to 0xc0000000 it is unlikely to apply. **[This has changed, and needs updating, please.]**

Selectors in the 80386

A segment selector is loaded into a segment register (cs, ds, etc.) to select one of the regular segments in the system as the one addressed via that segment register.

Segment selector Format:

15 3	**2 1**	**0**
index	TI	RPL

TI Table indicator:
 0 means selector indexes into GDT
 1 means selector indexes into LDT
RPL Privelege level. Linux uses only two privelege levels.
 0 means kernel
 3 means user

Examples:

Kernel code segment
 TI=0, index=1, RPL=0, therefore selector = 0x08 (GDT[1])
User data segment
 TI=1, index=2, RPL=3, therefore selector = 0x17 (LDT[2])

Selectors used in Linux:

TI	**index**	**RPL**	**selector**	**segment**	
0	1	0	0x08	kernel code	GDT[1]
0	2	0	0x10	kernel data/stack	GDT[2]
0	3	0	???	???	GDT[3]
1	1	3	0x0F	user code	LDT[1]
1	2	3	0x17	user data/stack	LDT[2]

Selectors for system segments are not to be loaded directly into segment registers. Instead one must load the TR or LDTR.

On entry into syscall:

- ds and es are set to the kernel data segment (0x10)
- fs is set to the user data segment (0x17) and is used to access data pointed to by arguments to the system call.
- The stack segment and pointer are automatically set to ss0 and esp0 by the interrupt and the old values restored when the syscall returns.

Segment descriptors

There is a segment descriptor used to describe each segment in the system. There are regular descriptors and system descriptors. Here's a descriptor in all its glory. The strange format is essentially to maintain compatibility with the 286. Note that it takes 8 bytes.

63-54	**55**	**54**	**53**	**52**	**51-48**	**47**	**46**	**45**	**44-40**	**39-16**	**15-0**
Base 31-24	G	D	R	U	Limit 19-16	P	DPL	S	TYPE	Segment Base 23-0	Segment Limit 15-0

Explanation:

R	reserved (0)
DPL	0 means kernel, 3 means user
G	1 means 4K granularity (Always set in Linux)
D	1 means default operand size 32bits
U	programmer definable
P	1 means present in physical memory
S	0 means system segment, 1 means normal code or data segment.
Type	There are many possibilities. Interpreted differently for system and normal descriptors.

Linux system descriptors:
TSS: P=1, DPL=0, S=0, type=9, limit = 231 room for 1 `tss_struct`.
LDT: P=1, DPL=0, S=0, type=2, limit = 23 room for 3 segment descriptors.
The base is set during `fork()`. There is a TSS and LDT for each task.

Linux regular kernel descriptors: (head.S)
code: P=1, DPL=0, S=1, G=1, D=1, type=a, base=0xc0000000, limit=0x3ffff
data: P=1, DPL=0, S=1, G=1, D=1, type=2, base=0xc0000000, limit=0x3ffff

The LDT for task[0] contains: (sched.h)
code: P=1, DPL=3, S=1, G=1, D=1, type=a, base=0xc0000000, limit=0x9f
data: P=1, DPL=3, S=1, G=1, D=1, type=2, base=0xc0000000, limit=0x9f

The default LDT for the remaining tasks: (`exec()`)
code: P=1, DPL=3, S=1, G=1, D=1, type=a, base=0, limit= 0xbffff
data: P=1, DPL=3, S=1, G=1, D=1, type=2, base=0, limit= 0xbffff

The size of the kernel segments is 0x40000 pages (4KB pages since G=1 = 1 Gigabyte). The type implies that the permissions on the code segment is read-exec and on the data segment is read-write.

Registers associated with segmentation.

Format of segment register: (Only the selector is programmer visible)

16-bit	**32-bit**	**32-bit**	
selector	physical base addr	segment limit	attributes

The invisible portion of the segment register is more conveniently viewed in terms of the format used in the descriptor table entries that the programmer sets up. The descriptor tables have registers associated with them that are used to locate them in memory. The GDTR (and IDTR) are initialized at startup once the tables are defined. The LDTR is loaded on each task switch.

Format of GDTR (and IDTR):

32-bits	**16-bits**
Linear base addr	table limit

The TR and LDTR are loaded from the GDT and so have the format of the other segment registers. The task register (TR) contains the descriptor for the currently executing task's TSS. The execution of a jump to a TSS selector causes the state to be saved in the old TSS, the TR is loaded with the new descriptor and the registers are restored from the new TSS. This is the process used by schedule to switch to various user tasks. Note that the field `tss_struct.ldt` contains a selector for the LDT of that task. It is used to load the LDTR. (sched.h)

Macros used in setting up descriptors

Some assembler macros are defined in sched.h and system.h to ease access and setting of descriptors. Each TSS entry and LDT entry takes 8 bytes.

Manipulating GDT system descriptors:

`_TSS(`*n*`)`, `_LDT(`*n*`)`
  These provide the index into the GDT for the *n*'th task.
`_LDT(n)` is stored in the the ldt field of the `tss_struct` by fork.
`_set_tssldt_desc(n, addr, limit, type)`
  `ulong *n` points to the GDT entry to set (see fork.c).
  The segment base (TSS or LDT) is set to 0xc0000000 + `addr`.
  Specific instances of the above are, where ltype refers to the byte containing P, DPL, S and type:
  `set_ldt_desc(n, addr)` ltype = 0x82
    P=1, DPL=0, S=0, type=2 means LDT entry.
    limit = 23 => room for 3 segment descriptors.
  `set_tss_desc(n, addr)` ltype = 0x89
    P=1, DPL=0, S=0, type = 9, means available 80386 TSS limit = 231 room for 1 tss_struct.
`load_TR(n)`,
  `load_ldt(n)` load descriptors for task number n into the task register and ldt register.
`ulong get_base (struct desc_struct ldt)`
  gets the base from a descriptor.
`ulong get_limit (ulong segment)`
  gets the limit (size) from a segment selector.
  Returns the size of the segment in bytes.
`set_base(struct desc_struct ldt, ulong base)`,
`set_limit(struct desc_struct ldt, ulong limit)`
  Will set the base and limit for descriptors (4K granular segments).
  The limit here is actually the size in bytes of the segment.
`_set_seg_desc(gate_addr, type, dpl, base, limit)`
  Default values 0x00408000 => D=1, P=1, G=0
  Present, operation size is 32 bit and max size is 1M.
  `gate_addr` must be a `(ulong *)`

Messages

1. paging initialization, doc update *by droux@cs.unm.edu*
2. User Code and Data Segment no longer in LDT. *by Lennart Benschop*

paging initialization, doc update

Forum: 80386 Memory Management
Keywords: paging, initialization, x86, CR0
Date: Mon, 03 Jun 1996 20:46:15 GMT
From: <droux@cs.unm.edu>

```
From the x86 memory management doc:

> Paging is enabled by setting the highest bit in CR0.
> [in head.S?]

This is correct, the editor note can be suppressed.
CR0 initialisation for paging is performed by
"setup_paging" which is implemented in (1.2.13)

  arch/i386/kernel/head.S
```

User Code and Data Segment no longer in LDT.

Forum: 80386 Memory Management
Date: Tue, 23 Jul 1996 09:39:45 GMT
From: Lennart Benschop <benschop@eb.ele.tue.nl>

The user code and data segments of a process are no longer in the LDT, but in the GDT instead. The code and data segment of each process starts at linear address 0 anyway, only the physical address is different (different page directory =CR3)

Processes still have an LDT, this can be used by certain applications (WINE).

In very early versions of Linux, user space was restricted to 64 MB and there were a maximum of 64 processes (including process 0, which had the kernel in its user space). Back then each process had a different linear address. making a total of 4GB. There was only one page directory, and there were per-process code and data segments, included in the LDT. This (somewhat elegant) scheme was abandoned to allow more than 64 processes and a per process virtual address space of more than 64MB. That's why certain kernels had they suer code and data segments in the LDT, though they were in fact the same segments for all processes.

How System Calls Work on Linux/i86

This section covers first the mechanisms provided by the 386 for handling system calls, and then shows how Linux uses those mechanisms. This is not a reference to the individual system calls: There are very many of them, new ones are added occasionally, and they are documented in man pages that should be on your Linux system.

What Does the 386 Provide?

The 386 recognizes two event classes: exceptions and interrupts. Both cause a forced context switch to new a procedure or task. Interrupts can occur at unexpected times during the execution of a program and are used to respond to signals from hardware. Exceptions are caused by the execution of instructions.

Two sources of interrupts are recognized by the 386: Maskable interrupts and Nonmaskable interrupts. Two sources of exceptions are recognized by the 386: Processor detected exceptions and programmed exceptions.

Each interrupt or exception has a number, which is referred to by the 386 literature as the vector. The NMI interrupt and the processor detected exceptions have been assigned vectors in the range 0 through 31, inclusive. The vectors for maskable interrupts are determined by the hardware. External interrupt controllers put the vector on the bus during the interrupt-acknowledge cycle. Any vector in the range 32 through 255, inclusive, can be used for maskable interrupts or programmed exceptions. Here is a listing of all the possible interrupts and exceptions:

0	divide error
1	debug exception
2	NMI interrupt
3	Breakpoint
4	INTO-detected Overflow
5	BOUND range exceeded
6	Invalid opcode
7	coprocessor not available
8	double fault
9	coprocessor segment overrun
10	invalid task state segment
11	segment not present

12	stack fault
13	general protection
14	page fault
15	reserved
16	coprocessor error
17-31	reserved
32-255	maskable interrupts

The priority of simultaneous interrupts and exceptions is:

HIGHEST	Faults except debug faults
.	Trap instructions INTO, INT n, INT 3
.	Debug traps for this instruction
.	Debug traps for next instruction
.	NMI interrupt
LOWEST	INTR interrupt

How Linux Uses Interrupts and Exceptions

Under Linux the execution of a system call is invoked by a maskable interrupt or **exception** class transfer, caused by the instruction `int 0x80`. We use vector 0x80 to transfer control to the kernel. This interrupt vector is initialized during system startup, along with other important vectors like the system clock vector.

iBCS2 requries an `lcall 0,7` instruction, which Linux can send to the iBCS2 compatibility module appropriate if an iBCS2-compliant binary is being executed. In fact, Linux will assume that an iBCS2-compliant binary is being executed if an `lcall 0,7` call is executed, and will automatically switch modes.

As of version 0.99.2 of Linux, there are 116 system calls. Documentation for these can be found in the man (2) pages. When a user invokes a system call, execution flow is as follows:

- Each call is vectored through a stub in libc. Each call within the libc library is generally a `syscallX()` macro, where *X* is the number of parameters used by the actual routine. Some system calls are more complex then others because of variable length argument lists, but even these complex system calls must use the same entry point: they just have more parameter setup overhead. Examples of a complex system call include `open()` and `ioctl()`.
- Each syscall macro expands to an assembly routine which sets up the calling stack frame and calls `_system_call()` through an interrupt, via the instruction `int $0x80`

  For example, the setuid system call is coded as

```
_syscall1(int,setuid,uid_t,uid);
```

which will expand to:

```
_setuid:
  subl $4,%exp
  pushl %ebx
  movzwl 12(%esp),%eax
  movl %eax,4(%esp)
  movl $23,%eax
  movl 4(%esp),%ebx
  int $0x80
  movl %eax,%edx
  testl %edx,%edx
  jge L2
  negl %edx
  movl %edx,_errno
  movl $-1,%eax
  popl %ebx
  addl $4,%esp
  ret
L2:
  movl %edx,%eax
  popl %ebx
  addl $4,%esp
  ret
```

The macro definition for the `syscallX()` macros can be found in /usr/include/linux/unistd.h, and the user-space system call library code can be found in /usr/src/libc/syscall/

- At this point no system code for the call has been executed. Not until the `int $0x80` is executed does the call transfer to the kernel entry point `_system_call()`. This entry point is the same for all system calls. It is responsible for saving all registers, checking to make sure a valid system call was invoked and then ultimately transfering control to the actual system call code via the offsets in the `_sys_call_table`. It is also responsible for calling `_ret_from_sys_call()` when the system call has been completed, but before returning to user space.

  Actual code for `system_call` entry point can be found in /usr/src/linux/kernel/sys_call.S Actual code for many of the system calls can be found in /usr/src/linux/kernel/sys.c, and the rest are found elsewhere. `find` is your friend.
- After the system call has executed, `_ret_from_sys_call()` is called. It checks to see if the scheduler should be run, and if so, calls it.
- Upon return from the system call, the `syscallX()` macro code checks for a negative return value, and if there is one, puts a positive copy of the return value in the global variable `_errno`, so that it can be accessed by code like `perror()`.

How Linux Initializes the system call vectors

The `startup_32()` code found in /usr/src/linux/boot/head.S starts everything off by calling `setup_idt()`. This routine sets up an IDT (Interrupt Descriptor Table) with 256 entries. No interrupt entry points are actually loaded by this routine, as that is done only after paging has been enabled and the kernel has been moved to 0xC0000000. An IDT has 256 entries, each 4 bytes long, for a total of 1024 bytes. When `start_kernel()` (found in /usr/src/linux/init/main.c) is called it invokes `trap_init()` (found in /usr/src/linux/kernel/traps.c). `trap_init()` sets up the IDT via the macro `set_trap_gate()` (found in /usr/include/asm/system.h). `trap_init()` initializes the interrupt descriptor table as shown here:

0	divide_error
1	debug
2	nmi
3	int3
4	overflow
5	bounds
6	invalid_op
7	device_not_available
8	double_fault
9	coprocessor_segment_overrun
10	invalid_TSS
11	segment_not_present
12	stack_segment
13	general_protection
14	page_fault
15	reserved
16	coprocessor_error
17	alignment_check
18-48	reserved

At this point the interrupt vector for the system calls is not set up. It is initialized by `sched_init()` (found in /usr/src/linux/kernel/sched.c). A call to `set_system_gate (0x80, &system_call)` sets interrupt 0x80 to be a vector to the `system_call()` entry point.

How to Add Your Own System Calls

1. Create a directory under the /usr/src/linux/ directory to hold your code.
2. Put any include files in /usr/include/sys/ and /usr/include/linux/.

3. Add the relocatable module produced by the link of your new kernel code to the `ARCHIVES` and the subdirectory to the `SUBDIRS` lines of the top level Makefile. See fs/Makefile, target fs.o for an example.
4. Add a `#define __NR_`*`xx`* to unistd.h to assign a call number for your system call, where *`xx`*, the index, is something descriptive relating to your system call. It will be used to set up the vector through `sys_call_table` to invoke you code.
5. Add an entry point for your system call to the `sys_call_table` in sys.h. It should match the index (*`xx`*) that you assigned in the previous step. The `NR_syscalls` variable will be recalculated automatically.
6. Modify any kernel code in kernel/fs/mm/, etc. to take into account the environment needed to support your new code.
7. Run make from the top level to produce the new kernel incorporating your new code.

At this point, you will have to either add a syscall to your libraries, or use the proper `_syscall`*`n`*`()` macro in your user program for your programs to access the new system call. The *386DX Microprocessor Programmer's Reference Manual* is a helpful reference, as is James Turley's *Advanced 80386 Programming Techniques*. See the Annotated Bibliography.

Messages

4. wrong file for system_call code *by Tim Bird*
3. would be nice to explain syscall macros *by Tim Bird*
2. wrong file for syscallX() macro *by Tim Bird*
1. the directory /usr/src/libc/syscall/ *by vijay gupta*
    1. ...no longer exists. *by Michael K. Johnson*
    -> the solution to the problem *by Vijay Gupta*

wrong file for system_call code

Forum: How System Calls Work on Linux/i86
Keywords: syscall assembly error
Date: Fri, 13 Sep 1996 01:44:29 GMT
From: Tim Bird <tbird@caldera.com>

This page contains the sentence "Actual code for system_call entry point can be found in /usr/src/linux/kernel/sys_call.S"

This should read: Actual code for the system_call entry point (for the intel architecture) can be found in /usr/src/linux/arch/i386/kernel/entry.S

would be nice to explain syscall macros

Forum: How System Calls Work on Linux/i86

Keywords: sycall
Date: Fri, 13 Sep 1996 01:37:11 GMT
From: Tim Bird <tbird@caldera.com>

The syscall macros are a little dense to decipher. It took me a while to determine how the macro syscall1(int,setuid,uid_t,uid) expanded into the assembly code shown.

It might be nice to show the macro, and explain a little about how it gets expanded.

Here is the source for the _syscall1 macro

```
        #define _syscall1(type,name,type1,arg1) \
        type name(type1 arg1) \
        { \
        long __res; \
        __asm__ volatile ("int $0x80" \
                : "=a" (__res) \
                : "0" (__NR_##name),"b" ((long)(arg1))); \
        if (__res >= 0) \
                return (type) __res; \
        errno = -__res; \
        return -1; \
        }

When expanded, this become the code
        int setuid(uid_t uid)
        {
                long __res;
                __asm__ volatile ("int $0x80" \
                        : "=a" (__res) \
                        : "0" (__NR_setuid), "b" ((long)(uid)));
                if (__res >= 0 )
                        return (int) __res;
                errno = -__res;
                return -1;
        }
```

```
It's pretty easy to see how the cleanup code converts
into assembly, but the setup code eluded me until
I figured out the following:
        "=a" (__res) means the result comes back in %eax
        "0" (__NR_setuid) means put the system call number
                into %eax on entry
        "b" ((long)(uid) means put the first argument
                into %ebx on entry
```

syscallX macros that use additional parameters use %ecx, %edx, %esi, and %edi to hold additional values passed through the call.

wrong file for syscallX() macro

Forum: How System Calls Work on Linux/i86
Date: Fri, 13 Sep 1996 01:25:44 GMT
From: Tim Bird <tbird@caldera.com>

```
This page contains the sentence:
" The macro definition for the syscallX() macros can be found
 in /usr/include/linux/unistd.h"
```

Actually, this file containts the system call numbers. The macros for system call generation are located in the file /usr/include/asm/unistd.h

the directory /usr/src/libc/syscall/

Forum: How System Calls Work on Linux/i86
Keywords: system call
Date: Sat, 18 May 1996 03:30:26 GMT
From: vijay gupta <vijay@crhc.uiuc.edu>

Hi,

```
        The directory /usr/src/libc/syscall/ is essential
for hacking the libc code in order to add a new
system call.

        Unfortunately, I have been unable to find this
directory from
libc-5.3.9.tar.gz from ftp://sunsite.unc.edu/pub/Linux/GCC/
or from libc-4.X.
```

Does anyone have any ideas on this ?

Thank you very much,

```
        Vijay Gupta
        (Email : vijay@crhc.uiuc.edu)
```

Messages

1. ...no longer exists. *by Michael K. Johnson*
-> the solution to the problem *by Vijay Gupta*

...no longer exists.

Forum: How System Calls Work on Linux/i86
Re: the directory /usr/src/libc/syscall/ (vijay gupta)
Keywords: system call libc
Date: Sat, 18 May 1996 12:52:48 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

The /usr/src/libc/syscall/ directory no longer exists.

Instead, they files you need to modify are in the libc/sysdeps/linux/ directory. If your system call only works on one architecture, then you need to use the architecture-dependent subdirectories i386 and m68k (at present; that will soon expand to at least sparc, and maybe other platforms).

If you need to know more, please ask more...

Messages

1. the solution to the problem *by Vijay Gupta*

the solution to the problem

Forum: How System Calls Work on Linux/i86
Re: the directory /usr/src/libc/syscall/ (vijay gupta)
Re: ...no longer exists. (Michael K. Johnson)
Keywords: system call libc
Date: Tue, 21 May 1996 23:05:39 GMT
From: Vijay Gupta <vijay@crhc.uiuc.edu>

Hi everybody,

```
       Thanks to the two people who replied to me on
this. The solution to the problem is as follows :
```

the khg seems to be wrong in assuming there was a directory syscall in the C library. Instead, there is a directory sysdeps/linux, which contains, among others, socketpair.c, which defines the function

```
int
socketpair(int family, int type, int protocol, int sockvec[2])
{
    unsigned long args[4];

    args[0] = family;
    args[1] = type;
    args[2] = protocol;
    args[3] = (unsigned long)sockvec;
    return socketcall(SYS_SOCKETPAIR, args);
}
```

If you look at /usr/src/linux/net/socket.c, you will find a good match with that code. The socketcall function then is not defined by a C macro, but by an assembler macro in __socketcall.S:

```
SYSCALL__ (socketcall, 2)
    ret
Please note that the socket system calls are special because of that level
of indirection. The wait(2) function is declared as

#ifdef __SVR4_I386_ABI_L1__
#define wait4   __wait4
#else
static inline
_syscall4(__pid_t,wait4,__pid_t,pid,__WAIT_STATUS_DEFN,status,int,options,struc
t
 rusage *,ru)
#endif

__pid_t
```

```
__wait(__WAIT_STATUS_DEFN wait_stat)
{
    return wait4(WAIT_ANY, wait_stat, 0, NULL);
}
```

(so it is actually wait(3) in Linux, with wait4(2) being the system call).

```
Thanks again,
       Vijay
```

Other Sources of Information

Other sources specifically about writing device drivers.

Randy Bentson recently wrote an interesting book called *Inside Linux*. It has some information on basic operating system theory, some that is specifically related to Linux, and occasional parts that aren't really related to Linux at all (such as a discussion of the Georgia Tech shell). ISBN 0-916151-89-1, published by Specialized System Consultants

Inline Assembly with DJGPP really applies to any version of GCC on a 386, and some of it is generic GCC inline assembly. Definitely required reading for anyone who wants to do inline assembly with Linux and GCC.

The Annotated Bibliography mentions plenty of books out that don't have ``Linux" in the title which may be useful to Linux programmers. Especially if you are new to kernel programming, you may do well to pick up one of the textbooks recommended in the bibliography.

Messages

8. Linux Pgrogrammers Guide (LPG) *by Federico Lucifredi*
7. TTY documentation *by Michael De La Rue*
    1. In the queue... *by Michael K. Johnson*
        1. TTY documentation *by Eugene Kanter*
    2. Untitled *by Yusuf Motiwala*
5. The vger linux mail list archives *by Drew Puch*
4. German book on Linux Kernel Programming *by Jochen Hein*
    1. English version of Linux Kernel Internals *by Naoshad Eduljee*
    -> Book Review? *by Josh Du"Bois*
    -> Thumbed through it *by Brian J. Murrell*
3. Multi-architecture support *by Michael K. Johnson*
    1. Linux Architecture-Specific Kernel Interfaces *by Drew Puch*
2. Analysis of the Ext2fs structure *by Michael K. Johnson*
1. To add more sources of information: *by Michael K. Johnson*

Linux Pgrogrammers Guide (LPG)

Forum: Other Sources of Information
Keywords: Linux Programming
Date: Fri, 18 Apr 1997 04:58:44 GMT
From: Federico Lucifredi <lucifred@cs.bc.edu>

Does anybody know what happened to the

LPG

? it doesn't seem to have been updated beyond v 0.4 (3.95)

TTY documentation

Forum: Other Sources of Information
Keywords: TTY TeX Documentation Kernel Device Driver
Date: Mon, 08 Jul 1996 07:40:29 GMT
From: Michael De La Rue <miked@ed.ac.uk>

I have a copy of some TTY documentation (describing the TTY driver code, what it is, what it does etc.) Written by Jan Charvat (jcharvat@cs.ucr.edu) and Barnett Hsu (barnett@cs.ucr.edu). It's in TeX, with a Fig Figure. I haven't had time to read it over in detail, but it might be worth asking for, at least as a basis of more info for the KHG. There's no copyright on the document so I won't put it up for people to read, but someone might like to get in contact with the authors and I presume that I can pass it on to people.

Messages

1. In the queue... *by Michael K. Johnson*
    1. TTY documentation *by Eugene Kanter*
2. Untitled *by Yusuf Motiwala*

In the queue...

Forum: Other Sources of Information
Re: TTY documentation (Michael De La Rue)
Keywords: TTY TeX Documentation Kernel Device Driver
Date: Wed, 31 Jul 1996 15:47:36 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

The authors have agreed to have it included in the KHG.

I have a copy of the article, and when I have time to set it in HTML, it will be added.

Thanks much!

Messages

1. TTY documentation *by Eugene Kanter*

TTY documentation

Forum: Other Sources of Information
Re: TTY documentation (Michael De La Rue)
Re: In the queue... (Michael K. Johnson)
Keywords: TTY TeX Documentation Kernel Device Driver
Date: Wed, 23 Oct 1996 16:34:05 GMT
From: Eugene Kanter <eugene.kanter@ab.com>

May I have at least plain text version of TTY document?

Thanks.

Untitled

Forum: Other Sources of Information
Re: TTY documentation (Michael De La Rue)
Keywords: TTY TeX Documentation Kernel Device Driver
Date: Sat, 29 Mar 1997 07:24:16 GMT
From: Yusuf Motiwala <yusuf@scientist.com>

Can you please mail me tty documentation.

Regards,

Yusuf

ymotiwala@hss.hns.com

The vger linux mail list archives

Forum: Other Sources of Information
Keywords: mailing list
Date: Mon, 27 May 1996 17:51:08 GMT
From: Drew Puch <aapuch@eos.ncsu.edu>

Good place to see if the question you are about to ask are already answered vger mail list for linux topics

German book on Linux Kernel Programming

Forum: Other Sources of Information
Keywords: German book in Linux Kernel Hacking
Date: Mon, 27 May 1996 13:19:59 GMT
From: Jochen Hein <jochen.hein@informatik.tu-clausthal.de>

```
There is a german book on "Linux-Kernel-Programmierung"
Algorithmen und Strukturen der Version 1.2
Michael Beck, Harald Böhme, Mirko Dziadzka, Ulrich Kunitz,
Robert Magnus, Dirk Verworner
Addison-Wesley, Germany, ISBN 3-89319-939-x, DEM 79,90
Covers Release 1.2 in detail; 500 pages.
I don't know, if there's an english translation.
```

Messages

1. English version of Linux Kernel Internals *by Naoshad Eduljee*
-> Book Review? *by Josh Du"Bois*
-> Thumbed through it *by Brian J. Murrell*

English version of Linux Kernel Internals

Forum: Other Sources of Information
Re: German book on Linux Kernel Programming (Jochen Hein)
Keywords: German book in Linux Kernel Hacking
Date: Sun, 02 Jun 1996 11:05:31 GMT
From: Naoshad Eduljee <naoshad@pacific.net.sg>

The English version of the german book on Linux Kernel Programming is published by Addison Wesley. Here is the text of the mail I recieved from Addison Wesley when I enquired about the book :

"LINUX Kernal Internals" is priced at $38.68 but will not be available until early June 1996.

Ordering information:

The Book Express will gladly ship your order to any international location. Orders can be prepaid by a valid credit card or a check drawn on a US bank. Orders are shipped to international locations via Air Printed Matter Registered with an estimated delivery time of eight business days from our warehouse in Indiana, USA. Charges for this service are $15.00 for the first book, $8.00 for each additional book on the order.

```
You may order by mail:   Addison-Wesley Book Express
                         One Jacob Way
                         Reading, MA  01867

by phone within the US:    1-800-824-7799
        outside the US:    1-617-944-7273 extension 2188
```

or by fax: 1-617-944-7273

When ordering by fax, please include the title or book number, quantity of each book, credit card number and expiration date, as well as the appropriate shipping address. Please do not send credit card information via the internet; use the fax number listed above for prompt service.

If you need further ordering assistance or title information, please let us know.

Sincerely, ADDISON-WESLEY BOOK EXPRESS

Messages

1. Book Review? *by Josh Du"Bois*
-> Thumbed through it *by Brian J. Murrell*

Book Review?

Forum: Other Sources of Information

Re: German book on Linux Kernel Programming (Jochen Hein)

Re: English version of Linux Kernel Internals (Naoshad Eduljee)

Keywords: German book in Linux Kernel Hacking
Date: Fri, 12 Jul 1996 23:50:34 GMT

From: Josh Du"Bois <duboisj@is.com>

Has anyone read the english version of this book? I'd love go get my hands on a good linux kernel-hacking guide. If anyone has read this and has comments please post them here or email me at duboisj@is.com. If I don't hear that it's worthless, or if it takes a while for anyone to respond, I'll try and pick up a copy and read it myself/post a review here.

Naoshad Eduljee - thanks for the tip,

```
                Josh.
-------------
duboisj@is.com
```

Messages

1. Thumbed through it *by Brian J. Murrell*

Thumbed through it

Forum: Other Sources of Information
Re: German book on Linux Kernel Programming (Jochen Hein)
Re: English version of Linux Kernel Internals (Naoshad Eduljee)
Re: Book Review? (Josh Du"Bois)
Keywords: German book in Linux Kernel Hacking
Date: Thu, 16 Jan 1997 05:18:02 GMT
From: Brian J. Murrell <brian@ilinx.com>

I thumbed through it today at the bookstore. I was particularly interested in how a driver uses a handle in the proc filesystem to write information to a process willing to read, like kmsg does. In my thumbing I did not really get my answer.

The book looked decent but what really disappointed me was that despite it's being a 1996 release, it only covers version 1.2 kernels. I now realize that this is because it was a translation from another book. :-(

I really would like to see an updated version of this book! It would definately be on my bookshelf if it got updated.

b.

Multi-architecture support

Forum: Other Sources of Information
Date: Thu, 23 May 1996 15:45:37 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Michael Hohmuth, of TU Dresden, wrote a new document on Linux's multiple architecture support. A PostScript version is also available.

Messages

1. Linux Architecture-Specific Kernel Interfaces *by Drew Puch*

Linux Architecture-Specific Kernel Interfaces

Forum: Other Sources of Information
Re: Multi-architecture support (Michael K. Johnson)
Keywords: instructions kernel header file intro
Date: Mon, 27 May 1996 17:38:34 GMT
From: Drew Puch <aapuch@eos.ncsu.edu>

Here is some kernel info by **header files**.

Thanks goes out to Michael Hohmuth of TU Dresden, Dept. of Computer Science, OS Group

document corresponds to Linux 1.3.78++

Analysis of the Ext2fs structure

Forum: Other Sources of Information
Date: Sat, 18 May 1996 00:45:42 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

There's not much available on filesystems yet, but Analysis of the Ext2fs structure, by Louis-Dominique Dubeau, is worth visiting.

To add more sources of information:

Forum: Other Sources of Information
Keywords: instructions
Date: Wed, 15 May 1996 15:53:28 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

In order to add another source to this page, simple respond to the page and mention the source. You **can**, if you like, simply type in the URL as your response--just click on the **Respond** button, enter the title of the web page you are connecting to in the **Title** box, click on the URL radiobutton for the **format**, and then type the URL into the large text window entitled **Enter your response here:**.

That is all that is *required*. Just click the **Preview Response** button, and then if it looks right, submit it by clicking on the **Post your Response** button.

If you want to be notified of further changes made to this page, you can **subscribe** to it. Subscribing makes you a member, with special privileges, and *also* puts you on a mailing list. Click on the **Membership** item at the bottom. Members can also edit their posts if they want to make changes later. Also, the more members there are, the more motivated I will be to maintain this new version of the KHG... :-)

If you aren't subscribed, you should probably leave your name and email address, and possibly home page if you have one.

At some point, the URL may be moved from the response list into the body of the article. If that sentence didn't make sense to you, you can safely ignore it.

Thank you for your help!

michaelkjohnson

Loading shared objects - How?

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 12 Aug 1998 19:31:19 GMT
From: Wesley Terpstra <terpstra@unixg.ubc.ca>

Does anyone know where I could find a good document about how shared objects are bound to an ELF executable before runtime? I would like to be able to import symbols from a .so file at runtime based on user input and call the imported symbol (a function). I suspect gdb must do this since it loads shared libraries for programs one debugs and allows one to call the imported functions. I hope to do this as portably as possible. Can anyone out there recommend a document?

Thanks.

How can I see the current kernel configuration?

Forum: The Linux Kernel Hackers' Guide
Date: Sun, 09 Aug 1998 10:32:18 GMT
From: Melwin <tsmelwin@hotmail.com>

Hi all,

I need help on how to see my current kernel configuration.

Thanks Melwin

My mouse no work in X windows

Forum: The Linux Kernel Hackers' Guide
Re: How can I see the current kernel configuration? (Melwin)
Date: Tue, 11 Aug 1998 23:04:48 GMT
From: alfonso santana <alfonsosantana@hotmail.com>

I have a serial mouse in Com1 but I can´t move my mouse in X windows (the cursor don´t move. I tried with mouseconfig, xf86config, XF86Setup, i killed ps of mouse, i used ls -l /dev/mouse and i got /dev/mouse --->/dev/cua0 but not work, i tried with many protocols, but nothing. Please help me. Thanks

The crash(1M) command in Linux?

Forum: The Linux Kernel Hackers' Guide
Keywords: kernel, crash
Date: Fri, 07 Aug 1998 16:29:12 GMT
From: Dmitry <defanov@romance.iki.rssi.ru>

Hi, all!

I know, that there is the crash(1M) command in System V. Is there something like crash(1M) in Linux?

And how to get adresses of kernel's tabeles & structuries (for instance, process table or u-area)?

Thanks!

Dmitry

Where can I gen detailed info on VM86

Forum: The Linux Kernel Hackers' Guide
Keywords: vm86
Date: Thu, 06 Aug 1998 14:53:02 GMT
From: Sebastien Plante <sebasp@cae.ca>

I have a project of emulating 8086 card processor. I think that I can do it under Linux by using VM86.

Where can I get enough information on VM86 to be able to use it ?

How to print floating point numbers from the kernel?

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 04 Aug 1998 16:51:29 GMT
From: <pkunisetty@hotmail.com>

I want to print floating point numbers from kernel module. printk is working fine for integers but not working for floating point numbers. Is there any otherway to print the floating point numbers? Thanks.

PS/2 Mouse Operating in Remote Mode

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 31 Jul 1998 20:00:10 GMT
From: Andrei Racz <andreir@worldnet.att.net>

```
I am experimenting with a PS/2 mouse in remote operation - which is, requesting for a
pointing packet and then waiting for it. No requests - no packets. Usually, the mouse
operates in stream mode, sending continuosly.
 I started with psaux.c code; first I added a timer which would fire the callback
which in turn would send the request.
I ended in hanging the machine - with some message ... iddle task could not sleep and
then AIEE ...scheduling in interrupt.
Trying with a task queue (tq_timer/tq_scheduler) did not help either.
  I have limited experience with Linux.
  I would appreciate some advice on this matter.
Regards, Andrei Racz
```

basic module

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 29 Jul 1998 06:55:21 GMT
From: <vano0023@tc.umn.edu>

what's wrong with this code? It will not print out current->pid

```
#define MODULE
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/module.h>
```

extern struct task_struct *current;

int init_module() {printk("<0>Process command %s pid %i",current->comm, current->pid); return 0;}

void cleanup_module() {printk("<0>Goodbye\n"); }

How to check if the user is local?

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 27 Jul 1998 16:39:18 GMT
From: <jb@nicol.ml.org>

Access to some resources should be limited for local users only (starting Xserver, access to diskette).

I wrote program that walks through /proc/*/stat files and checks if the tty field is between 1024 and 1087. If process has pseudoterminal it checks sid, ppid, sid.. etc. If it find process that is a deamon or has other terminal than vconsole or pseudoterm it tells that it is remote user.

Is it a save way?

Ldt & Privileges

Forum: The Linux Kernel Hackers' Guide
Keywords: LDT Memory Privilege
Date: Fri, 24 Jul 1998 15:17:57 GMT
From: Ganesh <ganesh@cs.sunysb.edu>

```
Hi,
 I need some help with something related to modify_ldt system call
which was added to Linux. I would greatly appreciate your help.

 I am experimenting with a new protection mechanism.
I want to push a user process to privilege level 2 in Linux( by
adding a system call) . If I do this, at the second level of
protection checks in the CPU (ie. at the paging level), the user
process would map to supervisor privileges.This is because x86
maps 0,1,2 to supervisor and 3 to user privilges at the paging
level(that is what I understood from the manual. Please correct me
if I am wrong). Will the process (at PL 2) be able to write to
kernel pages since the protection check would go through at the
page level?

  If so, I guess I can prevent it at the segment level by adding a
check to modify_ldt code to figure out whether the process is
making a pointer to a kernel segment. Is this correct? Anyway, the
process wont be actually able to reload LDTR or change the actual
LD Table directly without a system call(sys_modify_ldt). Or is
there some roundabout way in which a process at privilge level 2
can somehow make an entry in LDT/access the kernel pages?

Again, any help would be greatly appreciated. Thanks a lot.
```

---

skb queues

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 22 Jul 1998 01:19:19 GMT
From: Rahul Singh <rahul_sg@hotmail.com>

I was trying out some stuff that deals with creating Q of the sk_buffs as they are passed from routines in ip_output.c to dev_queue_xmit() in /net/core/dev.c . Using sk_buff_head to do the Q-ing and timer_list to control the rate at which skbs are passed from my routine to dev_queue_xmit().

The code is able to control the rate at which skbs are passed to dev_queue_xmit() but seems to have a few bugs.

The error msgs that I encountered are "killing of interrupt handlers" and "kfree-ing of memory allocated not using kalloc" (when I try to have an upper bound on the Q size).

It would be great if someone could give me a clue about the possible bugs.

Thanks.

Page locking (for DMA) and process termination?

Forum: The Linux Kernel Hackers' Guide
Keywords: dma, page locking, process termination
Date: Tue, 21 Jul 1998 17:40:58 GMT
From: Espen Skoglund <espensk@stud.cs.uit.no>

I have some questions concerning locking of pages belonging to a user-level process.

The scenario I have is as follows: I have implemented a driver for a PCI device (as a module). All processes that wants to access the device will have to do an open on it. When the device-file is opened, some of the device's memory is mapped into the user-level application. Communiction between the application and the device either goes through this buffer, or through the in-kernel module (via ioctl). The device is also able to initiate a DMA transfer all by itself to or from the application's memory.

To be able to do this DMA transfer I will have to pin some pages to memory, do some vitual to physical mapping, and also some scatter-gather mechanism. I am somehow able to cope with all this.

The problem that I am concerned with however, is the case when a DMA operation is going on (or about to be started), and the process that is the destination or source of the DMA transfer dies. What is the best way to make sure that the pages get pinned in memory until the device driver receives a release from the dying process? When this happens, the driver will be able to pause the termination of the process if a dangerous DMA transfer is in progress. When the DMA transfer has finished, it may then free the pinned pages and continue termination.

From what I've seen of the process termination code (I'm doing this in a 2.0.30 kernel), the memory mapings are freed before the open files are released (this rules out the obvious solution). I've thought of two other solutions:

```
1) Using a dummy "ghost thread" associated with the module.
   All locking of user-level pages are also made sharable, and
   shared with this thread.  During the termination of the
   process, the pinned memoryis not freed (I think) beacause
   the memory is also shared with this thread.

2) Making the pages "reserved" instead of locked.  From what
```

```
   I've seen of the kernel code, reserved pages are not freed
   by the free_pte() function.  They are however freed by the
   forget_pte() function which is called by the
   zeromap_page_range() function --- something which is only
   possible to do after doing an open on /dev/mem (right?).

   So, marking pages as reserved should probably work ok.  I
   am however reluctant to do this since I suspect I am
   abusing the whole conecpt of reserved pages.
```

Are there any other ways to accomplish what I am trying to do, or have I misinterpreted the whole kernel-code --- overlooked an amazingly simple fact? (I guess this is a fairly easy thing to do --- misrinterpreting the code I mean. It is afterall not what I would considered a well documented/commented piece of software.)

```
        eSk
```

SMP code

Forum: The Linux Kernel Hackers' Guide
Keywords: SMP code
Date: Mon, 20 Jul 1998 20:05:11 GMT
From: <97yadavm@scar.utoronto.ca>

Is anyone out there know a good source of explanation of the Linux SMP code?

I am writing an OS and after reading the Intel MP spec, after hearing all the problems with SMP on Linux, I bet there is a little more to it. If anyone is an expert in this area and wouldn't mind chatting for a bit, it'd be much appriciated.

Thanks!!!!!

Porting GC: Difficulties with pthreads

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 17 Jul 1998 00:34:11 GMT
From: Talin <Talin@ACM.org>

I'm attempting to get Hans Boehm's gc to run under Linux. (GC is a conservative, incremental, generational garbage collector for C and C++ programs). Apparently it has been ported to older versions of Linux, but the port appears broken. Searching around the web I notice that one or two other people have attempted to get this thing working without success. It's tantalizing because the bloody thing *almost* works. One major problem has to do with pthread support. GC needs to be able to stop a thread an examine it's stack for potential pointers, and there's no defined way in the pthreads API to do this. On SunOS, gc uses the /proc primitives to accomplish this task, unfortunately the Linux /proc lacks the ability to stop a process. Under Irix, it uses an evil hack -- it sends a signal to the pthread, and has the pthread wait on a condition variable *inside the signal handler!* Needless to say, this method is to be avoided if at all possible. Unfortunately, the author of gc says that this is unavoidable due to the limitations of the pthreads API. Does anyone have any ideas for how to go about suspending a thread and getting a copy of it's register set under Linux?

Linux for "Besta - 88"?

Forum: The Linux Kernel Hackers' Guide
Keywords: Besta, Sapsan
Date: Mon, 13 Jul 1998 17:02:17 GMT
From: Dmitry <defanov@romance.iki.rssi.ru.>

Is there anybody, who knows about Linux for Besta-88 workstation. It is based on MVME147 board, but has its own design.

MVME147 Linux

Forum: The Linux Kernel Hackers' Guide
Re: Linux for "Besta - 88"? (Dmitry)
Keywords: Besta, Sapsan
Date: Thu, 16 Jul 1998 11:51:12 GMT
From: Edward Tulupnikov <allin1@allin1.ml.org>

Looks like http://www.mindspring.com/~chaos/linux147/ has some info on the issue. Maybe that'll be helpful.

/proc/locks

Forum: The Linux Kernel Hackers' Guide
Keywords: /proc/locks
Date: Mon, 13 Jul 1998 12:15:29 GMT
From: Marco Morandini <marc2@vedyac.aero.polimi.it>

Can you give me informations about the /proc/locks file (what it is, its format etc....)? I was not able to find them in man pages etc...

Hoping this is the right forum.

Thanks in advance, Marco Morandini

syscall

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 08 Jul 1998 14:00:51 GMT
From: <ppappu@lrc.di.epfl.ch>

```
What is the syscall mechanism for calling system calls when
there is no library interface for the system call.
                        eg how do u call the
get_kernel_syms system call.
```

How to run a bigger kernel ?

Forum: The Linux Kernel Hackers' Guide
Keywords: kernel size
Date: Sat, 04 Jul 1998 05:13:31 GMT
From: Kyung D. Ryu <kdryu@cs.umd.edu>

```
I added some features on Linux 2.0.32. When I modified just a couple of lines and
rebuilt kernel, I was able to reboot and run new kernel.

So, I put a couple of functions in the source files and rebuilt it. The kernel size
got a bit bigger than last one. (kernel size: original:446281, a few chage: 446289,
more change: 446664)

It crashed when it was rebooted and attemped to uncompress the new kernel giving
message "ran out of input data...".

So, I guess the kernel size does matter. How can I make this bigger new kernel run ?

Any parameter to change ?

Thanks in advance
```

Linux Terminal Device Driver

Forum: The Linux Kernel Hackers' Guide
Keywords: device driver
Date: Wed, 24 Jun 1998 12:39:01 GMT
From: Nils Appeldoorn <appeldoo@hio.hen.nl>

```
Hello,

I'm a student from Holland and have received the following assignment:
Write a paper about the Linux Terminaldriver. Explain how it handles
all the interrupts, how the datastructure looks, what the functionality
of its parts is, etcetera, etcetera.

The problem is, we can't get a clear overview of all the needed source files.
We've found /usr/src/linux/drivers/char/tty_io.c but that's probably not
the only one, and we cannot figure it out. It's a little bit fuzzy, for
starters like me.

If you can help me just a little bit, I would appreciate it! Any help,
whatsoever, is good!

Thanks anyway,

Nils Appeldoorn
```

Terminal DD

Forum: The Linux Kernel Hackers' Guide
Re: Linux Terminal Device Driver (Nils Appeldoorn)
Keywords: device driver
Date: Tue, 30 Jun 1998 13:22:00 GMT
From: Doug McNash <dmcnash@computone.com>

Quickly:

serial.c - is the device driver for bare UARTS (8250-16550) others are present for various cards like stallion, digi, specialix et.al. but you probably can't get the tech doc for those. This is the interface with the hardware.

n_tty.c is the code for the line discipline which does the processing of the input/output stream as well as some control function. This is the interface between the "user" and the driver.

tty_io.c and tty_ioctl.c provide some common support functions

tty.h, termio[s].h, termbits.h, ioctl.h, serial.h contain the structure definitions and defines.

DMA to user allocated buffer ?

Forum: The Linux Kernel Hackers' Guide
Keywords: DMA user buffer physio
Date: Mon, 22 Jun 1998 17:07:54 GMT
From: Chris Read <support@active-silicon.com>

```
I am porting an application from Solaris and NT to Linux

I need to DMA fairly large ( >1 MByte ) data areas to a user
assigned buffer (assigned using malloc). I thus need to either

1) Lock down the pages manually in the driver and generate
a physical scatter/gather table for the DMA (I assume that
the malloc'ed pages will not be contiguous in physical memory)

2) Remap the user buffer into physically contiguous memory,
without changing the user virtual mapping (i.e. same user
virtual address)

3) Implement a Unix (Solaris) like physio routine to perform
the DMA in chunks, akin to the read entry point uoi->buf
mappings.

Any pointers or ideas ?
```

allocator-example in A.Rubini's book

Forum: The Linux Kernel Hackers' Guide
Re: DMA to user allocated buffer ? (Chris Read)
Keywords: DMA user buffer physio
Date: Tue, 07 Jul 1998 08:19:04 GMT
From: Thomas Sefzick <t.sefzick@fz-juelich.de>

Have a look into the examples from A.Rubini's 'Linux Device Drivers' book, downloadable from the website advertised on top of the KHG page. Could the 'allocator' code in ftp/v2.1/misc-modules/allocator.[ch] and ftp/v2.1/misc-modules/README.allocator be a solution to your problem?

Patching problems

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 17 Jun 1998 19:27:17 GMT
From: Maryam <mshsaint@direct.ca>

```
Hi Every one
 I currently have kernel v2.0.27  running in my computer and would like to
 patch the rtlinux and do some experience on it.
 So what I have done so far is I downloaded the patches from this page,
 unzipped it and stored it in a disk.
 While listing the files in windows I got these directory names: rtinux-0.5
 and underneath this directory was kernel_patch and etc...

 For mounting the files I used the following command in Linux:
 mount -t msdos /dev/fd0 /mnt
 But after the mounting in /mnt I had different garbled file names such as
 rtlinu~1 & kernel~1. As Michael said I should use exts filesystems which
 now I am using the msdos ones.
 But the problem is how? Is there anyway I can convert the files from msdos
 to ext2 in Linux? Which command should I use?
 "Please, I am stuck here :)"

 Another problem was, I tried to patch those files to the kernel and it
 started asking me the question : File patch to,
  which files should i mention, or is it another problem because of the
 original problem?

 Have a great day

 Thanks in advance,

 Maryam Moghaddas
 Tel/ Fax : (604) 925-4683
```

* Plan for Today, Tomorrow comes Next

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: Patching problems (Maryam)
Date: Wed, 01 Jul 1998 17:22:27 GMT
From: <welch@mcmail.com>

You need to mount a windows formatted disk with long filenames using the vfat filesystem

Ethernet Collision

Forum: The Linux Kernel Hackers' Guide
Keywords: ethernet collision packets sniffing
Date: Fri, 12 Jun 1998 14:19:27 GMT
From: jerome bonnet <bonnet@cran.esstin.u-nancy.fr>

Is there any way in the Linux network architecture that a program in the user space can get extended network statistics from an ethernet driver for example ? I would like to have information on collisions on the ethernet bus... If I can get collision timestamps this would be great. It is to be used to retrive network statistics, in both terms of useful trafic (that one tcpdump does it well) and bus occupation time (that one tcpdump and snifing device do not do it !)...

Cordialement,

Jerome bonnet PhD. Student.

Ethernet collisions

Forum: The Linux Kernel Hackers' Guide
Re: Ethernet Collision (jerome bonnet)
Keywords: ethernet collision packets sniffing
Date: Thu, 25 Jun 1998 14:34:35 GMT
From: Juha Laine <james@cs.tut.fi>

```
Hi.

You can see the network device specific collisions e.g. from
the proc file system (at least with kernels 2.0.34 and
2.1.105). Try yours - just 'cat /proc/net/dev' .

I do not know how you could get notices with time-stamps,
though...

Mabye there is some kind of hack/patch lying somewhere just
waiting for to be found ? :*)

Cheers !
```

Segmentation in Linux

Forum: The Linux Kernel Hackers' Guide
Date: Sun, 07 Jun 1998 06:45:00 GMT
From: Andrew Sampson <sampson@wantree.com.au>

From what I know of Linux, the address space on an Intel machine is split into 3GB for the user and 1GB for the kernel. What I would like to know is how the operating system handles interaction between the two address spaces. I know that a 48-bit pointer is needed, etc, but not how its handled within the C source code of the kernel...and I haven't been able to find anything on it in kernel sources.

If anyone could tell me how this is done I would be very happy.

Thanks,

Andrew Sampson (aspiring programmer)

How can the kernel copy directly data from one process to another process?

Forum: The Linux Kernel Hackers' Guide
Keywords: direct copy, user space, kernel space
Date: Sat, 06 Jun 1998 20:57:05 GMT
From: Jürgen Zeller <zeller@rupert.franken.de>

Hi,

in my first kernel-related project, i want to copy data in the kernel from one process to another.

I have the start point/size of the user-level buffers, but i found no way to do a _direct_ copy.

The copy takes place in a write() call of a character device driver, the source/size is the write buffer/size, the destinition is a other process, currently blocking in its read method.

Of course, i could do a kmalloc, copy to kernel, wake up the read, copy from the kmalloc'd area, kfree the area, but ... that is too much overhead.

Any hints?

Bye,

Jürgen

Use the /Proc file system

Forum: The Linux Kernel Hackers' Guide
Re: How can the kernel copy directly data from one process to another process? (Jürgen Zeller)
Keywords: direct copy, user space, kernel space
Date: Mon, 22 Jun 1998 16:15:21 GMT
From: <marty@twsu.campus.mci.net>

The /proc file system uses a directory for each pid. Do a man 5 proc on linux machine to find out more? Bye, Marty.

Remapping Memory Buffer using vmalloc/vma_nopage

Forum: The Linux Kernel Hackers' Guide
Keywords: mmap, vmalloc, DMA, and Interrupts
Date: Wed, 03 Jun 1998 16:43:15 GMT
From: Brian W. Taylor <taylor@lowell.edu>

Hello All!

Here is an interesting/odd problem that has arisen while trying to setup a large buffer of memory allocated by a kernel driver to be remapped into user space. The driver is for a CCD camera that is DMA and Interrupt driven system and I am able to get good consistant images using "memcpy_tofs()". What I would like to do is to have a large buffer that can be remapped to user space so that the data can be transferred via the network while the CCD is reading out. The camera DMAs a line at a time(1712 bytes) to a kmalloc'ed buffer of 2048 bytes and is copied into the remappable buffer when the problem occurs. Using two different methods I have come up with some really strange results.

The Problem:

When I readout a full frame(~1.3MB of integers), if the data is realatively uniform there is no problem. But if the data is not uniform some of the lines will transfer fine but most will end up with zeros filling up some or all of the values in the line. This will happen no matter how many lines are readout at a time.

The Method:

I am using the 2.0.33 kernel, initially with Matt Welsh's bigphysarea and recently using vmalloc and the example of remapping virtual memory example in Alessandro Rubini's "Linux Device Drivers". From what I have been able to determine the values are good until the copy from the DMA buffer into the remapped buffer.

I am also locking the application memory using the driver and using SCHED_FIFO for priority scheduling. The driver functions very well until I start trying to use the remapped memory.

```
   Any Ideas????
```

```
Thanks In Advance,
Brian W. Taylor
```

Fixed.... strncpy to blame

Forum: The Linux Kernel Hackers' Guide
Re: Remapping Memory Buffer using vmalloc/vma_nopage (Brian W. Taylor)
Keywords: mmap, vmalloc, DMA, and Interrupts
Date: Thu, 11 Jun 1998 20:42:17 GMT
From: Brian W. Taylor <taylor@lowell.edu>

Hello All,

Well, I was able to solve the problem by setting a loop to transfer the data byte by byte. I had been using "strncpy(map_buffer, dma_buffer, dma_count)" to transfer the data.

```
   Now why?
```

Why was this a problem and why did it behave so strangly? When the data was fairly uniform there were no problems. But, when the was a discouniuity in the date the transfer would latch the rest of the dma transfer data to 0?? Very Odd.

strncpy is defined in the kernel and should be usable with vmalloc if anything else.....

Any Ideas??? I would really like to understand the mechanism inside the kernel that caused this problem.

```
   Thanks
   Brian
```

Does memory area assigned by "vmalloc()" get swapped to disk?

Forum: The Linux Kernel Hackers' Guide
Keywords: disk swapping
Date: Sun, 31 May 1998 22:27:59 GMT
From: Saurabh Desai <sauru@juno.com>

Hi Friends !!

I am developing an Informed Prefetching and Caching system as part of the kernel in which it is very important that the memory assigned to hold the data does not get swapped to disk. Without any thought to swapping I used vmalloc() to assign memory to the data structures of the Informed prefetching system. I then realized that the final prefetching system was inefficient in terms of speed when the space assigned to the data structures using vmalloc() was in excess of 10KB. I later realized that most probably the pages assigned using vmalloc() were getting swapped to the disk and when I referenced them, they were brought from the disk rather than memory.

Is there any way to make sure that the assigned pages remain in memory until I release them?

I would appreciate any help in this matter.

Thanx Saurabh Desai

Messages

1. Lock the pages in memory *by balaji@ittc.ukans.edu* NEW
-> How about assigning a fixed size array...does it get swapped too? *by saurabh desai* NEW

Lock the pages in memory

Forum: The Linux Kernel Hackers' Guide
Re: Does memory area assigned by "vmalloc()" get swapped to disk? (Saurabh Desai)
Keywords: disk swapping
Date: Mon, 01 Jun 1998 17:11:06 GMT
From: <balaji@ittc.ukans.edu>

Hi, To make sure a set of pages dont get swapped out to disk just lock them in. Look at the implementation of mlock to find out how thats done...(If you are smart enough you can call the sys_mlock within the kernel) balaji PS: This may not be what you wanted bcos of certain constraints...if so let me know balaji

Messages

1. How about assigning a fixed size array...does it get swapped too? *by saurabh desai* NEW

How about assigning a fixed size array...does it get swapped too?

Forum: The Linux Kernel Hackers' Guide
Re: Does memory area assigned by "vmalloc()" get swapped to disk? (Saurabh Desai)
Re: Lock the pages in memory
Keywords: disk swapping
Date: Thu, 04 Jun 1998 15:27:38 GMT
From: saurabh desai <sauru@juno.com>

I am sure mlock() works fine..and I will also use sys_mlock() in my kernel code if I have to but the problem with it is that my kernel code becomes part of any user process as it is a Prefetching system and mlock() as seen from its implementation checks for the super user privileges suser(). So probably a user process trying to prefetch is going to be deprived of this request.

I was thinking if the fixed size array (e.g. buffer[100]) gets swapped to disk. I am sure not but just wanted to make sure. For my prefetching system I don't need a whole lot of memory. probably about 2-3K max. Hence I can afford to assign a static array.

thanx pal.

Creative Lab's DVD Encore

Forum: The Linux Kernel Hackers' Guide
Keywords: DVD Encore drivers creativelabs Linux
Date: Tue, 26 May 1998 22:01:02 GMT
From: Brandon <kool@goodnet.com>

Hi. I'm totally new to Linux. A friend of mine told me how interesting and engrossing it was so I grabbed it off the net and installed it on my system. One problem... I have Creative Lab's DVD Encore and I haven't found any drivers for it to play movies in X Windows. Does anyone have these drivers? If so, please let me know and I'll be happy to receive them. If not, I have a friend who's willing to write the drivers. So in any case, please let me know.

Brandon (kool@goodnet.com)

TCP sliding window

Forum: The Linux Kernel Hackers' Guide
Keywords: TCP, sliding window
Date: Fri, 15 May 1998 07:26:07 GMT
From: Olivier <aussage@imag.fr>

hello,

I would like to make a trace of the size of the TCP sliding window during a connexion (to see how the size change), but I can't find where in the kernel this variable is. Please, do you have any clue on how I can print the size of the sliding window ?

Many thanks in advance for your help,

Olivier.

Packets and default route versus direct route

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 14 May 1998 12:28:23 GMT
From: Steve Resnick <steve@ducksfeet.com>

Hi

I have a machine with two ethernet cards, two class C networks and roughly 50 IP aliases on various devices.

The two class C networks are distinctly different, i.e., the MSB of the network address is different, and both use a 24 bit netmask.

for the sake of argument:

eth0 is setup on net 1: 192.168.98.0 eth1 is setup on net 2 192.168.99.0

The default route is to our router on eth0, the address is 192.168.98.10

So far, so good.

We bill our customers based on traffic usage and I wrote a libpcap based package to track network usage and calculate aggregates for 5 minute periods and flush this data to disk. I originally wrote this on a Sun machine running solaris 2.5.1.

This worked rather well and I was able to account for all traffic by walking through the ethernet and tcp/ip headers to find the data size.

I rewrote this package to run under 2.0.33 and now I have an odd problem: Packets sent to a particular address all use the same address on the return path.

If, from a different machine on our network at 192.168.98.15, I ping, with record route, to an address on the machine in question, I see:

```
PING 192.168.98.42 (192.168.98.42): 56 data bytes
64 bytes from 192.168.98.42: icmp_seq=0 ttl=64 time=1.3 ms
RR:     192.168.98.15
        192.168.98.42
        192.168.98.10
        192.168.98.15
```

And if I traceroute from the machine in question to another machine on our local network, and that address is on net 2, it still goes out over net 1:

```
traceroute -n 192.168.99.36
traceroute: Warning: Multiple interfaces found; using 192.168.98.10 @ eth0
traceroute to 192.168.99.36 (192.168.99.36), 30 hops max, 40 byte packets
 1  192.168.99.36  0.704 ms  0.606 ms  0.604 ms
```

So, the problem here is that I cannot track the traffic generated by by a particular website, since the source address of all outbound traffic is not the address of the website, but rather the primary address on eth0 (192.168.98.10)

I have checked /proc/sys/net/ipv4/ip_forwarding and the value is 0, so I am assuming ip_forwarding is turned off.

Is there a way to make this work properly, that is to say, if I request data from a particular address the address used on the sending is correct as well?

What else am I missing or should I be looking for?

TIA, Steve

IPv6 description - QoS Implementation - 2 IP Queues

Forum: The Linux Kernel Hackers' Guide
Keywords: IPv6 Networking Kernel
Date: Thu, 07 May 1998 17:16:31 GMT
From: <wehrle>

Hello, I want to implement a QoS support for IPv6. It is a proposal, which is in discussion at the IETF. For the implementation, I have to split the sending queue of IPv6 Packets into two queues. And also, I have to implement a packet classifier. I am sitting now 1 week over the 2.1.98 kernel and searching for the way, a IPv6 packet would take through the kernel of a linux software router. I do not know, where the packets are inqueued into the IPv6 Sending Queue, and where they were dequeued by the devices. I need to know this, because I have to change the behaviour of this functions. Can anybody help me ? Does anybody know a (very good) description of the ipv6 implementation, their data structures, ...

Many thanks

Ciao Klaus

See the kernel IPv4 implementation documentation

Forum: The Linux Kernel Hackers' Guide
Re: IPv6 description - QoS Implementation - 2 IP Queues
Keywords: IPv6 Networking Kernel
Date: Thu, 25 Jun 1998 14:03:43 GMT
From: Juha Laine <james@cs.tut.fi>

```
Hi.

If you haven't read the "The Linux Kernel" guide networking
sections, they might give you a hint.

They are NOT IPv6 specific, but I assume that the implementation
is not so different from the IPv4. The data structures will
differ but I think that operations are quite similar...

"The Linux Kernel" guide is located on the LDP homepages.
[ http://sunsite.unc.edu/mdw/ ]

Hope this helps.
```

writing to user file directly from kernel space, How can it be done?

Forum: The Linux Kernel Hackers' Guide
Keywords: driver write memory file help
Date: Wed, 06 May 1998 08:54:05 GMT
From: Johan <d3sunden@dtek.chalmers.se>

Hello, I am collecting data at a high rate into a circular buffer residing in my driver. I want this data to be written to disk or network. To avoid unnecessary copying from kernel space to user space to kernel space again I want a application to open a file or socket and then do a ioctl to my driver which I then want to start performing write's from the buffer to the file/socket using the file operation write in the file block of the opened file. Unfortenaly for me I get a page fault when I try f_op->write(...). My buffer is in the kernel whith a kernel address but the write wants a virtual address (I guess).

How can I get around this? Other ideas?

I use Linux 2.0.33

Thanks in advance/ Johan

how can i increase the number of processes running?

Forum: The Linux Kernel Hackers' Guide
Keywords: proc
Date: Wed, 06 May 1998 08:30:39 GMT
From: ElmerFudd <elfuddo@hotmail.com>

i can't seem to get more than 240 processes running (programs seg fault and nasty stuff like that) i have been looking into it and i think it must have something to do with a limit on /proc? (i'm running 2.1.33 and libc 5.4.17)

Please email me at elfuddo@hotmail.com if you are able to shed some light on this problem, thanks

How do I change the amount of time a process is allowed before it is pre-empted?

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 21 Apr 1998 14:20:10 GMT
From: <Escher@dn101aw.cse.eng.auburn.edu>

I need help badly....I need to know how to change this field, where it is located, etc.

Network device stops after a while

Forum: The Linux Kernel Hackers' Guide
Keywords: network device, ioctl
Date: Tue, 21 Apr 1998 13:00:25 GMT
From: Andrew Ordin <ordin@dialup.ptt.ru>

I have a strange problem with a network device driver i have written. The device runs under IP for a while and then kernel stops passing ioctl()s down to the driver. The ioctl() is used by a process to communicate with the device.

Anybody knows where the dog is buried?

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: Network device stops after a while (Andrew Ordin)
Keywords: network device, ioctl
Date: Sat, 25 Apr 1998 13:21:35 GMT
From: Andrew <waddington@usa.net>

Precisely which dog is this???

Does MMAP work with Redhat 4.2?

Forum: The Linux Kernel Hackers' Guide
Keywords: MMAP memory error
Date: Tue, 21 Apr 1998 10:54:04 GMT
From: Guy <theguy@ionet.net>

Hi,

I'm getting an error with MMAP on Redhat 4.2... The same code worked just fine when compiled on a Slakware 1.2.13 system...

The code actually compiles just fine, but I get a runtime error which says, "Segmentation Fault (Core Dumped)" when I try to access the memory -- I later found that MMAP was returning a "EINVAL" error...

Does the MMAP function work on Redhat 4.2?

What are the differences between Redhat 4.2 and Slakware 1.2.13 with respect to the MMAP function?

I've verified that my start and length values are appropriate (I even tried setting my start value to zero, but that didn't seem to work). How do I find a PAGESIZE boundary? How do I "allign" this?

```
                        thanks for your help,
                                 Guy
                         (theguy@ionet.net)
```

Messages

1. Yes, it works just fine. *by Michael K. Johnson* NEW
-> It Works! Thanks! *by Guy* NEW

Yes, it works just fine.

Forum: The Linux Kernel Hackers' Guide
Re: Does MMAP work with Redhat 4.2? (Guy)
Keywords: MMAP memory error
Date: Tue, 21 Apr 1998 12:12:15 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

mmap() works fine on a Red Hat 4.2 system, which ships a perfectly standard Linux kernel. A great deal of the source code included in the distribution uses mmap() explicitly, and every dynamically loaded program uses mmap() implicitly in the dynamic loader.

I suggest "man mmap" as a start. Notice the reference at the bottom of the page to getpagesize(), and consider the modulus operator (%). That should help you out... :-)

You'll also find sample code that uses mmap() at http://www.redhat.com/~johnsonm/lad/src/map-cat.c.html Other sample source you might find useful is at http://www.redhat.com/~johnsonm/lad/src/

Messages

2. It Works! Thanks! *by Guy* NEW

It Works! Thanks!

Forum: The Linux Kernel Hackers' Guide
Re: Does MMAP work with Redhat 4.2? (Guy)
Re: Yes, it works just fine. (Michael K. Johnson)
Keywords: MMAP memory error
Date: Tue, 21 Apr 1998 14:13:58 GMT
From: Guy <theguy@ionet.net>

Thanks!

What about mprotect?

Forum: The Linux Kernel Hackers' Guide
Re: Does MMAP work with Redhat 4.2? (Guy)
Re: Yes, it works just fine. (Michael K. Johnson)
Keywords: MMAP memory error
Date: Fri, 17 Jul 1998 04:48:53 GMT
From: Sengan Baring-Gould <*unknown*>

```
If I use the example given by man mprotect on Redhat 4.2, the
program does not run:

./mprotect

Couldn't mprotect: Invalid argument

Why?

Thanks,

Sengan

       #include <stdio.h>
       #include <stdlib.h>
       #include <errno.h>
       #include <sys/mman.h>

       int
       main(void)
       {
           char *p;
           char c;

           /* Allocate a buffer; it will have the default
              protection of PROT_READ|PROT_WRITE. */
           p = malloc(1024);
           if (!p) {
               perror("Couldn't malloc(1024)");
               exit(errno);
           }

           c = p[666];         /* Read; ok */
```

```
        p[666] = 42;        /* Write; ok */

        /* Mark the buffer read-only. */
        if (mprotect(p, 1024, PROT_READ)) {
            perror("Couldn't mprotect");
            exit(errno);
        }

        c = p[666];         /* Read; ok */
        p[666] = 42;        /* Write; program dies on SIGSEGV */

        exit(0);
    }
```

multitasking

Forum: The Linux Kernel Hackers' Guide
Keywords: multitasking
Date: Fri, 17 Apr 1998 17:01:10 GMT
From: Dennis J Perkins <dperkins@btigate.com>

I'm starting to learn about how the kernel works. Does Linux use cooperative or preemptive multitasking? I know that the scheduler is called when returning from a system call, but does this mean that Linux uses cooperative multitasking, since a timer interrupt does not force a context xwitch?

Answer

Forum: The Linux Kernel Hackers' Guide
Re: multitasking (Dennis J Perkins)
Keywords: multitasking
Date: Mon, 20 Apr 1998 15:32:10 GMT
From: David Welch <welch@mcmail.com>

Both. User level processes can preempted in user mode but code running in kernel mode (either by using a system call or by a dedicated kernel thread) is only preempted when it chooses to give up control

multitasking

Forum: The Linux Kernel Hackers' Guide
Re: multitasking (Dennis J Perkins)
Re: Answer (David Welch)
Keywords: multitasking
Date: Tue, 21 Apr 1998 17:41:27 GMT
From: Dennis J Perkins <dperkins@btigate.com>

So, preemptive multitasking does not mean that a process stops running, that is, it is no longer current, as soon as do_timer decrements its priority to the point where it no longer has the highest priority? It continues running until it returns from a system call?

answer

Forum: The Linux Kernel Hackers' Guide
Re: multitasking (Dennis J Perkins)
Re: Answer (David Welch)
Re: multitasking (Dennis J Perkins)
Keywords: multitasking
Date: Tue, 28 Apr 1998 12:59:48 GMT
From: David Welch <welch@mcmail.com>

Not quite. If a process is executing a system call and interrupts are disabled then it will not be preempted (because the timer interrupt can't happen). Since interrupts are automatically disabled on entry to a system call and are only reenabled when the process will have to wait for a long time, most of time a process won't be preempted inside the kernel. However it can be preempted at any time when running in user mode.

---

linux on sparc

Forum: The Linux Kernel Hackers' Guide
Keywords: sparc kernel linux
Date: Mon, 06 Apr 1998 10:52:01 GMT
From: darrin hodges <darrin.hodges@qmi.com.au>

has anybody been able to build a recent (2.0.33) kernel on a sparc, it seems that many of the defines are missing, eg GFP_IO is defined in the i386 & alpha include tree, yet not in the sparc tree. I cant seem to find much about linux-sparc, im familiar enough with i386 assembly and arch, but not sparc arch. if anybody has any clues, share with me? :)

How to call a function in user space from inside the kernel ?

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 02 Apr 1998 16:03:21 GMT
From: Ronald Tonn <tonn@infotech.tu-chemnitz.de>

Is there a way to call a function inside an application from a device driver (in kernel mode)?

I've come across this problem while working on an my ATM network driver. When opening a channel the ATM application calls the driver with a pointer to a receive function that should be called whenever data is received on that channel.

If I use this pointer (which points into user space) inside the driver as shown below the whole systems crashes appearently because I'm trying to execute a user program in kernel mode.

```
        ...
        receive_ptr(buffer, buffer_size);
        ...
```

Does anyone know how to fix this problem?

Thanks,

Ronald

How to call a user routine from kernel mode

Forum: The Linux Kernel Hackers' Guide
Re: How to call a function in user space from inside the kernel ? (Ronald Tonn)
Date: Thu, 09 Apr 1998 21:11:48 GMT
From: David Welch <welch@mcmail.com>

There are several possible solutions

i) On kernels before 2.1.xx the kernel uses a different code segment to user programs. To call user space would require a far call. The functions get_user and put_user similarly use a segment override to access user space. Calls are far more complicated because (for protection) the processor won't allow a user routine to return to a more priveleged code segment. The best way round this would be to use a similar method to signal handlers i.e. setup a routine on the stack which on return from the signal handler executes a system call which returns to kernel mode. See arch/i386/kernel/signal.c for more information.

The problem is further complicated because on the i386 executing a system call will restore the default kernel stack. The vm86 routines allocate a new kernel stack so when a GPF happens in vm86 mode they can restore the original stack and return to the caller process. You may be able to use a similar method.

ii) On kernels after 2.1.xx the kernel segment has the same base as user space. It should be possible to directly call a user routine and have it return normally. However the user routine will have normal kernel priveleges (a big security hole!).

iii) Use kerneld. If the desired routine can be written as a seperate program then kerneld can be called from kernel mode to execute it. An example is calling the request-routine script when a network connection is attempted.

iv) If the calling process is multithreaded then you should be able to use the ipc semaphores from kernel mode to signal to another thread to execute the routine.

Can I map kernel (device driver) memory into user space ?

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 02 Apr 1998 15:28:36 GMT
From: Ronald Tonn <tonn@infotech.tu-chemnitz.de>

I'm trying to find out a way to map memory that has been allocated by a device driver (of course in kernel mode) into user space so that applications can have access to it.

This issue has come up while writing an ATM network driver. If send or receive buffers (allocated by the driver) could be passed directly to an application that makes use of the driver, data could be passed with a single-copy operation. e.g the application requests a send buffer from the driver, copies the data into it and then forces the driver to transmit the buffer.

Right now I'm using copy_from(to)_user in order to get the data from a user allocated buffer into the driver allocated send buffer and vice versa.

Any help would be much appreciated.

Thanks,

```
Ronald
```

driver x_open,x_release work, x_ioctl,x_write don't

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 31 Mar 1998 19:41:58 GMT
From: Carl Schwartz <schwcarl@e-z.net>

```
Using RedHat 5.0 and following KHG I performed the following steps in developing a
driver 'x':
1) created x.o with gcc -O -DMODULE -D__KERNEL__
2) created /dev/x crw-r--r-- 126 0
3) insmod ./x.o
4) lsmod listed it but ksyms did not
5) user root: fildef = open('/dev.x',O_RDWR);  (fildef = 17)
6) user root:ioctl(fildef,_IOW(126,1,sizeof(daud)),(int)&daud)
     returns -1 as well as does all other ioctl's and write's
     I try from user app and do not print "printk's".

7) rmmod removes it OK.  It seems that 'open' and 'release'
     are the only functions that work (perform as expected and
     "printk's" work).
```

I copied device file_operations, ioctl and write parameter lists from KHG, basically replacing 'foo' with 'x'.

I copied 'x.o' to /lib/modules/2.0.31/char and added 'alias char-major-126 x' to conf.module. Depmod -a does not add it to modules.dep and Modprobe doesn't know that 'x' exists.

Messages

1. Depmod Unresolved symbols? *by Carl Schwartz* NEW

Depmod Unresolved symbols?

Forum: The Linux Kernel Hackers' Guide
Re: driver x_open,x_release work, x_ioctl,x_write don't (Carl Schwartz)
Date: Tue, 31 Mar 1998 22:50:12 GMT
From: Carl Schwartz <schwcarl@e-z.net>

Sorry, I forgot to click '?' on original message. One of my original problems that I just discovered is that I had put 'x.o' under /lib/modules/2.0.31/ 'char' instead of 'misc'. Relating 'modules' to 'drivers', I guess I had created the 'char' subdirectory which Modprobe apparently does not search. My remaining problem seems to be that Depmod -a does not resolve the symbols, but does go ahead and put 'x' in the modules.dep. I need to determine what, if not Depmod, updates the symbol table. The symbols returned by depmod -e are all found in ksyms and some of them work just fine in the 'cleanup', 'init' and 'open' functions which work OK. I haven't found the *.c source to help in figuring out how things fit together for the module utilities, just the *.o files.

How to sleep for x jiffies?

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 26 Mar 1998 17:02:51 GMT
From: Trent Piepho <xyzzy@u.washington.edu>

I'd like to have a delay in a device driver of X jiffies, where X = 1 on intel and 8 on alpha. What's a clean and friendly way to do this? udelay(8333)?

Use add_timer/del_timer (in kernel/sched.c)

Forum: The Linux Kernel Hackers' Guide
Re: How to sleep for x jiffies? (Trent Piepho)
Date: Tue, 23 Jun 1998 22:27:06 GMT
From: Amos Shapira <amos-khg@gezernet.co.il>

At least for 2.1.xx, use the *_timer function family as defined in kernel/sched.c.

The "Linux Internals" book (not quite recommanded, better than nothing from a quick glance) mentioens only these functions, but I see other functions with interesting names:

mod_timer which seems to change an existing timer in-place

detach_timer which seems to do what del_timer does (actually it's used by del_timer) without clearing the timer_list 'next' and 'prev'.

There are a few interesting comments in include/linux/timer.h which describe a few more functions.

One thing I learned from the book is that when you call add_timer, you specify the absolut time in jiffies (i.e. for one sec say 'jiffies + (1*HZ)')

Adding code to the Linux Kernel

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 25 Mar 1998 18:04:48 GMT
From: <barry@skynet.csn.ul.ie>

```
Hi all,
       I am currently attempting my first kernel hack. I am adding code into the IP
layer to randomly drop packets.
I have two questions:
```

1) I want to be able to include a header file called random.h (usr/src/linux/include/net). This is so that I can generate random numbers. When I include this in my own C file, parse errors appear everywhere. Is this the right place to include this header file?

2) I would eventually like to get a random number generated based on seeding it from the current system time. This is so that I can get a new random number on each call to ip_input. Is there a way around not being able to include standard lib files in the kernel (ie time.h, stdlib.h etc)?

Also it there any documentation on adding such modules to the kernel. It would greatly simplify my task.

Thanks

Patrick

/dev/random

Forum: The Linux Kernel Hackers' Guide
Re: Adding code to the Linux Kernel
Date: Sun, 29 Mar 1998 11:20:52 GMT
From: Simon Green <sgreen@emunet.com.au>

Recent kernels (at least 2.0.31 and later) include a device file called /dev/random (no. 1,8). You can read from this device file to get as many random bytes as you need.

Just read in bytes a pair/quad at a time and typecast to int.

This way, you don't need to keep reseeding the generator.

Simon

MSG_WAITALL flag

Forum: The Linux Kernel Hackers' Guide
Date: Sun, 22 Mar 1998 16:38:16 GMT
From: Leonard Mosescu <lemo@lmct.sfos.ro>

Why Linux doesn't support MSG_WAITALL flag in the recv() call ?

possible bug in ipc/msg.c

Forum: The Linux Kernel Hackers' Guide
Keywords: ipc bug msg
Date: Sat, 21 Mar 1998 04:57:44 GMT
From: Michael Adda <m_photon@usa.net>

```
hi
first, i hope that this is the right place, :-> ,
since i an not sure about the 'finding' ...
i need an advice. i am currently reading the kernel's code
systematiclay, and i believe i stumbled into a bug in ipc/msg.c
lines 326,329. i am talking about kernel 2.0.30-2.0.33 ( which i
am working with ) and not about the development kernels... please
read the relevent code ...
since we are no longer ( between this lines ) in atomic
operations, someone can suspend are in say line 326, recieve the
current message ( the one we have nmsg as pointer to ) and leave
us with pointer to garbage...
i belive that we should put lines 326-329 in cli/restoreflags()
pair after checking that the message is valid via the pointer flag
( not IPC_UNUSED/IPC_NOOID ).
i hope that i am not bothering you for nothing...
i have a possible patch.

    thank you for your time
      Michael ( m_photon@usa.net )
```

scheduler Question

Forum: The Linux Kernel Hackers' Guide
Keywords: scheduler sleep priority
Date: Sat, 07 Mar 1998 12:12:06 GMT
From: Arne Spetzler <arne@smooth.netzservice.de>

Hi!

For my diploma i'am modifying the Linux Kernel to support ACL's. At some places the Kernel has to sleep on interal structures and after wake up the process has to run as soon as possible to minimize delay (e.g. the acl-header-cache). As i know the traditional approach (Maurice J. Bach: The Design of the UNIX Operating System) is to set the process to a fixed (and high) "sleep priority" on which the process will run after wake up.

But i couldn't find the related code in the Linux kernel (2.0.33).

Does anybody know how Linux doing this??

thanks for every help

Arne

Messages

1. Untitled *by Ovsov* NEW

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: scheduler Question (Arne Spetzler)
Keywords: scheduler sleep priority
Date: Sun, 08 Mar 1998 00:46:24 GMT
From: Ovsov <ovsov@bveb.belpak.minsk.by>

Actually Linux's wake_up_process does nothing but changing a process' state from TASK_(UN)INTERRUPTIBLE to TASK_RUNNING and then it may be chosen for running the next time schedule() is invoked depending on its priority (counter-parameter in task_struct for the processes scheduled under OTHER-policy and rt_proirity-parameter for real-time processes scheduled under FIFO or RobinRound (RR)).

Best regards. Kostya.

thanks

Forum: The Linux Kernel Hackers' Guide
Re: scheduler Question (Arne Spetzler)
Re: Untitled (Ovsov)
Keywords: scheduler sleep priority
Date: Sat, 16 May 1998 06:48:05 GMT
From: arne spetzler <*unknown*>

Hmmm. Thats exactly what i found.

So i think there is no chance to give my proccesse a higher priority in order to run quicker :-(

arne

File Descriptor Passing?

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 27 Feb 1998 06:58:32 GMT
From: The Llamatron <mc2@geocities.com>

Quick questions: Has file descriptor passing been implemented yet, which kernel version to use, and which method (SYSV or Berkeley)?

Thanks bunches and bunches!

Linux SMP Scheduling

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 26 Feb 1998 06:50:08 GMT
From: Angela <okchan@cs.hku.hk>

Dear sir, i am a student working on a project about Linux SMP scheduling.

i would be very grateful if you could kindly help me answering the following questions:

From the source code:"sched.c" - what is the constant "NO_PROC_ID" used for? and how to determine its value?

From the source code: "fork.c" - what is the use of the integer "lock_depth" and what are the meaning of the values it can have(eg. 1)?

Regards, Angela

Finding definitions in the source

Forum: The Linux Kernel Hackers' Guide
Re: Linux SMP Scheduling (Angela)
Date: Mon, 16 Mar 1998 16:34:54 GMT
From: Felix Rauch <rauch@inf.ethz.ch>

The definition of NO_PROC_ID can be fount by, e.g., the following command:

```
find . -name "*.[ch]" -print | xargs grep NO_PROC_ID | grep #define
```

(assuming you are in the Linux-source-directory).

This will show that NO_PROC_ID is defined as follows:

```
#define NO_PROC_ID 0xFF
```

Re: Linux SMP Scheduling

Forum: The Linux Kernel Hackers' Guide
Re: Linux SMP Scheduling (Angela)
Keywords: SMP Linux
Date: Fri, 27 Feb 1998 13:35:31 GMT
From: Franky <genie@ucsd.com>

```
Hello:

It seems to me that NO_PROC_ID is just an indicator that no processes is scheduled
for execution on this processor(i.e. it is free to be used for running next task).

As of lock_depth - I did not find anything usefull about it - I am not so good at
Linux SMP kernel features, sorry :)))
```

Difference between ELF and kernel file

Forum: The Linux Kernel Hackers' Guide
Keywords: ELF binary kernel gcc output
Date: Tue, 17 Feb 1998 22:43:54 GMT
From: Thomas Prokosch <Thomas.Prokosch@jk.uni-linz.ac.at>

I would like to know the difference between a normal ELF executable and a kernel image. Of course I know that the ELF file format contains pointers and different sections and a kernel file should be rather a binary image only, but how are both related to each other? Both, ELF and kernel image, are output by gcc, how do I convert an ELF binary to a kernel image?

How kernel communicates with outside when it's started?

Forum: The Linux Kernel Hackers' Guide
Keywords: kernel,file system,
Date: Mon, 16 Feb 1998 23:03:17 GMT
From: Xuan Cao <cao@cobalt.chem.uidaho.edu>

I am writing some kernel modules that is supposed to executed after the init process has been started. The purpose of these modules is to write some kernel data out to a normal file. But I don't know what kinds of functions are available for those modules. Some functions such as open(), read(), write() which worked before init process don't work in those kernel modules after init process is started. I guess the main problem might be the kernel don't have control over the system after init process is started. But how can I get some of the data out of kernel to a file by using the kernel modules?

The problem is can or how I use some functions (file operations, IPC, etc) in a kernel module which is only executed after the system has started and is running normally.

Thanks in advance!

Printing to the kernel log

Forum: The Linux Kernel Hackers' Guide
Re: How kernel communicates with outside when it's started? (Xuan Cao)
Keywords: kernel,file system,
Date: Tue, 24 Feb 1998 20:59:16 GMT
From: Thomas Prokosch <Thomas.Prokosch@jk.uni-linz.ac.at>

During my recent studies of the Linux kernel I came across a function "fprintk" which prints kernel error messages in the approprate logfile. Perhaps you could use this?

The way from a kernel hackers' idea to an "official" kernel?

Forum: The Linux Kernel Hackers' Guide
Keywords: generic driver major number
Date: Fri, 13 Feb 1998 12:51:42 GMT
From: Roger Schreiter <schreite@helena.physik.uni-stuttgart.de>

We (see below) want to write a generic MLC driver (MLC stands for multiple logical channel). MLC is perhaps not as important as SCSI, but has a similar structure - HP thinks about making it an IEEE standard.

Is it the right way for us to discuss our ideas about how to build the new driver in the appropriate KHG list? Where do we have to send the code when ready, for it will be part of future stable kernels? Who decides, which major numbers and so on will be assigned to our new type of hardware driver?

Perhaps some KHG readers have already discovered on the Linux Parallel Port Home Page (http://www.torque.net/linux-pp.html) the new link to the "Linux driver for the HP officejet" project (http://www.ifs.physik.uni-stuttgart.de/Personal/RSchreiter/hpoj/). The HP officejet communicates via MLC. We think further MLC capable devices will follow, so it will be useful to write a generic MLC driver.

Messages

1. linux-kernel@vger.rutgers.edu *by Michael K. Johnson* NEW

linux-kernel@vger.rutgers.edu

Forum: The Linux Kernel Hackers' Guide
Re: The way from a kernel hackers' idea to an "official" kernel? (Roger Schreiter)
Keywords: generic driver major number
Date: Fri, 13 Feb 1998 16:41:17 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

You need to subscribe to linux-kernel@vger.rutgers.edu (send email with **BODY**, not subject, of `subscribe linux-kernel` to majordomo@vger.rutgers.edu.

Discuss your ideas there.

Lastly, see Writing a parport Device Driver.

Writing a parport Device Driver

What parport does

One purpose of parport is to allow multiple device drivers to use the same parallel port. It does this by sitting in-between the port hardware and the parallel port device drivers. When a driver wants to talk to its parallel port device, it calls a function to "claim" the port, and "releases" the port when it is done.

Another thing that `parport` does is provide a layer of abstraction from the hardware, so that device drivers can be architecture-independent in that they don't need to know which style of parallel port they are using (those currently supported are PC-style, Archimedes, and Sun Ultra/AX architecture).

Interface to parport

Finding a port

To obtain a pointer to a linked list of parport structures, use the `parport_enumerate` function. This returns a pointer to a `struct parport`, in which the member `next` points to the next one in the list, or is `NULL` at the end of the list.

This structure looks like (from linux/include/linux/parport.h):

```
/* A parallel port */
struct parport {
        unsigned long base;     /* base address */
        unsigned int size;      /* IO extent */
        char *name;
        int irq;                /* interrupt (or -1 for none) */
        int dma;
        unsigned int modes;

        struct pardevice *devices;
        struct pardevice *cad;  /* port owner */
        struct pardevice *lurker;

        struct parport *next;
        unsigned int flags;

        struct parport_dir pdir;
        struct parport_device_info probe_info;

        struct parport_operations *ops;
        void *private_data;     /* for lowlevel driver */
};
```

Device registration

The next thing to do is to register a device on each port that you want to use. This is done with the `parport_register_device` function, which returns a pointer to a `struct pardevice`, which you will need in order to use the port.

This structure looks like (again, from linux/include/linux/parport.h):

```
/* A parallel port device */
struct pardevice {
        char *name;
        struct parport *port;
        int (*preempt)(void *);
        void (*wakeup)(void *);
        void *private;
        void (*irq_func)(int, void *, struct pt_regs *);
        int flags;
        struct pardevice *next;
        struct pardevice *prev;
        struct parport_state *state;     /* saved status over preemption */
};
```

There are two types of driver that can be registered: "transient" and "lurking". A lurking driver is one that wants to have the port whenever no-one else has it. PLIP is an example of this. A transient driver is one that only needs to use the parallel port occasionally, and for short periods of time (the printer driver and Zip driver are good examples).

Claiming the port

To claim the port, use `parport_claim`, passing it a pointer to the `struct pardevice` obtained at device registration. If `parport_claim` returns zero, the port is yours, otherwise you will have to try again later.

A good way of doing this is to register a "wakeup" function: when a device driver releases the port, other device drivers that are registered on that port have their "wakeup" functions called, and the first one to claim the port gets it. If the parport claim fails, you can go to sleep; when the parport is free again, your wakeup function can wake you up again. For example, declare a global wait queue for each possible port that a device could be on:

```
static struct wait_queue * wait_q[MAX_MY_DEVICES];
```

The wakeup function looks like:

```
void my_wakeup (void * my_stuff)
{
        /* this is our chance to grab the parport */

        struct wait_queue ** wait_q_pointer = (struct wait_queue **) my_stuff;

        if (!waitqueue_active (wait_q_pointer))
                return; /* parport has messed up if we get here */

        /* claim the parport */
        if (parport_claim (wait_q_pointer)))
                return; /* Shouldn't happen */

        wake_up(wait_q_pointer);
}
```

Then, in the initialisation code, do something like:

```
struct pardevice * pd[MAX_MY_DEVICES];

int my_driver_init (void)
{
  struct parport * pp = parport_enumerate ();

  int count = 0;
  while (pp) { /* for each port */

    /* set up the wait queue */
    init_waitqueue (&wait_q[count]);

    /* register a device */
    pd[count] = parport_register_device (pp, "Me",
        /* preemption function */        my_preempt,
        /* wakeup function */            my_wakeup,
        /* interrupt function */         my_interrupt,
        /* this driver is transient */   PARPORT_DEV_TRAN,
        /* private data */               &wait_q[count]);

    /* try initialising the device */
    if (init_my_device (count) == ERROR)
      /* failed, so unregister */
      parport_unregister_device (pd[count]);
    else if (++count == MAX_MY_DEVICES)
      /* can't handle any more devices */
      break;
  }
```

Then a typical thing to do to obtain access to the port would be:

```
        if (parport_claim (pd[n]))
                /* someone else had it */
                sleep_on (&wait_q[n]);  /* will wake up when wakeup */
                                        /* function called */

        /* (do stuff with the port here) */

        /* finished with the port now */
        parport_release (pd[n]);
```

Using the port

Operations on the parallel port can be carried out using functions provided by the parport interface:

```
struct parport_operations {
        void (*write_data)(struct parport *, unsigned int);
        unsigned int (*read_data)(struct parport *);
        void (*write_control)(struct parport *, unsigned int);
        unsigned int (*read_control)(struct parport *);
        unsigned int (*frob_control)(struct parport *, unsigned int mask, unsigned
int val);
        void (*write_econtrol)(struct parport *, unsigned int);
        unsigned int (*read_econtrol)(struct parport *);
        unsigned int (*frob_econtrol)(struct parport *, unsigned int mask, unsigned
int val);
        void (*write_status)(struct parport *, unsigned int);
        unsigned int (*read_status)(struct parport *);
        void (*write_fifo)(struct parport *, unsigned int);
        unsigned int (*read_fifo)(struct parport *);

        void (*change_mode)(struct parport *, int);

        void (*release_resources)(struct parport *);
        int (*claim_resources)(struct parport *);

        unsigned int (*epp_write_block)(struct parport *, void *, unsigned int);
        unsigned int (*epp_read_block)(struct parport *, void *, unsigned int);

        unsigned int (*ecp_write_block)(struct parport *, void *, unsigned int, void
(*fn)(struct parport *, void *, unsigned int), void *);
        unsigned int (*ecp_read_block)(struct parport *, void *, unsigned int, void
(*fn)(struct parport *, void *, unsigned int), void *);

        void (*save_state)(struct parport *, struct parport_state *);
        void (*restore_state)(struct parport *, struct parport_state *);

        void (*enable_irq)(struct parport *);
        void (*disable_irq)(struct parport *);
        int (*examine_irq)(struct parport *);

        void (*inc_use_count)(void);
        void (*dec_use_count)(void);
};
```

However, for generic operations, the following macros should be used (architecture-specific parport implementations may redefine them to avoid function call overheads):

```
/* Generic operations vector through the dispatch table. */
#define parport_write_data(p,x)           (p)->ops->write_data(p,x)
#define parport_read_data(p)              (p)->ops->read_data(p)
#define parport_write_control(p,x)        (p)->ops->write_control(p,x)
#define parport_read_control(p)           (p)->ops->read_control(p)
#define parport_frob_control(p,m,v)       (p)->ops->frob_control(p,m,v)
#define parport_write_econtrol(p,x)       (p)->ops->write_econtrol(p,x)
#define parport_read_econtrol(p)          (p)->ops->read_econtrol(p)
#define parport_frob_econtrol(p,m,v)      (p)->ops->frob_econtrol(p,m,v)
#define parport_write_status(p,v)         (p)->ops->write_status(p,v)
#define parport_read_status(p)            (p)->ops->read_status(p)
#define parport_write_fifo(p,v)           (p)->ops->write_fifo(p,v)
#define parport_read_fifo(p)              (p)->ops->read_fifo(p)
#define parport_change_mode(p,m)          (p)->ops->change_mode(p,m)
#define parport_release_resources(p)      (p)->ops->release_resources(p)
#define parport_claim_resources(p)        (p)->ops->claim_resources(p)
```

Releasing the port

When you have finished the sequence of operations on the port that you wanted to do, use `release_parport` to let any other devices that there may be have a go.

Unregistering the device

If you decide that you don't want to use the port after all (perhaps the device that you wanted to talk to isn't there), use `parport_unregister_device`.

Something to bear in mind: interrupts

Parallel port devices cannot share interrupts. The parport code shares a parallel port among different devices by means of scheduling - only one device has access to the port at any one time. If a device (a printer, say) is going to generate an interrupt, it could do it when some other driver (like the Zip driver) has the port rather than the printer driver. That would lead to the interrupt being missed altogether. For this reason, drivers should poll their devices unless there are no other drivers using that port. To see how to do this, you might like to take a look at the printer driver.

Curious about sleep_on_interruptible() in ancient kernels.

Forum: The Linux Kernel Hackers' Guide
Keywords: sleep_on wake_up signal history
Date: Thu, 12 Feb 1998 21:19:36 GMT
From: Colin Howell <howell@cs.stanford.edu>

I've recently become interested in the design and history of the Linux kernel, so I decided to start at the beginning. That's right, 0.01. :-)

Of course, I realize such old code may be very buggy or incomplete, but that's part of what makes it interesting.

Anyway, I noticed a peculiarity in the scheduler function sleep_on_interruptible(). Before kernel version 1.0, both forms of sleep_on() just waited on a task_struct pointer variable, setting that variable to point to the last caller of sleep_on(). wake_up() would thus only directly awaken the last caller of sleep_on(); that caller was responsible for waking up the previous caller (a pointer to which it had saved), and so on down the chain. There was no explicit wait queue, only what you might call an "implicit wait stack".

For uninterruptible waits, this will do, but if the wait is interruptible, there seems to be a problem. Suppose one of the tasks waiting on the pointer variable gets a signal. It then is awakened by the signal-handling code in the scheduler. This tasks will then awaken *all* the tasks waiting on the pointer variable, just like wake_up() would. (If the task is at the top of the "wait stack", the behavior is just like a call to wake_up(). If the task is somewhere else in the wait stack, it wakes up the task at the top, puts itself back to sleep, and waits to be awakened. Then things proceed as with wake_up().) Of course, you have to do things this way with such an implementation, since there's no practical way of unlinking a task from the middle of such a wait stack. But the behavior still seems odd.

Maybe I'm mistaken about the intended behavior of sleep_on_interruptible(), but I thought a signal was only supposed to wake the receiving task, not all the tasks. Am I wrong? It certainly seems to work this way in versions 1.0 and later, which used a conventional wait queue approach.

Mind you, the old kernel code still works, because tasks which are awakened always seem to recheck the condition they are waiting for before proceeding; if it hasn't occurred, they go back to sleep again. But it seems inefficient to wake all the tasks simply because one got a signal, when they will all just call sleep_on_interruptible() again.

Comments? Is this a valid criticism and a reason the kernel was changed, or am I just confused about sleep_on()? And, assuming the criticism is valid, why did they wait until version 1.0 to make the change?

P.S. Is there any archive or record of early discussions about the kernel design? The oldest thing I can find is an archive of the linux-kernel mailing list which only dates back to the summer of 1995, four years after the project started.

Server crashes using 2.0.32 and SMP

Forum: The Linux Kernel Hackers' Guide
Keywords: SMP Crash Help!
Date: Tue, 03 Feb 1998 18:31:59 GMT
From: Steve Resnick <steve@ducksfeet.com>

Greetings,

I have been having some problems with server crashes. On two occasions I was able to have personnel at the co-location facility, where my server lives, look at the console immediately after a crash.

The kernel version running was 2.0.32 w/ SMP support on a dual Pentium Pro box.

When the server would crash, a message would be continuously displayed on the console (but not in the syslog):

Aiee: scheduling in interrupt: 0012BBD1

A search of the sources found that this condition was tested for in /usr/src/linux/sched.c on line 396 and the message printed on line 497.

It would appear that an interrupt was encountered during the schedule() operation. This would be a bad thing. (It's not nice to re-enter the scheduler via an interrupt)

Since the address being printed is, presumably, the return address after the schedule call, and is consistent, I am assuming that the scheduler is being re-entered while servicing some sort of interrupt from within the same ISR.

First, are my assumptions even close to reality?

Secondly, is this a "known" issue with the 2.0.32 kernel. I understand there have been some changes in the kernel SMP code between 2.0.32 and 2.0.33 so I am wondering if upgrading the kernel will fix this.

Thirdly, does this indicate some sort of hardware failure and if so, how can I trace this back to the device in question.

Finally, I am open to suggestions for other ideas and/or options here.

As always, any help is appreciated. Most suggestions taken seriously :)

Thanks, in advace,

Steve

Messages

1. Debugging server crash *by Balaji Srinivasan*

Debugging server crash

Forum: The Linux Kernel Hackers' Guide
Re: Server crashes using 2.0.32 and SMP (Steve Resnick)
Keywords: SMP Crash Help!
Date: Tue, 03 Feb 1998 19:21:37 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

To get more information on where exactly schedule was called within the interrupt handler, compile the kernel with the -g flag (also remove the -fomit-frame-pointer flag). These options can be set via the CFLAGS definition in the main Makefile of the kernel.

The address that is printed seems to be bogus. Recompile the kernel with debugging enabled (-g) and see what address it prints out. You can then check out what function the address actually is in by using gdb on the vmlinux file.

Also you might just try upgrading to 2.0.33

Hope this helps balaji

More Information

Forum: The Linux Kernel Hackers' Guide
Re: Server crashes using 2.0.32 and SMP (Steve Resnick)
Re: Debugging server crash (Balaji Srinivasan)
Keywords: SMP Crash Help!
Date: Thu, 05 Feb 1998 19:22:49 GMT
From: Steve Resnick <steve@ducksfeet.com>

Ok ... A little more detail here:

I moved to 2.0.33, and build a kernel with SMP support, removing the -fomit_frame_pointer flag and added -g.

My server crashed this morning with the same error message, the address being somewhat different, but close enough to the original that I suspect the address to be the same function, in relation to 2.0.32

The scheduler was re-entered (apparently) from __wait_on_buffer in /usr/src/linux/fs/buffer.c, line 121.

I also read elswhere that enhanced RTC support is needed for the kernel, which had not been added before.

Could this be a cause of the crashes?

Cheers,

Steve

it should not have happenned...

Forum: The Linux Kernel Hackers' Guide
Re: Server crashes using 2.0.32 and SMP (Steve Resnick)
Re: Debugging server crash (Balaji Srinivasan)
Re: More Information (Steve Resnick)
Keywords: SMP Crash Help!
Date: Sun, 08 Feb 1998 23:33:44 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

As far as i know, wait_on_buffer should not be called from within the interrupt context. (since it explicitly calls schedule). Along with the crash, there should be a call trace that gets printed out. With that you might be able to find out what exactly called __wait_on_buffer

```
Hope this helps.
balaji
PS: I doubt if RTC has anything to do with the crash...
        (maybe im wrong)
```

Broken URL: http://www.redhat.com:8080/HyperNews/get/khg/208/1/1/www.ducksfeet.com/steve

Try: http://www.redhat.com:8080/HyperNews/get/khg/208/1/1.html

Signals ID definitions

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 02 Feb 1998 13:54:56 GMT
From: Franky <genie@risq.belcaf.minsk.by>

I found recently that signal IDs, defined in src/include/asm-i386/signal.h are confused:

```
#define SIGABRT   6
#define SIGIOT      6
```

is that the right approach?

the segment D000 is not visible

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 02 Feb 1998 13:00:01 GMT
From: <martinv2@ctima.uma.es>

Hi all

```
   I am working on the design of ISA board for a PC. It has
96 Kb of ROM with addresses between C800:0 and D000:FFFF. The board works ok in a
pentium 75 but it does not work in more recent systems.
   The reason is that the segment D000 is not visible.
   Does anibody know what is going on and how to solve it ?
```

Thanks in advance.

Manuel J. Martin Universidad de Málaga.

ICMP - Supressing Messages in 2.1.82 Kernel

Forum: The Linux Kernel Hackers' Guide
Keywords: ICMP request IPV4 broadcast
Date: Fri, 30 Jan 1998 20:26:25 GMT
From: Brent Johnson <brent@foxtrot.dyn.ml.org>

On our local network we are using a terminal server which uses something called "Win-S" - it intentionally sends an ICMP (ping) request to the broadcast address - because thats what its supposed to do (i dont know the details of why).

Because of this I keep getting a message like: ipv4: (1 messages suppresed. Flood?) xxx.xxx.xxx.xxx send an invalid ICMP error to a broadcast.

How can I stop this error message from being displayed?

Messages

1. Change /etc/syslog.conf *by Balaji Srinivasan*

Change /etc/syslog.conf

Forum: The Linux Kernel Hackers' Guide
Re: ICMP - Supressing Messages in 2.1.82 Kernel (Brent Johnson)
Keywords: ICMP request IPV4 broadcast
Date: Sat, 31 Jan 1998 00:56:29 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

The kernel message is printed with KERN_CRIT level. so modify your /etc/syslog.conf to log KERN_CRIT messages to /var/adm/kernel or some other file. Note that in that way you will potentially loose a lot of important messages...

If you want to fix this particular message then just modify your kernel (file net/ipv4/icmp.c line 683) to printk(KERN_INFO "%s sent an invalid ICMP error to...); Dont try this if you dont want to mess around with the kernel. balaji

Modem bits

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 30 Jan 1998 09:32:12 GMT
From: Franky <Franky>

Hello, all:

In my current project I should obtain the current status of a modem - CD, CTS, DTR, etc. Could anyone give me a hint of how to do that?

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: Modem bits (Franky)
Date: Sat, 31 Jan 1998 18:57:59 GMT
From: Kostya <ovsov@bveb.belpak.minsk.by>

It's quite simple. I think the code will explain it the best.

/****************************/

#include <termios.h>

int status ;

```
ioctl (fd,TIOCMGET,&status) ; // fd - already opened device.
if (status & TIOCM_DTR)
       printf ("DTR is asserted\n"); // for example
```

/******************************/

Info about '#defines' for modem signals in /usr/src/linux/include/asm/termios.h

Best regards.

I need some way to measure the time a process spend in READY QUEUE

Forum: The Linux Kernel Hackers' Guide
Keywords: kernel, scheduler, ready queue
Date: Tue, 27 Jan 1998 17:00:46 GMT
From: Leandro Gelasi <gelaslean@sunto.ing.unisi.it>

```
Hi everybody!

I need some way to get an accurate measurement of  the time a
process spend in the scheduler's "ready queue" during his life

Any help will be appreciated.

Leandro Gelasi
```

How to make sockets work whilst my process is in kernel mode?

Forum: The Linux Kernel Hackers' Guide
Keywords: socket, kernel
Date: Fri, 23 Jan 1998 14:10:05 GMT
From: Mikhail Kourinny <misio@kurin.kharkov.ua>

Hi all, I've written a kernel module that communicates with a remote device via TCP. It works fine while data is passed at less than ~4K per write. Otherwise I receive a "Broken pipe" mistake :(Looks like kernel internal buffers are not being empied while I'm in kernel mode. I tried setsockopt(TCP_NODELAY), sleeping { current->state = INTERRUPTIBLE; current->timeout = jiffies + 10; schedule();} with no success.

GODS! Where is my mistake? How should I make sockets to work? I'd be happy being pointed to some TFM :)

There should be probably less tricky ways to solve my problem, but I just wish to complete this approach.

Thanks in advance,

Mikhail

Realtime Problem

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 14 Jan 1998 20:51:59 GMT
From: Uwe Gaethke <Uwe.Gaethke@t-online.de>

Hi!

I do have a problem with the realtime capabilities of Linux. I wrote a loadable driver which uses the RealTimeClock to generate periodic interrupt at a frequency of 1024Hz. With each interrupt the driver increments an internal counter. The 'read' function of the driver returns simply that counter.

Next I wrote a realtime task (with the highest priority) which read that RTC continuously. This tasks checks if the read counter is incremented by one between two read calls. If not, it prints an error message which includes also the time between the last two read calls.

What I expect is: 1. The task calls 'read', 2. The RTC generates an Interrupt, 3. The task returns from 'read'.

If this is done fast enough (less than 1ms) every interrupt will get through to the realtime task.

And here is my surprise: Everything worked as expected until I called 'free' or 'ps' from a different shell. At this time the task seem to loose interrupts for almost exact 10ms (i.e. 10 to 11 interrupts). It seems that the scheduling is blocked for one tick.

Does anybody know why this happens?

```
Thanks for your help,
  Uwe
```

Messages

1. SCHED_FIFO scheduling *by Balaji Srinivasan* NEW

SCHED_FIFO scheduling

Forum: The Linux Kernel Hackers' Guide
Re: Realtime Problem (Uwe Gaethke)
Date: Thu, 15 Jan 1998 00:41:30 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

I have a question regarding your initial query.

In your mail you said that the expected sequence of events is: 1: The task calls 'read' 2: RTC generates an interrupt 3: The task returns from read.

I dont understand why the task should wait till the rtc generates an interrupt. Am i missing something here?

As far as your query goes: What might be happening is that ps/free might be waking up some other SCHED_FIFO (which i guess is what you are using) scheduled process (some kernel threads are scheduled using SCHED_FIFO). This might schedule that process in instead of yours.

If you need predictable performance then you might try using KURT: KU Real-Time Linux (http://hegel.ittc.ukans.edu/projects/kurt)

I hope this helps balaji

inodes

Forum: The Linux Kernel Hackers' Guide
Keywords: inodes' locks
Date: Tue, 13 Jan 1998 22:49:14 GMT
From: Ovsov <ovsov@bveb.belpak.minsk.by>

What if after wait_on_inode () but before inode->lock = 1 in the inode.c module some hardware interrupt comes and any other process will be scheduled that wants to use the same inode ???

Difference between SOCK_RAW SOCK_PACKET

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 07 Jan 1998 14:23:15 GMT
From: Chris Leung <eg_lch@stu.ust.hk>

```
We are doing a project that required us to change some
chatacteristics of the TCP protocols and/or add a buffer
into the TCP layer also. It seems to us that the SOCK_RAW or
SOCK_PACKET are good choice to override the TCP or even
IP level of the networking software.
Unfortunately, we cannot find any documents or sample programs
about SOCK_RAW or SOCK_PACKET... :~(
Please help us out ~~~~~~
```

SOCK_PACKET

Forum: The Linux Kernel Hackers' Guide
Re: Difference between SOCK_RAW SOCK_PACKET (Chris Leung)
Keywords: SOCK_PACKET
Date: Wed, 10 Jun 1998 18:01:01 GMT
From: Eddie Leung <edleung@uclink4.berkeley.edu>
Body-URL: http://www.senie.com/dan/technology/sock_packet.html

f78

Using the SOCK_PACKET mechanism in Linux To Gain Complete Control of an Ethernet Interface

Daniel Senie
Amaranth Networks, Inc.

I have put together this web page in response to many queries from multiple people. Rather than continue to write individual responses, I have put together this page to explain what I was trying to do, and how I got it to work.

First, some background. To simulate software that was intended to run on a different (and not yet built) platform, I needed a convenient way to exercise the code against live networks. I first tried using a Solaris system, using the DLPI driver. This allowed me to do most things, but failed when I needed to be able to set the source Ethernet MAC address. The Solaris DLPI driver provides no way to override the hardware on a per-packet basis.

Next, I started looking at mechanisms in Linux. The mechanism that seemed to fit the best was SOCK_PACKET, which is used by tcpdump among other things. To Make this work for me, though, it was necessary to keep the Linux machine from doing anything on the interface, other than letting my programs at it.

How To Do It

This information and these instructions work for RedHat Linux 4.2 with a 2.0.30 kernel. I expect they'll work fine on a 2.0.32 kernel as well, and with other Linux distributions. I have heard that a better mechanism for providing this facility is coming in a newer kernel. If or when I get more information on that, I'll see about adding another page on that.

First, the interface needs to be told NOT to run ARP. Promiscuous mode should be enabled if you need to hear everything on the wire.:

```
        ifconfig eth1 -ARP PROMISC UP 10.1.1.1
```

Then tell the Linux stack it's not supposed to see any of the traffic to or from this port:

```
        ipfwadm -O -a deny -P all -S 0/0 -D 0/0 -W eth1
        ipfwadm -I -a deny -P all -S 0/0 -D 0/0 -W eth1
```

In the program, you need to do several things. First, the socket call:

```
        s = socket(AF_INET, SOCK_PACKET, htons(0x0003));
```

to get the socket set up.

Next I bind the specific Ethernet NIC I want:

```
        struct sockaddr myaddr;

        memset(&myaddr, '\0', sizeof(myaddr));
        myaddr.sa_family = AF_INET;
        strcpy(myaddr.sa_data, "eth1");  /* or whatever device */

        r = bind(s, &myaddr, sizeof(struct sockaddr));
```

and check the return code for any errors.

Now, when you want to send or receive, this socket is bound to the proper device. One word of caution, though, ALWAYS check the received packets to be sure you got them on the right device. There's a race condition between making the socket call and the bind call where you'll get all packets from ALL interfaces... not what you want!

So, to send a packet:

```
        struct sockaddr from;
        int fromlen;

        memset(&from, '\0', sizeof(from));
        from.sa_family = AF_INET;
        strcpy(from.sa_data, "eth1"); /* or whatever device */
        fromlen = sizeof(from);

        r = sendto(s, msg, msglen, 0, &from, fromlen);
```

and check the return code. Note that msg is the pointer to the packet, starting with the MAC header. Be sure you put the proper source MAC address into your packets! Also, msglen is the length of the packet including the MAC header, but not including the CRC (which I do not worry about, but the hardware does supply).

Receive is pretty similar:

```
        struct sockaddr from;
        int fromlen;

        fromlen = sizeof(from);

        r = recvfrom(s, msg, 2048, 0, &from, &fromlen);
        if (r == -1)
        {
                /* deal with error */
        }
        if
5fc
(strcmp(from.sa_data, "eth1") != 0)
        {
                /* not from the interface we wanted, discard */
        }
```

if r == -1, you have an error. If r > 0, then r is the length of the received packet. The strcmp ensures the packet came from the right interface.

If you want to receive for MAC addresses other than the one the board has in it, use promiscuous mode. To get the mac address from your program, there's an ioctl call SIOCGIFHWADDR. In the return from that call is also the hardware type, so you can ensure it's Ethernet. Another call, SIOCGIFMTU will tell you the MTU of the interface.

Caveats

- **Do not** use this methodology on your primary Ethernet interface. Instead, install a second (and if needed, third) NIC card for use in this way. I've successfully used 5 NIC cards in one machine, 1 under the control of Linux, the rest bypassed to my programs.
- **Be VERY sure** you set up the **ipfwadm** commands. Failure to do so will make a huge mess, likely causing networking problems for other hosts on your lan.

If you found this information helpful and useful, please let me know. If you require further information or assistance in this area, this can be arranged. For consultation beyond simple questions, Amaranth Networks, Inc. can provide advice, services and information for a fee.

0

Need additional termcap entries for TERM=linux

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 05 Jan 1998 07:47:29 GMT
From: Karl Bullock <karl@dixie-net.com>

I have an application that needs the graphics characters for the default G1 character set, but I can't find the escape sequences for the G1 set anywhere. Anyone know where I can find the full G1 specification?

Question on Umount or sys_umount

Forum: The Linux Kernel Hackers' Guide
Keywords: umount sys_umount do_umount
Date: Wed, 31 Dec 1997 14:40:07 GMT
From: teddy <teddy@gti.net>

```
the program umount.c unmounts devices.
This program calls a function called umount(...)
this function calls the kernel function do_umount
located in super.c in kernel.
=================================
I can not find the function umount!
Where is it?

FYI:  There is a kernel function called sys_umount,
the comments in this section say it is "umount"!

If this is true how does sys_umount get associated with umount?
===============================
email = teddy@gti.net
```

Passing file descriptors to the kernel

Forum: The Linux Kernel Hackers' Guide
Keywords: file descriptor userfs pipe module
Date: Tue, 30 Dec 1997 18:54:12 GMT
From: Pradeep Gore <pradeepg@corelcomputer.com>

I am porting a filesystem(the userfs0.9.4.2) source from the linux x86 version to another for the arm processor. The userfs filesystem has a kernel module that accepts a pair of file descriptors passed to it in a user defined structure in the mount function. The kernel module receives this structure in the read_super function of the file_system_type struct.(defined in fs.h).

now the problem.. The file descriptors passed are invalid, the values are correct but when the kernel module code tries to convert the file descriptor to a file pointer,it fails.

```
struct file* fdtofp(int fd)
{
  struct file* fp;
  fp = current->files->fd[fd];
  if (fp->f_inode == NULL)
  {
     return NULL; // fp->f_inode turns out to be null and the
                  // function returns here
  }
}
```

This code works perfectly on the x86 linux but not on the arm version.The descriptors are created successfully using the pipe function.

can anyone explain what could be going wrong?How can i debug? Is there a way to convert a file descriptor to a struct file* in a user process?

```
thanks for any help,
Pradeep Gore
```

A way to "transform" a file descriptor into a struct file* in a user process

Forum: The Linux Kernel Hackers' Guide
Re: Passing file descriptors to the kernel (Pradeep Gore)
Keywords: file descriptor userfs pipe module
Date: Wed, 14 Jan 1998 23:07:42 GMT
From: Lorenzo Cavallaro <lc529863@silab.dsi.unimi.it>

```
Well, I don't know if I can help you and
maybe this'll be a stupid answer, but
about finding a way to transform a file descriptor into
a struct file*, try:

struct file *fdopen(int filedes, char *mode)

Where filedes is your file descriptor (Geeeee :) )
and mode can be either "w" or "r" (write/read mode)

This functin returns upon successfull a struct file*, else
a NULL pointer;

Let you also look on the man page;
This should work in a user process.
```

? Dead Man Timer

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 23 Dec 1997 16:39:31 GMT
From: Jody Winston <jody@sccsi.com>

I have a device driver that needs a dead man timer to go off and invoke a function if and only if the time since the last interrupt is greater than a given value. Should I use interruptible_sleep_on to implement the dead man timer?

raw sockets

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 11 Dec 1997 05:02:52 GMT
From: lightman <lightman2@hotmail.com>

```
I'm using Raw sockets and have sent away a TCP packet with the SYN flag on. I get the
SYN|ACK response and responds to it. But the kernel beats me to it, and sends away a
RST packet before my ACK response to the SYN|ACK. How can you stop the kernel from
responding with a RST to the SYN|ACK? I use two seperate sockets, one for
transmitting
and one for receiving.
```

a kernel-hacking newbie

Forum: The Linux Kernel Hackers' Guide
Keywords: Thinkpad IBM Mwave newbie
Date: Thu, 11 Dec 1997 04:52:17 GMT
From: Bradley Lawrence <cecil@niagara.com>

I've never hacked the Linux kernel before, and I'm not very experienced in C++ at all, but I am a man on a mission.

I've got a Thinkpad 760E, and the modem simply will not work in Linux. It's a nasty little peice of hardware, IBM's 'Mwave' soundcard/modem combo... and I've finally gotten tired of waiting for someone to come out with support for it, so I'm going to try and do it myself... I probably won't get anywhere, but I'm going to try.

The only source of information I have right now is the the Win95 driver, and since I don't know assembly that's not very informative at all. I'm trying to get IBM to give me some kind of specs for it, but so far I've gotten nowhere.

The more I think about it the more I realize I'm never going to get anywhere. But I'm desperate. Linux without a modem is rather pointless for me, and after spending $3,999 on this computer, I don't have the change lying around to buy an external modem...

So anyway, my question is ... where do I start? I know bits and peices of C++, and have never played with the kernel code before. Thanks a lot. Sorry, but I really honestly have no clue how to learn about this subject.

A place to start.

Forum: The Linux Kernel Hackers' Guide
Re: a kernel-hacking newbie (Bradley Lawrence)
Keywords: starting newbie
Date: Fri, 20 Mar 1998 22:40:03 GMT
From: *<unknown>*

A good place to start would be getting a subscription to the Linux Journal. http://www.linuxjournal.com. They have informative articles on driver programming and kernel hacking. Also you should by a book on C, not C++. All the code I've seen for the kernel is C and assembly. Also you really shouldn't have to know too much assembly to write a driver for your modem. I seriously doubt that IBM is going to give you too much info about the modem, but best of luck.

Les Thompson

Modems in general

Forum: The Linux Kernel Hackers' Guide
Re: a kernel-hacking newbie (Bradley Lawrence)
Keywords: Thinkpad IBM Mwave newbie
Date: Thu, 15 Jan 1998 09:03:11 GMT
From: Ian Carr-de Avelon <ian@emit.com.pl>

```
I don't know this modem but as an ISP I deal with lots of others.
99.9% of modems deal with the Hayes commands internally. A typical
attack on this would look like:
Fire up computer in an OS for which you have all the support
software. In this case 95.
Link to the modem with a terminal program eg Hyperterm
Type:
ATZ
See how it says:
OK
Note all the settings.

Now we need to get that is Linux.
Move to linux with Linloader. Connect to the modem with a Linux
terminal software eg kermit. Take as an example a modem which was
com2 under Win95

set line /dev/ttyS1
set speed 115200 (assuming new hardware here)
c

Now try the ATZ again. If it does not work I suggest you either
give up and swap the modem, or try to get a contract to write the
driver. It will be a big job.
If it does work you can start reading the HOWTOs about PPP get
mgetty for remote login etc.
You can try going directly to Linux with LILO. If that does not
work, but after Win95 start did work, the driver does some kind of
PnP initialisation and you had better keep Win95

Ian
```

How to write CD-ROM Driver ? Any Source Code ?

Forum: The Linux Kernel Hackers' Guide
Keywords: CD-ROM Device Driver Source code
Date: Mon, 08 Dec 1997 06:53:49 GMT
From: Madhura Upadhya <mupadhya@in.oracle.com>

Please any one can specify how to write CD-ROM Driver on linux. Is there any ready made reference source code available ?. How exactly it works ?

Measuring the scheduler overhead

Forum: The Linux Kernel Hackers' Guide
Keywords: scheduler overhead
Date: Thu, 04 Dec 1997 22:34:31 GMT
From: Jasleen Kaur <jks@cs.utexas.edu>

```
Hi,

I am implementing a different scheduling policy in the
scheduler of Linux 2.0.0.

I need to measure the time taken for choosing the next
task to be scheduled, i.e., the actual time spent in the
schedule() function, without the actual context switch
having taken place. Since I clear interrupts in schedule(),
how do I measure the time? Does 'jiffies' still contain the
actual time? Or is the time asynchronously updated in some
other variable while the timer interrupt is disabled?

Thanks,

Jasleen Kaur
```

Where can I find the tcpdump or snoop in linux?

Forum: The Linux Kernel Hackers' Guide
Keywords: tcpdump snoop
Date: Thu, 04 Dec 1997 02:19:11 GMT
From: <wangc@taurus>

Hi ,in some other Unix systems ,I can find very useful tools such as tcpdump and snoop , but in Linux , how can I get them?

man which

Forum: The Linux Kernel Hackers' Guide
Re: Where can I find the tcpdump or snoop in linux?
Keywords: tcpdump snoop
Date: Mon, 25 May 1998 00:36:31 GMT
From: <trajek@j00nix.org>

[root@localhost /root]# which tcpdump /usr/sbin/tcpdump [root@localhost /root]#

Timers don't work??

Forum: The Linux Kernel Hackers' Guide
Keywords: timer
Date: Tue, 02 Dec 1997 21:18:09 GMT
From: Joshua Liew <jliew@pacific.net.sg>

I've been playing around with the add_timer and del_timer functions and can't seem to get it to work. Say if I want to execute a IRQ routine using a 10sec timer for 10 times, only the first time there is a 10 sec delay, but subsiquent interrupts are simultaneous.

static struct timer_list timer = { NULL, NULL, INTERVAL, 0L, &irq };

main() { add_timer(&timer); }

irq() { del_timer(&timer); timer.expires = INTERVAL; add_timer(&timer); printk(KERN_DEBUG "Hello."); }

Is there any documentation on timers? Appreciate all the help I can get. I have a project that needs this and I'm stuck.

Thanks.

Messages

1. Timers Work... *by Balaji Srinivasan* NEW

Timers Work...

Forum: The Linux Kernel Hackers' Guide
Re: Timers don't work?? (Joshua Liew)
Keywords: timer
Date: Wed, 03 Dec 1997 01:25:06 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

```
The mistake with your code is that you need to
update INTERVAL every time you add_timer.
The expires field in timer_list is an absolute
time not a relative time.

for example:

irq() {
    timer.expires = jiffies + INTERVAL;
    add_timer(&timer);
}
```

problem of Linux's bridge code

Forum: The Linux Kernel Hackers' Guide
Keywords: bridge code IEEE 802.1d
Date: Tue, 02 Dec 1997 09:17:55 GMT
From: <wangc@taurus>

```
In the Linux's bridge source code ,it refers to the IEEE 802.1d specification section
4.9.1. where can I get this documentation?
        thanks!
```

Documention on writing kernel modules

Forum: The Linux Kernel Hackers' Guide

Keywords: modules insmod
Date: Mon, 01 Dec 1997 20:22:08 GMT

From: Erik Nygren <nygren@mit.edu>

As far as I can tell, this site doesn't yet have information on how to write a loadable kernel module (although there are quite a few queries from people asking about how to do it). After looking around, I found that the insmod/modules.doc and insmod/HOWTO-modularize files in the modules-2.0.0.tar.gz package contained a fairly good description of some of the things you need to do when writing a kernel module. The insmod/drv_hello.c and insmod/Makefile files in that package provide an example character device driver that can be built as a module.

It would be nice if these files (or the relevant contents) could get incorporated into the KHG at some point.

To summarize, it looks like modules should be built with the following compiler options (at least, this is the way the Makefile for drv_hello.o goes):

```
-O6 -pipe -fomit-frame-pointer -Wall -DMODULE -D__KERNEL__ -DLINUX
```

Of the above, it seems likely that the key arguments are the `-DMODULE -D__KERNEL__` and possibly the `-O6` are actually needed. If you want to build your module with versioning support, add the following options:

```
-DMODVERSIONS -include /usr/include/linux/modversions.h
```

From the examples and docs, it looks like modules should be in the form:

```
#include   /* must be first! */
#include   /* optional? */
#include  /* ? */


/* More includes and the body of the driver go here. */
/* Note that any time a function returns a resource to the kernel
 * (for example, after a open),
 * call the MOD_INC_USE_COUNT; macro.  Whenever the
 * kernel releases the resource, call MOD_DEC_USE_COUNT;.
 * This prevents the module from getting removed
 * while other parts of the kernel still have
 * references to its resources.
 */


#ifdef MODULE  /* section containing module-specific code */

int
init_module(void)
{
        /* Module initialization code.
         * Registers drivers, symbols, handlers, etc. */
        return 0; /* on success */
}

void
cleanup_module(void)
{
        /* Do cleanup and unregister anything that was
         * registered in init_module. */
}

#endif
```

Again, see the documentation scattered through the modules-2.0.0 package (and also presumably through the newer modutils-2.1.x packages) for more detailed information.

How to display a clock on my console?

Forum: The Linux Kernel Hackers' Guide
Keywords: memory clock
Date: Fri, 28 Nov 1997 09:43:27 GMT
From: keco <hu@billow.ml.org>

```
Hi:

   i want to display a clock on my console under text mode,
Can i access the video memory directly?

   i try many method,but failed,how should i do?
thank you every much!

   ##I cant often read "kernel hack guide" :( ,please mail
to me. eamil: hu@billow.ml.org
```

Difference between SCO and Linux drivers.

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 27 Nov 1997 09:42:04 GMT
From: M COTE <ndg@mcii.fr>

```
I've a driver that was developped under SCO Unix and i'me wonderring if it is easy to
port under Linux.

Can somebody help me ???.

Thanks ...

       M COTE (ndg@mcii.fr)
```

Changing the scheduler from round robin to shortest job first for kernel 2.0 and up

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 26 Nov 1997 10:39:26 GMT
From: <royal_and_mary_harrell@msn.com>

How do you recode the kernel from round robin to shortest job first in later versions of Linux. I'm using Red Hat 4.2.? I need this for a operating systems class and for better understanding of Linux.

Improving the Scheduer

Forum: The Linux Kernel Hackers' Guide
Re: Changing the scheduler from round robin to shortest job first for kernel 2.0 and up
Keywords: scheduler linux
Date: Fri, 06 Mar 1998 10:23:30 GMT
From: Lee Ingram <lee@ingram.clara.net>

How could I improve the scheduler. Please HELP

Messages

1. Improving the Scheduler : use QNX-like *by Leandro Gelasi* NEW

Improving the Scheduler : use QNX-like

Forum: The Linux Kernel Hackers' Guide
Re: Changing the scheduler from round robin to shortest job first for kernel 2.0 and up
Re: Improving the Scheduer (Lee Ingram)
Keywords: scheduler linux QNX
Date: Fri, 06 Mar 1998 15:13:41 GMT
From: Leandro Gelasi <gelaslean@sunto.ing.unisi.it>

HI!

There are some Linux modified scheduler on Internet.

The best one for general purpouse application is QNX-like scheduler patch from Adam McKnee (amckee@poboxes.com). It is a patch for kernel 2.0.32.

I am testing the performance of this scheduler for a University exam and the preliminary results says it is better than standard one .

It would grant good interactive performance under heavy CPU load.

You can find the patch at www.linuxhq.com or by contacting the autor.

Hope this helps

Leandro

Re: Changing the sched. from round robin to shortest job first for kernel 2.0 and up.

Forum: The Linux Kernel Hackers' Guide
Re: Changing the scheduler from round robin to shortest job first for kernel 2.0 and up
Date: Mon, 05 Jan 1998 03:53:12 GMT
From: Pirasenna V.T. <piras@pspl.co.in>

```
Hello,
You must have got a lot of replies, but I am just sending
my views about it.  It get ur idea that you wanna better
understanding of the OS.  But I really wonder whether the
system will behave sane if you do the changes.  You have
not told abt ur M/c.  If it is a IBM compatible, then
you should go to the source code and try to grep on the
kernel data structures that you know, like, GDT, LDT, etc,.

Pl. tell me if you get some results, thanx,
Piras
piras@pspl.co.in
```

meanings of file->private_data

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 24 Nov 1997 11:09:40 GMT
From: <ncuandre@ms14.hinet.net>

Could anyone tell me what's the meaning of private_data in the structure file (file->private_data), and under what condition should I use it.

/dev/signalprocess

Forum: The Linux Kernel Hackers' Guide
Keywords: signal process device
Date: Mon, 24 Nov 1997 00:19:50 GMT
From: <flatmax>

Has anyone started on /dev/signalprocess ?

how to track VM page access sequence?

Forum: The Linux Kernel Hackers' Guide
Keywords: Virtual Memory, page access
Date: Sat, 22 Nov 1997 02:26:54 GMT
From: shawn <shawnc@cs.berkeley.edu>

Does anyone know how one can keep track of VM page access sequence numbers of some application programs? One way I thought was to mark each virtual page as protected at allocation time, so an access to such a page will result a page fault, which is much easier to record. However, I couldn't find any kernel functions that will lock only one virtual page. The functions I found were just marking an entire virtual memory area as not readable, not writable, etc. I am trying to create some user application program page access profiles. Any hints are greatly appreciated.

Shawn

Whats the difference between dev_tint(dev) and mark_bh(NET_BH)?

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 21 Nov 1997 14:39:17 GMT
From: Jaspreet Singh <jaspreet@sangoma.com>

Hi,

I don't know what the difference between the two is. It looks as if "net_bh()" calls devtint (and i guess doing mark_bh(NET_BH) calls net_bh()) .

My driver calls dev_tint again and again due to heavy traffic and as a result the kernel crashes with the error: "release:kernel stack corruption".

```
 However when I replace that call with a mark_bh(NET_BH)  then my kernel doesn't
crash.
```

If someone could shed some light as to what the difference between the two is that would be great.

Thanks

Jaspreet Singh

PCI

Forum: The Linux Kernel Hackers' Guide
Keywords: PCI fault
Date: Wed, 19 Nov 1997 21:43:04 GMT
From: <mullerc@iname.com>

i try to write a device driver (module) for a card on the pci bus but i get segmentation fault in my x86 kernel (2.0.30) when i try to access the pci bus (memory) at 0xE4102000 for example. why?

Messages

1. RE: PCI *by Armin A. Arbinger* NEW

RE: PCI

Forum: The Linux Kernel Hackers' Guide
Re: PCI
Keywords: PCI fault
Date: Thu, 20 Nov 1997 03:43:44 GMT
From: Armin A. Arbinger <armin.arbinger@bonn.netsurf.de>

did you make an ioperm() on the memory you want to access?

Can I make syscall from inside a kernel module?

Forum: The Linux Kernel Hackers' Guide
Keywords: syscall, module, lock physical page
Date: Tue, 18 Nov 1997 21:48:35 GMT
From: Shawn Chang <shawnc@cs.berkeley.edu>

I have a silly question about syscalls. I wrote a kernel module for user controlled page allocation, which needs to call some kernel function to lock some physical pages allocated, so they won't get swapped out. After searching in the kernel source, I only found sys_mlock() in mm/mlock.c seem to be a good function for my purpose. But sys_mlock() is not a directly exported kernel symbol, so my module can't call it directly. Then I found that one of the pre-defined syscalls is actually mlock, so I was thinking if I could make a syscall from inside my kernel module. Is it possible to do that? Otherwise, how do I export sys_mlock() so my kernel module will be able to call it?

Another related question. My kernel module is accessed by user application through syscall(164,...). Suppose I want to access some functions in the kernel module from some original kernel functions, such as do_page_faults() in fault.c. What should I do?

Thanks in advance for any answers.

Shawn

Re: Can I make syscall from inside a kernel module?

Forum: The Linux Kernel Hackers' Guide
Re: Can I make syscall from inside a kernel module? (Shawn Chang)
Keywords: syscall, module, lock physical page
Date: Sat, 17 Jan 1998 07:39:25 GMT
From: Massoud Asgharifard <asghari@ce.sharif.ac.ir>

hi, Well, no. when you are doing a syscall from user space, the syscall parameters are in segment register fs (assuming x86). The functions memcpy_fromfs and memcpy_tofs are called in kernel to retrieve the parameters for kernel function. but module code is kernel code anyway, and can't do that. (Segmentation fault....) (try rewriting the syscall, but reference kernel memory for parameters.) If you want to call kernel-internal-functions from your module code, (which is not exported normally) you should register it into file linux/kernel/ksyms.c this will export that function's name and insmod will install your module. Sorry for typos and mistakes, if any.

Make a syscall despite of wrong fs!!

Forum: The Linux Kernel Hackers' Guide
Re: Can I make syscall from inside a kernel module? (Shawn Chang)
Re: Re: Can I make syscall from inside a kernel module? (Massoud Asgharifard)
Keywords: syscall, module, fs
Date: Fri, 23 Jan 1998 12:00:16 GMT
From: Mikhail Kourinny <misio@kurin.kharkov.ua>

Yes, despite of fs pointing to wrong place. There are functions somewhere in the kernel named put_fs and get_fs. Using these functions you should place KERNEL_DS in fs. And after syscall restore it, certainly. You can avoid it if you don't pass user space pointers a paramters, IMHO.

Regards, Mikhail

code snip to make a sys_* call from a module

Forum: The Linux Kernel Hackers' Guide
Re: Can I make syscall from inside a kernel module? (Shawn Chang)
Keywords: syscall, module, lock physical page
Date: Thu, 08 Jan 1998 22:35:21 GMT
From: Pradeep Gore <pradeepg@corelcomputer.com>

```
As an example, here is a code snip that make a sys_read call from
a kernel module.

// module.c
....
#include <sys/syscall.h>
extern long sys_call_table[]; // arch/i386/kernel/entry.S

int (*sys_read)(unsigned int fd, char *buf, int count);
// pointer to sys_read. linux/fs/read_write.c

....
int init_module(void)
{
...
 sys_read = sys_call_table[SYS_read];
 //now you can use sys_read
 sys_read(...);
...
}
```

↳ Dont use system calls within kernel...(esp sys_mlock)

Forum: The Linux Kernel Hackers' Guide
Re: Can I make syscall from inside a kernel module? (Shawn Chang)
Keywords: syscall, module, lock physical page
Date: Wed, 26 Nov 1997 13:16:17 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

If you want to use a system call within a kernel module then export the system call using EXPORT_SYMBOL macro.

A better solution would be to use mlock in the user space before entering the kernel (ie. write a wrapper function for your entry point that would lock pages in for you before it enters the kernel) This in my opinion is a cleaner solution than exporting sys_mlock.

In addition since sys_mlock acts on the current process it might not have desirable effects in certain cases. Hope this helps balaji

Untitled

Forum: The Linux Kernel Hackers' Guide
Keywords: fake source IP address
Date: Tue, 18 Nov 1997 19:51:23 GMT
From: Steve Durst <sdurst@rl.af.mil>

This is a follow-up to a question in June, about how to "cheat" and change the outgoing IP source address.

I'm trying to do that too, but I only want to change packets belonging to particular user-level processes (e.g. telnet). So I'm going to set up a table that both the kernel and a user-side daemon can write to, then invoke the daemon to run whatever process I want. The daemon will get the PID and the desired fake source IP address and write it to the table.

The appropriate function (I think it's ip_build_xmit()) will read the table and change only those packets sent by the processes listed in the table. Right now I'm using printk() lines to debug this thing.

Question: HOW do you find the PID associated with a given packet? I tried current->pid but apparently it's not reliable... While some outgoing packets occur when current->pid does reflect the correct process, other times outgoing packets known to be associated with, say, telnet, occur with the current->pid indicating, say, syslogd.

Shouldn't the PID be accessible through an sk_buff? The packet had to come from somewhere, and incoming packets have to be delivered to the right processes eventually. Right?

-Steve

RAW Sockets

Forum: The Linux Kernel Hackers' Guide
Keywords: raw sockets
Date: Tue, 18 Nov 1997 19:35:46 GMT
From: Art <art@falt.deep.ru>

```
Hi All :)
  Where can I find the BIG documentstion of RAW SOCKETS?
Pls HELP.
                                           Art
```

use phy mem

Forum: The Linux Kernel Hackers' Guide
Keywords: vremap()?
Date: Wed, 12 Nov 1997 07:57:22 GMT
From: WYB <tit@public.bta.net.cn>

```
My pc has 64M memory, now I want to map the
63-64M mem into kernel, so I call:
   vremap( 63*1024*1024, 1024*1024 ) ;
1.If this block of memory has been occupied
  by others, does vremap() return fail. Otherwise
  I will crash the system.
2.After vremap(), does this block of memory marked
  alloced, and other mem request does not get any page
  of this block?
3.Are there any way to know which phy mem is free?
```

HyperNews for RH Linux ?

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 10 Nov 1997 22:00:25 GMT
From: Eigil Krogh Sorensen <eks@aar-vki.dk>

Are there RPMs with HyperNews for RH Linux ?

Not really needed

Forum: The Linux Kernel Hackers' Guide
Re: HyperNews for RH Linux ? (Eigil Krogh Sorensen)
Date: Fri, 16 Jan 1998 22:10:52 GMT
From: Cameron <cls@greems.org>

Hypernews is written entirely in Perl, and contains nothing Linux-specific. Most of the installation is configuration via forms, not appropriate to pre-packaging in an RPM or .deb file.

The one thing that threw me the first time I installed it was that all of Hypernews' data are owned by the Web server user, not by your user account, not by root. If you try to own any of it yourself it just makes a security and permissions mess. That is why you must use the setup and edit-article forms. Don't even try the command line version of setup. (Well, it might work if you `su - www` first...)

Cameron

about raw ethernet frame: how to do it ?

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 07 Nov 1997 14:27:19 GMT
From: <crbild@smc.it>

Hi, how can i send raw frames on ethernet devices ? and/or in other network devices ? thanks

Roberto Favaro

process table

Forum: The Linux Kernel Hackers' Guide
Keywords: task process
Date: Thu, 06 Nov 1997 21:15:27 GMT
From: Blaz Novak <blaz.novak@guest.arnes.si>

Hi! Could someone please tell me if it is possible for a user level program to get address of kernels task struct(process table)?

Blaz

blaz.novak@guest.arnes.si bjt@gw2.s-gimb.lj.edus.si blaz.novak@usa.net

Stream drivers

Forum: The Linux Kernel Hackers' Guide
Keywords: stream driver
Date: Thu, 06 Nov 1997 17:56:25 GMT
From: Nick Egorov <nic@item.ru>

Who knows somthing about stream drivers in LiNUX ? Maybe this is something about Net Drivers ?

Thank you ! Nick.

Streams drivers

Forum: The Linux Kernel Hackers' Guide
Re: Stream drivers (Nick Egorov)
Keywords: stream driver
Date: Fri, 13 Feb 1998 06:47:19 GMT
From: <*unknown*>

The STREAMS package for Linux is available at the site www.gcom.com

Regards,

anand.

Stream in Solaris

Forum: The Linux Kernel Hackers' Guide
Re: Stream drivers (Nick Egorov)
Keywords: stream driver
Date: Mon, 12 Jan 1998 00:33:19 GMT
From: <cai.yu@rdc.etc.ericsson.se>

```
Hi :
        I have info. about stream in Solaris .
```

Streams Programming Guide http://locutus.sas.upenn.edu:8888/ Streams protocols and drivers http://home.eznet.net/~herbert/streams_ppt_notes.htm

```
        Now I do a project about Streams , so I want a example about it ,I have no
more experience of linux . If you have some example , please forward to me .
```

Best regards

Xircom External Ethernet driver anywhere?

Forum: The Linux Kernel Hackers' Guide
Keywords: network xircom adapter driver device
Date: Mon, 03 Nov 1997 22:52:27 GMT
From: mike head <bc80267@binghamton.edu>

Are there any device drivers availible for the Xircom External Etheret adapter ee-10bu. I am looking into writing a windows driver for this adapter and need to get some technical info on it. (I'd also like to use it to hook a IPIP network to my main server, rather then buy a new internal card).

bc80267@binghamton.edu

interruptible_sleep_on() too slow!

Forum: The Linux Kernel Hackers' Guide

Keywords: device driver interrupts interruptible_sleep_on()
Date: Fri, 31 Oct 1997 06:37:26 GMT
From: Bill Blackwell <wjb@mit.edu>

```
Hi,
   I'm in the process of writing a driver for a data
acquisition board.  I'm having some difficulties setting up
the interrupt handler.  Here's the problem:
```

I first write a byte to a register on the board which initiates a data conversion (when data is ready to be read, an interrupt is generated). The next line of code is an interruptible_sleep_on() call. On some occasions, the A/D board generates an interrupt BEFORE the i_s_o() call is complete, so the task is never put to sleep and added to the queue (I hope I have that right, I'm very new to this stuff...). When the companion wake_up_interruptible() call is made at the end of the interrupt handler routine, the program stalls, since there is nothing to be awakened.

```
   Is there a fix for this?
```

Thanks for reading this - sorry if this is excruciatingly trivial.

wrong functions

Forum: The Linux Kernel Hackers' Guide
Re: interruptible_sleep_on() too slow! (Bill Blackwell)
Keywords: device driver interrupts interruptible_sleep_on()
Date: Fri, 31 Oct 1997 13:39:12 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Getting around this common race condition without disabling interrupts is one of Linux's features.

You need to add the function to the wait queue before you write to the board and cause the interrupt to occur. Look at kernel/sched.c at the definition of `interruptible_sleep_on()`: You want to do something like this:

```
current->state = TASK_INTERRUPTIBLE;
add_wait_queue(this_entry, &wait);
trigger_interrupt_from_board();
schedule();
remove_wait_queue(this_entry, &wait);
```

Linux's serial.c uses this trick.

creating a kernel relocatable module

Forum: The Linux Kernel Hackers' Guide
Keywords: kernel relocatable module
Date: Wed, 29 Oct 1997 18:41:01 GMT
From: Simon Kittle <simon@nfm.co.uk>

is there any tutorial/documentation for writing relocatable modules for the kernel or just stuff on the structure of it. I have been able to get a sort of bare bones module that does nothing loaded and unloaded (the code was just stripped down from some other device driver module) but I cant get new functions, I supose they would be syscalls to work. If I just want to add a few syscalls and not deal with any hardware, how do I "register" them so the kernel knows to use them. When I wrote one such function just to try and return one char, I wrote program to test it but could not get it linked

any hep much apprieciated.

tanks - Simon Kittle

? Up to date serial console patches

Forum: The Linux Kernel Hackers' Guide
Keywords: serial console
Date: Mon, 27 Oct 1997 01:06:54 GMT
From: Simon Green <sgreen@emunet.com.au>

I've been looking for up-to-date patches for a serial console for Linux. I know most of you will be wondering why I'm bothering... after the difficulties I've had finding any info, *I'm* starting to wonder too.

Anyway, if anyone can tell me where some info on serial consoles can be found, please let me know. The latest links I can find are about July '95, they point to non-existant pages, and they're for kernel v1.3 anyway (I currently run 2.1.56).

Kernel-Level Support for Checkpointing on Linux?

Forum: The Linux Kernel Hackers' Guide
Keywords: checkpointing, kernel
Date: Sat, 11 Oct 1997 21:57:32 GMT
From: Argenis R. Fernandez <argenis@usa.net>

```
        Has anybody heard of somebody doing this?
```

Working on it.

Forum: The Linux Kernel Hackers' Guide
Re: Kernel-Level Support for Checkpointing on Linux? (Argenis R. Fernandez)
Keywords: checkpointing, kernel
Date: Sun, 16 Nov 1997 14:40:14 GMT
From: Jan Rychter <jwr@icm.edu.pl>

I'm working on it. I should have something ready in about three weeks. Expect basic process checkpointing. Open files will be restored, network connections for obvious reasons will not. This greatly limits the use of checkpointing.

Also, I probably won't even try to do process trees nor any form of IPC stuff in the first approach. Than can be worked on later.

And BTW, I have two questions right away:

1. Do the VFS inode numbers change after boot ? (e.g. can I just store the inode info for open files?)

2. Is there any way to map inode numbers back to full path names ? (needed for migration, or if (1) is not true)

--J.

Problem creating a new system call

Forum: The Linux Kernel Hackers' Guide
Date: Sat, 11 Oct 1997 15:10:13 GMT
From: <sdesai@titan.fullerton.edu>

Thanks for your time, friend. I have a small request for you. My question is a little descriptive, so if you can, please read it completely. I would appreciate it. Thanks.

Just to learn how to generate a system call, I created a simple system call that basically sets and resets the value of a global parameter "PREFETCH".

**************** SYSTEM CALL code ********************************** int PREFETCH = 0; /* this is a global variable */

```
int initialize (int nr_hints)
{
 printk ("prefetch routine to be initialized\n");

 if (PREFETCH == 1)
        return 0;
 PREFETCH = 1;
 return 1;
}

int terminate ()
{
 PREFETCH = 0;
 return 1;
}

asmlinkage int sys_prefetch (int mode, int nr_hints)
{
 printk ("prefetch system call called\n");

 if (mode >= 0)
        return initialize (nr_hints);
 else
        return terminate ();
}
**************** SYSTEM CALL code END**********************************
```

I included this code in /usr/src/linux/fs/buffer.c I then added the following line to arch/i386/kernel/entry.S

```
     .long SYMBOL_NAME (sys_prefetch)  /* 166 */
```

```
and changed
     .space (NR_syscalls - 166)*4
to
     .space (NR_syscalls - 167)*4

I then added the following line to include/asm/unistd.h
        #define __NR_prefetch 166
```

To execute the sys_prefetch system call, I wrote a prefetch.c file with the following code.

**********************code to call sys_prefetch**************** #include <linux/unistd.h> _syscall2 (int, prefetch, int, mode, int, nr_hints)

```
void main()
{
 (few declarations and statements)
 return_value = prefetch(1, 100);   /* initialize */
 printf ("%d", return_value);
}
******************************************************************
This code compiles and runs but always returns a -1 value and does not
even print the messages on the screen that I inserted using printk() in the
system call code in buffer.c
```

Since the messages are not getting printed, I have no way to know if the system call is getting called AT ALL !!!

Thanks for reading it. If you have any insights into the problem, please let me know.

Thanks again. saurabh desai <sdesai@ecs.fullerton.edu>

How did the file /arch/i386/kernel/entry.S do its job

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Date: Sun, 01 Mar 1998 04:18:04 GMT
From: Wang Ju <wangju@envst-1.ict.ac.cn>

```
Hi,All
  I am a newer to KHG, I am sorry to ask
this trival problem.
  while adding a system call, one need to            edit the system_call_table to
add an entry,
what is it do with file entry.S and How?
Thanks
```

system call returns "Bad Address". Why?

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Date: Tue, 14 Oct 1997 20:44:22 GMT
From: <sdesai@titan.fullerton.edu>

thanks for your attention.

When I try to call my new system call, it always returns -1. I then check the error message using perror( ). The error message is always "Bad Address".

Could anybody please tell me what can cause a system call to return "Bad Address" error. All my variables are initialized and defined.

Thanks in advance for any insights from you.

saurabh desai <sdesai@ecs.fullerton.edu>

Re:return values

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Re: system call returns "Bad Address". Why?
Date: Wed, 15 Oct 1997 18:04:38 GMT
From: C.H.Gopinath <gopich@cse.iitb.ernet.in>

```
 I created the following call it is working fine, but i don't know about that bad
address.

sys call function looks like this.

int sys_print_data(char *s1,char *s2,int flag,int size1,int size2)
{
/* size1 and size2 are strlen of s1 and s2 respectively */
      if (flag) {
                 sys_write(1,s1,size1);
                 return 1;
      else {
                sys_write(1,s2,size2);
                return 2;

      }
}

This is working fine. But i have another problem. I wrote a function  int
string_len(char *s), which will return the
length of the string as follows.

int string_length(char *s)
{
        int len=0;

        while(*(s+len))
                len++;
        return len;
}

i am calling this in the sys_print_data instead of passing size1 and size2. Exactly
at this call it is saying
segmentation fault and dumping all the registers.

i also used the standard strlen call(linux/string.h).
It is also doing the same.

Can any body please clarify this.

Thanx in advance,
        Gopinath
```

Re:return values

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Re: system call returns "Bad Address". Why?
Re: Re:return values (C.H.Gopinath)
Date: Mon, 22 Dec 1997 08:41:22 GMT
From: Sameer Shah <ssameer@novell.com>

```
You cannot do the string_length because you are trying
to access a location that resides in the user space.
When switching to kernel mode, the data segment register
is changed to a location inside the kernel. But to allow
for such operations the kernel maintains address of
user's data segment in some other register (FS).
To access any string or some indirection data, you
have to actually copy that inside the kernel and then
you can go on with the normal strcpy functions. There
are few functions, I don't exactly recall their names
but with names like copy_fs_to_kernel, copy_kernel_to_fs
which allow you to copy between user and kernel spaces.

Just look at the implementation of some system call where
entire structures are passed (through a pointer to the
structure) e.g. ioctl() and you may need to do something
similar.

Hope this helps,
Sameer
```

possible reason for segmentation fault

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Re: system call returns "Bad Address". Why?
Re: Re:return values (C.H.Gopinath)
Date: Thu, 16 Oct 1997 18:52:04 GMT
From: <unknown>

```
I tried to implement the system call, the way you did. That is
passing 2 strings s1, s2 and finding out their lengths. It did
give me segmentation fault.
     I even tried to just print the strings within sys_print_data ()
using printk() as well as sys_write(), it did the same thing.
     The message it gave was that the kernel was unable to
do paging at virtual address xxxxx..
     I suppose there must be another way to pass strings to
the system call, but I don't know at this point. If I do in the
future, I will let you know.
```

saurabh desai.

Creating a new sytem call: solution

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Date: Sun, 12 Oct 1997 06:32:03 GMT
From: C.H.Gopinath <gopich@cse.iitb.ernet.in>

```
The problem is using printk, it won't print on the stdout,
it will be stored in the buffers. If you want the data
to be displayed on the stdout use

        sys_write(1,ptr,len);

where ptr is the string to be displayed and len is its length.

Using this you can check your sys call is created or not.

But regarding assigning a value to a global variable or local
variable i don't know. I am also struggling for the past
one week. I tried to assign and print a string in the system
call. It is compiling without any problem butwhen i try to
execute that, at the assignment it is giving

General Protection:000
and then dumping all the register values with Segmentation
Fault.

Can cny body explain why it is happeing like this.

By the by is there any kernel debugging tool for 4.2 kernel,
if so can anybody please give pointers for that.

Thanx in advance,
                        --
                                C.H.Gopinath
                                gopich@cse.iitb.ernet.in
```

Messages

1. Kernel Debuggers for Linux *by sauru*

Kernel Debuggers for Linux

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Re: Creating a new sytem call: solution (C.H.Gopinath)
Date: Sun, 12 Oct 1997 13:22:06 GMT
From: <sdesai@titan.fullerton.edu>

```
    First of all, I think that the segmentation fault you are getting
must be because of your system call code. You may want
to check it for any offending pointers.
    There are debuggers for the Linux kernel. They are as
follows.
(1) xkgdb :- this is the debugger that allows you to debug the
     ---------
kernel by putting the break points. It was developed by
John Heidemann <johnh@isi.edu>. It was later revised by
Keith Owens <kaos@ocs.com.au>. The latest version of
xkgdb is available for kernel 2.1.55 which is an experimental
kernel (risk of crashing). If you want you can obtain it from
<http://sunsite.unc.edu/pub/Linux/kernel/v2.1/>.


(2) kitrace :- This debugger traces the system calls. It is
     ----------
very useful. It was developed by Geoff <geoff@fmg.cs.ucla.edu>
You can obtain it from :
<http://ficus-www.cs.ucla.edu/ficus-members/geoff/kitrace.html>
This debugger will run smoothly on your Red Hat 4.2; kernel
2.0.30 kernel.
```

Hope this info becomes useful. good luck saurabh desai <sdesai@titan.fullerton.edu>

problem with system call slot 167

Forum: The Linux Kernel Hackers' Guide
Re: Problem creating a new system call
Re: Creating a new sytem call: solution (C.H.Gopinath)
Keywords: printk SYMBOL_NAME sys_call_table entry.S unistd.h
Date: Thu, 22 Jan 1998 17:18:44 GMT
From: Todd Medlock <medlota@nu.com>

```
I am using linux v2.0.33. In
 /usr/src/linux/arch/i386/kernel/entry.S
SYMBOL_NAME slots 164,165,166 are as follows:

.long 0,0
.long SYMBOL_NAME(sys_vm86)

I found that I could not add a new system call at 167. When
I did, it was called by something else for who knows what
reason. I know this because the only thing in my new system
call was a printk statement (which displays whenever the new
 system call is called). With the system call at 167 I would
receive unwanted printk messages at boot time, at shutdown
 time, and when I executed ifconfig! Hence, I put the
 following at 167 and put my new system call at 168.

.long 0

That seems to have made everything work! Another strange
thing is that with my system call at 167 the insmod
function reports "unresolved symbol" messages and will not
install modules?? My guess is that one of the modutil
modules is using the 167 slot! Anyone have any ideas?

Regarding printk:

It is my understanding that printk messages will appear on
the console (I am assuming you are at a console if you are
modifying system calls and generating the kernel ) as long
 as your message level is less then your log level. I use
 <0> for testing to be sure of having the lowest message level.
 For example:
```

```
printk("<0>syscallname: entering my test syscall\n");
```

This works for me.

Resetting interface counters

Forum: The Linux Kernel Hackers' Guide
Keywords: network interface counters
Date: Fri, 03 Oct 1997 22:10:44 GMT
From: Keith Dart <kdart@cisco.com>

Is there an ioctl or some other way to reset the network device counters (as shown in /proc/net/dev) to zero?

TIA, Keith Dart

writing/accessing modules

Forum: The Linux Kernel Hackers' Guide
Keywords: module
Date: Wed, 01 Oct 1997 12:43:36 GMT
From: Jones MB <jonesmb@ziplink.net>

I am writing a module whose main purpose is to allow a user app to change the values of some variables in the kernel's memory area. Using the modules in /usr/src/linux/drivers/net/ as a starting point, I have been able to create the module. I can insmod and rmmod it successfully (configreed via printk's to syslog). I am now looking for a way for the user level application to be able to access the module. I searched high and low for info on how to do this with no success. Any pointers in the right direction are most welcome.

Thanks

Messages

1. Use a device driver and read()/write()/ioctl() *by Michael K. Johnson* NEW

Use a device driver and read()/write()/ioctl()

Forum: The Linux Kernel Hackers' Guide
Re: writing/accessing modules (Jones MB)
Keywords: module
Date: Wed, 01 Oct 1997 14:12:17 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

There's a whole huge section of the KHG on writing device drivers. Register a character device driver and use either read()/write() or ioctl() to communicate between the user-level app and your module.

getting to the kernel's memory

Forum: The Linux Kernel Hackers' Guide
Re: writing/accessing modules (Jones MB)
Re: Use a device driver and read()/write()/ioctl() (Michael K. Johnson)
Keywords: module memory
Date: Fri, 03 Oct 1997 16:11:37 GMT
From: Jones MB <jonesmb@ziplink.net>

I have now been able to open and read/write to the module via a user level app. The point of the user app/module combination was to allow some variables to be changed in the kernel. These are variables which are created and malloc'ed when the module is loaded, so till the module comes up they do not exist. Now the part of the kernel that will use these variables will not compile as at compile time it does not know of the variables, so compiling fails. Is there a way to get around this?

JonesMB

use buffers!

Forum: The Linux Kernel Hackers' Guide
Re: writing/accessing modules (Jones MB)
Re: Use a device driver and read()/write()/ioctl() (Michael K. Johnson)
Re: getting to the kernel's memory (Jones MB)
Keywords: module memory
Date: Fri, 27 Feb 1998 13:27:39 GMT
From: Rubens <mytsplick@hotmail.com>

If you use a module with read() and write() functions, use the buffers that each functions has. Example: when you write to the module, the read() function is called. read() stores the received data in your buffer. Don't create and allocate variables, tranfer your data through buffers. If you have questions, please send e-mail.

RUBENS

Help with CPU scheduler!

Forum: The Linux Kernel Hackers' Guide
Keywords: scheduler
Date: Sun, 28 Sep 1997 05:45:53 GMT
From: Lerris <lerris@bigfoot.com>

Please help me if you can. I am doing a report on CPU scheduling in Linux. I have a copy of Linux Kernel Internals, but that does not help me very much because I am just now learning C. If you have a web page with a nice overview, or would be willing to provide one yourself, I would be eternally grateful!

Response to "Help with CPU scheduler!"

Forum: The Linux Kernel Hackers' Guide
Re: Help with CPU scheduler! (Lerris)
Keywords: scheduler
Date: Mon, 29 Sep 1997 19:34:43 GMT
From: Jeremy Impson <jdimpson@syr.edu>

I wrote something for Linux kernel 2.0.30. I'm not sure if it has changed in 2.1.x, or will in 2.0.31. But, it should still be helpful. Keep in mind that it is unedited, and no one (but myself) has checked it for accuracy. I actually wrote it for inclusion here in the KHG, but haven't gotten around to submitting it. It can be found at http://camelot.syr.edu/linux/scheduler.html Mr. michaelkjohnson, if you are interested, please feel free to copy it from that location to the KHG, or, if necessary, edit it to your heart's content. Just let me know when and if you do it. Thanks! --Jeremy Impson

Response to "Help with CPU scheduler!" (Redux)

Forum: The Linux Kernel Hackers' Guide
Re: Help with CPU scheduler! (Lerris)
Re: Response to "Help with CPU scheduler!" (Jeremy Impson)
Keywords: scheduler
Date: Wed, 22 Apr 1998 00:52:45 GMT
From: Jeremy Impson <jdimpson@syr.edu>

The URL in my previous message has changed. For now, it is available at

http://source.syr.edu/~jdimpson/camelot/linux/scheduler.html

It may change again, after I graduate, start my new job, and get a new Web acccount.

--Jeremy

calling interupts from linux

Forum: The Linux Kernel Hackers' Guide
Keywords: interrupts callable from C.
Date: Thu, 18 Sep 1997 14:55:46 GMT
From: John J. Binder <binder@cs.berkeley.edu>

I'm trying to figure out how to run irq 0x10 from gcc so as to interact with the video card directly. I believe it will have to be done with inline assembler.

```
 The general question is "How do you make interupts work from gcc.
```

To say get the video mode (coded for with 0x0f in register ah when irq 0x10 is called) I tried a fragment like:

```
___________
int ans;
__asm__ __volatile__ (
        "movb $0x0F,%%ah\n\t" \
        "int $0x10\n\t" \
        "movl %%eax,ans\n\t" \
        :"=memory" (ans) \
        :
        :"ax"
);
```

printf("ans='%d'\n",(int) ans);

____

But it gives a segmentation fault.

Permissions? ioperm?? whats the answer?

Thanks John

Messages

1. You can't *by Michael K. Johnson*

You can't

Forum: The Linux Kernel Hackers' Guide
Re: calling interupts from linux (John J. Binder)
Keywords: interrupts callable from C.
Date: Thu, 18 Sep 1997 15:09:43 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

If you had **read** the KHG, you would have discovered that you can only access interrupts from kernel code, not from user-level code. Please read what has already been written before you ask questions.

Calling BIOS interrupts from Linux kernel

Forum: The Linux Kernel Hackers' Guide
Re: calling interupts from linux (John J. Binder)
Re: You can't (Michael K. Johnson)
Keywords: interrupts callable from C.
Date: Thu, 25 Sep 1997 06:45:59 GMT
From: Ian Collier <imc@comlab.ox.ac.uk>

John J. Binder asks:

I'm trying to figure out how to run irq 0x10 from gcc so as to interact with the video card directly.

Michael K. Johnson replies:

If you had read the KHG, you would have discovered that you can only access interrupts from kernel code, not from user-level code.

May I take it then that it *is* possible from kernel code?

I have toured the KHG and found no mention of calling software interrupts. Receiving hardware interrupts is of course covered, but this is different.

It would be nice to be able to write a device driver for VBE2 display devices. This would give the application programmer access to the display, similarly to svgalib but with higher resolutions and more colours. It would also allow an X server to be written for those devices which are currently unsupported, including NeoMagic graphics adapters found in many laptops (including mine - hence my interest in this subject).

The VBE2 interface requires one to call `int 0x10' with, among other things, the real-mode address of a block of memory in ES:DI.

An alternative to the device driver would be to implement a system call like int86x() to emulate software interrupts in real mode and export the address mapping functions to user level code.

imc.

Messages

1. Possible, but takes work *by Michael K. Johnson*

Possible, but takes work

Forum: The Linux Kernel Hackers' Guide
Re: calling interupts from linux (John J. Binder)
Re: You can't (Michael K. Johnson)
Re: Calling BIOS interrupts from Linux kernel (Ian Collier)
Keywords: interrupts callable from C.
Date: Thu, 25 Sep 1997 20:27:48 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

I'm sorry, I didn't notice that he was talking about software interrupts in the BIOS. Not paying close enough attention...

In order to call anything in the BIOS, you need to put the processor in VM86 mode. From user mode, it is possible to do that in vm86 mode if you map the BIOS into the process's memory. dosemu does this in order to use the video bios. However, you won't be able to just call int 0x10. Linux reprograms the interrupt controller on boot. You can put code in to see what slot 0x10 points at and save that pointer and call it directly.

From kernel mode, you can look at how the APM bios calls are made in the file /usr/src/linux/drivers/char/apm_bios.c and copy how it is done there. Even in kernel mode, you need to get a pointer rather than blindly call int 0x10. And for 16-bit bioses, you need segment/offset, then use them to do an lcall. See arch/i386/boot/setup.S and arch/i386/kernel/setup.c for how these kinds of parameters get passed into the kernel on startup. The are recorded before the kernel goes into protected mode at all.

Note that this is completely dependent on not booting with a protected-mode boot utility that starts up the kernel already in 32-bit mode. Several such utilities exist, but they aren't much used at this point by Linux folks.

However, calling the BIOS is **slow**. It's fine for the apm stuff that doesn't need to be high-performance, but I wouldn't touch it for a video driver. Every call involves saving the processor state, changing processor mode, calling the bios, changing processor mode, and restoring the processor state. Assuming that you are calling into the kernel to do this, that's really an extra set of context switches. If you are doing it in a user-level library, you have device contention to deal with, as well as security issues, since you certainly need to be root to do this.

If I were in your shoes, I would try to use this interface only to set up a framebuffer, and then have the X server write to memory which is mmap()ed to that framebuffer. That will probably be faster than thunking through a 16-bit bios call level for every screen operation... There's a generic Linux framebuffer interface that is used on Linux/m68k, Linux/PPC, Linux/SPARC, and I think other platforms as well. You can start looking at that by reading /usr/src/linux/drivers/char/fbmem.c; I don't know the interface in any detail and can't help you beyond that.

If the bios is a 32-bit bios, you can skip saving state; that won't be such a problem. But since it wants a real mode address for a block in memory, I doubt that's the case.

Good luck, but I won't be able to be much more help than this.

VBE video driver

Forum: The Linux Kernel Hackers' Guide
Re: calling interupts from linux (John J. Binder)
Re: You can't (Michael K. Johnson)
Re: Calling BIOS interrupts from Linux kernel (Ian Collier)
Re: Possible, but takes work (Michael K. Johnson)
Keywords: interrupts callable from C.
Date: Fri, 26 Sep 1997 14:56:15 GMT
From: Ian Collier <imc@comlab.ox.ac.uk>

Thanks for your informative answer. I wonder if you can point me at any docs on VM86 mode.

Anyway, the idea was to create a device which represents the video memory that an application can just mmap and write to. This won't need to go through the BIOS so no performance problems there. VBE is supposed to provide a linear frame buffer without the need to do banking, and returns the physical address of the frame buffer in one of the query functions that you call when you want to set the video mode.

In order to change video modes, the application would (probably) do an ioctl on the device file. This would need to go through the BIOS, but it will only be called a few times per application anyway. There may be a separate device file where one can read and write the palette registers. VBE2 can give you a protected-mode address to call to do this instead of going via the interrupt, so performance should be acceptable. You also get a protected-mode interface for changing the viewport.

The application will of course require r/w permissions on the devices involved. The best way of doing this might be to arrange that the devices get chowned when one logs in on the console (although this will hinder silly tricks like switching from an X session to another virtual console and letting someone else log in and also run an X session).

imc

Messages

1. VM86 mode at which abstraction level? *by Michael K. Johnson*

VM86 mode at which abstraction level?

Forum: The Linux Kernel Hackers' Guide
Re: calling interupts from linux (John J. Binder)
Re: You can't (Michael K. Johnson)
Re: Calling BIOS interrupts from Linux kernel (Ian Collier)
Re: Possible, but takes work (Michael K. Johnson)
Re: VBE video driver (Ian Collier)
Keywords: interrupts callable from C.
Date: Fri, 26 Sep 1997 15:53:09 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

If you mean VM86 mode as implemented by the I386 and higher processors, you want James Turley's *Advanced 80386 Programming Techniques*. Unfortunately, it is out of print. Fortunately, he has given me permission to scan it in and put it on the web. Unfortunately, that's a slow process, nowhere near completed.

If you mean to ask how a user-space program can use Linux's vm86() syscall, use "man vm86". You may find a use to modify a process's ldt, in which case you will want to read "man modify_ldt". Those man pages may be slightly obsolete -- check them against recent dosemu and/or Wine source code.

It seems clear to me from your description that your job should be relatively easy to do as a kernel device driver, for two reasons:

1. You can make calls to some address from protected mode.
2. It sets up a framebuffer, and Linux already has a standard way to set up framebuffers through ioctl's, as I already mentioned.

Given those two considerations, you shouldn't have to know anything about vm86 mode at all.

DVD-ROM and Linux? (sorry if it's off topic...)

Forum: The Linux Kernel Hackers' Guide
Keywords: DVD DVD-ROM MPEG ISO9660
Date: Wed, 17 Sep 1997 19:22:47 GMT
From: Joel Hardy <deeng@inficad.com>

```
     Please forgive me (and don't flame me too hard) if this has
already been discussed or if there's someplace better to discuss
this. I know Linux should support any SCSI or IDE DVD-ROM drive,
but so far (at least to my knowledge), it'll only act like a CD-
ROM drive. I really don't know much about the DVD standard (if
anybody knows, please point me to some documentation!), but my
guess is that DVD-ROM discs (I think I saw Walnut Creek selling
4.7 gigs of stuff on one disc) would probably just be the same
ISO9660 standard, so there wouldn't be any need to support
anything extra with that. My question is this: is there any
support planned to read the MPEG streams and whatever else a DVD
player can get off of a DVD movie disc? Does anybody know anything
about the format that a DVD movie is stored in? Does anybody even
know if this'd be best implemented as a new filesystem or
something completely different? I'd really love to get this
working, but I'm a newbie to kernel programming, so if there's
anybody else out there with similar goals (and especially some
information about this!), please contact me!

-Joel Hardy (deeng@inficad.com)
```

DVD-ROM and linux

Forum: The Linux Kernel Hackers' Guide
Re: DVD-ROM and Linux? (sorry if it's off topic...) (Joel Hardy)
Keywords: DVD DVD-ROM MPEG ISO9660
Date: Tue, 28 Jul 1998 14:44:16 GMT
From: <Yuqing_Deng@brown.edu>

We need to impelement UDF file system to read DVD-ROM on linux. There is a UDF project, check out the URL

http://www.netcom.ca/~aem/udf/

What I understand is that, the encryt algorithms should only be impelemented on hardware according to the DVD standard. That might acctually make life easier. We only have to write a drive for the decoder card.

Response to DVD and Mpeg in Linux

Forum: The Linux Kernel Hackers' Guide
Re: DVD-ROM and Linux? (sorry if it's off topic...) (Joel Hardy)
Keywords: DVD DVD-ROM MPEG ISO9660
Date: Sun, 12 Apr 1998 09:30:24 GMT
From: Mike Corrieri <mc@imagine-software.com>

Well...

To the best of my understanding we could IF we could get the encryption software. But it could not be under the GPL. It would have to be a 4sale application for Linux.

DVD movie roms have a special copy encryption, that is specific to an area the DVD is sold in. Pretty scary, heh?

So, if you bought your unit in Europe, it would not work in the usa.

Anyways, if we could get the encryption software, under development OEM license, it would conflict with the GPL. We would not be able to make the code public.

I would like to see an answer to this myself, having to continue running Win 95 ONLY FOR DVD MOVIES!

DVD Encryption

Forum: The Linux Kernel Hackers' Guide
Re: DVD-ROM and Linux? (sorry if it's off topic...) (Joel Hardy)
Re: Response to DVD and Mpeg in Linux (Mike Corrieri)
Keywords: DVD DVD-ROM MPEG ISO9660
Date: Tue, 26 May 1998 15:09:08 GMT
From: Mark Treiber <mrtreibe@engmail.uwaterloo.ca>

I checked the creative site and for their player you have to add the country code when its installed and then its permanent. I'm assuming that they are talking about the drive so if the drive is already setup, reading from it should be okay without worrying about the encryption.

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: DVD-ROM and Linux? (sorry if it's off topic...) (Joel Hardy)
Re: Response to DVD and Mpeg in Linux (Mike Corrieri)
Re: DVD Encryption (Mark Treiber)
Keywords: DVD DVD-ROM MPEG ISO9660
Date: Mon, 01 Jun 1998 11:09:26 GMT
From: Tim <ice@space.net.au>

Yes, I believe the creative dvd bundled mpeg1&2 decoder card performs the decryption. It is possible (but probably not legal) to change the region supported by the card many many times by writing some info to a flashrom on the card. DVD under linux is possible, however am not sure if it is 100% legal. I am not sure if DVD-ROMS support two different access methods - normal and dvd
.

DVD?

Forum: The Linux Kernel Hackers' Guide
Re: DVD-ROM and Linux? (sorry if it's off topic...) (Joel Hardy)
Re: Response to DVD and Mpeg in Linux (Mike Corrieri)
Re: DVD Encryption (Mark Treiber)
Re: Untitled (Tim)
Keywords: DVD DVD-ROM MPEG ISO9660
Date: Thu, 16 Jul 1998 23:51:42 GMT
From: <unknown>

```
It's time we hacked it all up otherwise we're pretty much stuck with win98 ;(  If
anyone needs a box to test on I got one ;) -FireBall
```

Kernel Makefile Configuration: how?

Forum: The Linux Kernel Hackers' Guide
Keywords: Makefile, configuration
Date: Sun, 14 Sep 1997 03:41:59 GMT
From: Simon Green <sgreen@emunet.com.au>

```
I'm aware of the make config/menuconfig/xconfig etc., but a
program I am writing for Uni will require that I am able to
directly configure the Makefile(s)...

Can anyone give me a general overview of where all the
CONFIG_blah_blah_blahs go? I had a look at the Makefile, but it
wasn't very instructive. For example, in drivers/net/Makefile:

.
.
.
ifeq($(CONFIG_NE2000),y)
L_OBJS += ne.o
.
.
.

Where does CONFIG_NE2000 get defined? I'd appreciate it if someone
could tell me the general rule.
Thanks
```

How to add a driver to the kernel ?

Forum: The Linux Kernel Hackers' Guide
Re: Kernel Makefile Configuration: how? (Simon Green)
Keywords: Makefile, configuration, drivers, epic100
Date: Mon, 22 Dec 1997 07:08:52 GMT
From: jacek Radajewski <jacek@usq.edu.au>

```
Hi All,

I am in the process of rebuilding our beowulf cluster system and have to include
support for channel bonding and the epic100 SMC card.  So far the
epic100.o module works fine, but I need to compile epic100.c into the
kernel. (I need a monolithic kernel to boot clients of a floppy disk and
mount / via NFS).  Anyway, I put an antry in .config "CONFIG_EPIC100=y",
in drivers/net I put :

ifeq ($(CONFIG_EPIC100),y)
L_OBJS += epic100.o
endif

in the Makefile and compiled the kernel. (epic100.o did compile)

I added an entry in /etc/lilo.conf :

append="root=/dev/hda4 mem=256MB ether=0,0,eth0 ether=0,0,eth1
ether=0,0,eth2"

and ran lilo

"I have 2 SMC cards and 3C900 for the outside world"

When I rebooted the machine only 3C900 was detected, but I have no
problems loading the module.

Questions.

1.  What did I do wrong in the kernel configuration ?
2.  Is there any documentation available on how to add drivers to the
kernel ?
```

See include/linux/autoconf.h

Forum: The Linux Kernel Hackers' Guide
Re: Kernel Makefile Configuration: how? (Simon Green)
Keywords: Makefile, configuration
Date: Mon, 13 Oct 1997 12:05:56 GMT
From: Balaji Srinivasan <BalajiSrinivasan>

When you run make config (or its siblings) it creates a file in include/linux directory. This file (autoconf.h) is included in include/linux/config.h in all the required C files...

For the makefile the place that these config options are specified is in the .config file in the TOPLEVEL directory.

Hope this helps balaji

Multiprocessor Linux

Forum: The Linux Kernel Hackers' Guide
Keywords: SMP multiprocessor
Date: Tue, 09 Sep 1997 19:18:33 GMT
From: Davis Terrell <caddy@csh.rit.edu>

If anyone could tell me how or point to information on setting up Linux 2.x (RedHat 4.2) for SMP support I would be very grateful... thanks...

Building an SMP kernel

Forum: The Linux Kernel Hackers' Guide
Re: Multiprocessor Linux (Davis Terrell)
Keywords: SMP multiprocessor
Date: Wed, 10 Sep 1997 11:03:27 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

1. Get the **latest** 2.0.x kernel sources from ftp.kernel.org. I recommend waiting for 2.0.31, which should be out soon (as of this writing) and fixes some deadlocks in SMP.
2. Unpack it in /usr/src
3. Edit the Makefile and uncomment the SMP = 1 line near the top.
4. `make config` and choose your configuration.
5. `make clean; make dep`
6. `make zImage; make modules`
7. Move the kernel into place, `make modules_install`
8. Run lilo and reboot. Keep a known-working kernel around to revert to

Note: You **must** make new modules for your SMP kernel. Loading modules that are built for a non-SMP kernel into an SMP kernel (and vice versa) breaks systems horribly.

SMP and module versions

Forum: The Linux Kernel Hackers' Guide
Re: Multiprocessor Linux (Davis Terrell)
Re: Building an SMP kernel (Michael K. Johnson)
Keywords: SMP multiprocessor
Date: Tue, 28 Oct 1997 21:20:40 GMT
From: <linux@catlimited.com>

I have not had any trouble configuring and compiling the kernel and modules for SMP, but I cannot get it to boot properly. It complains about the module versions and won't load them. There must be some aspect of the config I am missing. Can anyone tell me what I am missing here?

Thanks Bruce

Improving event timers?

Forum: The Linux Kernel Hackers' Guide
Keywords: event timers improvement
Date: Fri, 05 Sep 1997 16:15:40 GMT
From: <bodomo@hotmail.com>

```
I want to evaluate my new implementation of event timers, and
compare its performance with the one I have (2.0.29) for different
usage patterns.

Three questions about add_timer, init_timer, del_timer:

- I want to know how existing apps (x,tcp/ip,ppp,latex, etc) use
event timers. Can I just trace some syscall using strace (which
ones?)? The alternative would be instrumenting the kernel to keep
track of the calls, that would take me more time coding.

- are these primitives C library wrappers around system calls that
do the same, or does the C part implement more functionality on
top of the kernel part?

- which library contains the primitives for C? gcc can't compile
because the linker can't find add_timer, ... I tried to find where
they are with "nm", but they don't seem to be in any file in
my/usr/lib. Should I download another library from some internet
site?


Thanks a lot,
  Guillermo
  bodomo@hotmail.com
```

measuring time to load a virtual mem page from disk

Forum: The Linux Kernel Hackers' Guide
Keywords: virtual memory linux timing time
Date: Tue, 02 Sep 1997 13:23:33 GMT
From: <kandr@giasmd01.vsnl.net.in>

Is it possible to track the number of page-faults that occured during the course of a Linux session? I want all kinds of intricate details like how long it took from the time of the page fault occuring to the time the system recovers by loading the page from disk. (I know the the 'time' command can give the number of page-faults but I also need to know the total time taken to service the page faults.)

If a readymade utility is not available, would anyone please suggest ways in which to modify the kernel so that I can collect these statistics.

Thanks in advance. And cheers to the KHG!

-- Ranganathan <kandr@giasmd01.vsnl.net.in>

using cli/sti() and save_flags/restore_flags()

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 30 Jul 1997 21:43:57 GMT
From: george <georgel@cs.ucla.edu>

snippets of kernel code look like the following

```
  save_flags(flags);
  cli();
  ...
  restore_flags(flags);
```

does restore_flags() somehow magically call sti()?

the answer: yes.

why/how?

remember that interrupt enable is _also_ a flag. thus restoring it would take care of everything. be careful _not_ to simply call sti(). sti() would enable interrupts, even if they were disabled to start with. in this case, you have just messed things up, as the example below illustrates

```
foo(){
  cli();
  ...
  sti();
}

bar(){
  cli();
  foo();  /* must not mess with cli() setting */
  this_must_have_ints_off();
  sti();
  foo();
}
```

this was in response to a query i had about their use and there wasn't anything in the khg about this. hence, i am adding this knowledge so others may know.

Protected Mode

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 24 Jul 1997 02:10:25 GMT
From: ac <*unknown*>

Please need information about protected mode. I've been looking for books but they only give you routines to enter and leave from pm. Is there any good reference? such as information about TSS,LDT,etc. I don't want an introductory text,but I'm wondering if there exists some book about this (if posible with example sources)

thanks

ac

Messages

1. Advanced 80386 Programming Techniques *by Michael K. Johnson* NEW

Advanced 80386 Programming Techniques

Forum: The Linux Kernel Hackers' Guide
Re: Protected Mode (ac)
Date: Thu, 24 Jul 1997 03:22:21 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

The book you really want is Advanced 80386 Programming Techniques. Unfortunately, it is out of print.

Fortunately, the author, Jim Turley, has expressed interest in getting the book on the web. That will be a somewhat long process, but at some point it should actually be available. When it is available, there will be a link to it in the annotated bibliography. In the meantime, you'll just have to browse your local bookstore for useful books on the subject.

'Developers manual' from Intel(download)...

Forum: The Linux Kernel Hackers' Guide
Re: Protected Mode (ac)
Date: Wed, 15 Oct 1997 21:22:56 GMT
From: Mats Odman <mats-odm@dsv.su.se>

If you have the time, there are "Pentium Processor Family Developers Manual, vol3 Architectur and programming manual" available for download from Intel, only problem is: it's about 1000 pages to print :-)

DMA buffer sizes

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 22 Jul 1997 15:19:29 GMT
From: <nomrom@hotmail.com>

I want to allocate a DMA buffer larger than 128*1024. Is it possible to configure the kernel to allow bigger DMA buffer sizes? I've attempted to increase the PAGE_SIZE to 8192 but that crashes the system.

DMA limits

Forum: The Linux Kernel Hackers' Guide
Re: DMA buffer sizes
Keywords: DMA limit
Date: Sat, 26 Jul 1997 02:16:07 GMT
From: Albert Cahalan <acahalan at cs.uml.edu> <*unknown*>

```
> I want to allocate a DMA buffer larger than 128*1024.
> Is it possible to configure the kernel to allow bigger
> DMA buffer sizes? I've attempted to increase the PAGE_SIZE
> to 8192 but that crashes the system.
```

I hope that is not PC hardware! The ISA bus has a hard limit of 128 kB (16-bit DMA) or 64 kB (8-bit).

Even 128 kB is hard though, because memory fragmentation makes it unlikely that you can allocate a contiguous 128 kB chunk under the 16 MB ISA DMA limit (or elsewhere).

Not page size, page order

Forum: The Linux Kernel Hackers' Guide
Re: DMA buffer sizes
Date: Thu, 24 Jul 1997 03:25:43 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

You want to change the largest order of pages available from 5 to 6 -- that will give you twice as large regions.

Problem Getting the Kernel small enough

Forum: The Linux Kernel Hackers' Guide
Keywords: Kernel too large
Date: Mon, 21 Jul 1997 00:54:30 GMT
From: <afish@tdcdesigncorps.com>

```
 I've been trying to get my Kernel small enough to boot from a floppy (zImage and
bzImage).  Removed all unecessary drivers, etc.. however I still can't get it on a
floppy.  I know its possible anyone have any global ideas about what exactly I am
ignorant about.  By the way I've read all the Howto's and tried it on both 2.0.xx and
2.1.xx kernels and still get the same problem.
```

Check it's the right file, zImage not vmlinux

Forum: The Linux Kernel Hackers' Guide
Re: Problem Getting the Kernel small enough
Keywords: Kernel too large
Date: Fri, 16 Jan 1998 22:39:25 GMT
From: Cameron <cls@greens.org>

I get questions like that every week, from people making LILO floppies. The most common problem is that they are trying to install

/usr/src/linux/vmlinux

on their floppy.

That's the wrong file! It's too big!

The file you probably wanted was

/usr/src/linux/arch/i386/boot/zImage

Usually easy, but....

Forum: The Linux Kernel Hackers' Guide
Re: Problem Getting the Kernel small enough
Keywords: Kernel too large
Date: Thu, 15 Jan 1998 08:32:12 GMT
From: Ian Carr-de Avelon <ian@emit.com.pl>

Normally it is easy. A typical zImage is 300-400kb so fits fine on a 1.4MB floppy. Obviously it depends on what you include, but normally you can get a whole compressed file system on there as well and run Boot/Root. Like in the HOWTO of the same name. How big is your kernel and floppy

Ian

How to create /proc/sys variables?

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 11 Jul 1997 11:08:50 GMT
From: Orlando Cantieni <ocantien@itr.ch>

Hi there

I'im going to do a kernel hack. I plan to insert a static variable (that is: static int) into the modified kernel. The value of it should be tunable by an external program.

Now, How to do so ? I think best way is to create a variable into the /proc/sys mirror and tune it by echo. But how can I create such a file ? Does sysctl() do something ?

Thanks for any help

Orla

-- Orlando Cantieni .. ITR Rapperswil ocantien@itr.ch .. www.itr.ch/~ocantien

"ERROR: Volume Earth is full up to 99%. Please delete some users."

Linux for NeXT black?

Forum: The Linux Kernel Hackers' Guide
Keywords: port NeXT
Date: Thu, 03 Jul 1997 02:49:45 GMT
From: Dale Amon <amon@gpl.com>

Could some one email me some good starting points for the Linux system porting bootstrapping process? I'm looking at what it would take to do a port to NeXT black.

vremap() in kernel modules?

Forum: The Linux Kernel Hackers' Guide
Keywords: vremap() kernel module map
Date: Tue, 01 Jul 1997 14:49:07 GMT
From: Liam Wickins <lwickins@mpc-data.co.uk>

Is vremap() the only way of mapping in an area of physical memory to a virtual address? I understand that its use is restricted to drivers statically compiled into the kernel. I want to be able to map in an area of physical memory (0xd0000, say) within a kernel module. Is this possible?

giveing compatiblity to win95 for ext2 partitions (for programmers forced to deal with both)

Forum: The Linux Kernel Hackers' Guide
Date: Sat, 28 Jun 1997 01:05:46 GMT
From: pharos <noonie@toledolink.com>

```
well working on a deviced driver to use linux partitions is not easy, I wanted to
make ext2 partitions look like a cdrom so I can use mscdex to give it a drive letter.
I would appreciate any low level info on ext2 partitions or if something like this
extists, for me to be told so I don't waste my time coding it ;P
 hows this sound?
devicehigh=c:\pharos\ext2.sys -d:mscd0002 -p=/dev/hda8
```

Well, What's the status of the Windows / Dos driver for Ext2?

Forum: The Linux Kernel Hackers' Guide
Re: giveing compatiblity to win95 for ext2 partitions (for programmers forced to deal with both) (pharos)
Keywords: win95 ext2 driver
Date: Thu, 14 May 1998 16:57:15 GMT
From: Brock Lynn <brock@cyberdude.com>

Just wondering what the status is. I can't seem to get to the address: http://www.globalxs.nl/home/p/pvs/. Looking to see if there has been an update, but can't get there.

Though a few months ago I had someone on irc.linpeople.org go there and send me the proggie via email. It didn't work so great, and every time I mounted an ext2, and then opened a few directories as win95 folders Win95 would go down in flames.

I tried the ext2tools and they work quite well, but very awkward.

Anyluck with the IFS, or mscdex ???

Microsoft is such a turd... Why not open up their standards so more people can develop for it. Is MS afraid that someone smarter than they are will come along and actually improve upon their standards??? (probably so :)

Well, what's the word?

Brock Lynn brock@cyberdude.com

Working on it!

Forum: The Linux Kernel Hackers' Guide
Re: giveing compatiblity to win95 for ext2 partitions (for programmers forced to deal with both) (pharos)
Keywords: ext2 for Win95, WinNT, and maybe DOS
Date: Tue, 05 Aug 1997 15:26:48 GMT
From: ibaird <topdawg@grfn.org>

```
I'm currently working on a driver for Windows 95 and Windows NT that will
run under the IFS (Installable File System) for Win32. I'm at the stage
right now where I can successfully read in the superblock and the group
descriptors of a test ext2 partition in my concept testing program and am
about ready to attempt to read the root directory. In a few days
(hopefully) I'm going to be able to read some files from the drive. Write
access will looks like it will take longer (because of the allocation
algorithms and other bull).
```

revision

Forum: The Linux Kernel Hackers' Guide
Re: giveing compatiblity to win95 for ext2 partitions (for programmers forced to deal with both) (pharos)
Re: Working on it! (ibaird)
Keywords: ext2 for Win95, WinNT, and maybe DOS
Date: Fri, 08 Aug 1997 13:51:47 GMT
From: ibarid <topdawg@grfn.org>

```
Since I first wrote my message I've discovered Microsoft wants about $1,000
for any info regarding their installable file system. Does anyone know
anything about it?
```

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: giveing compatiblity to win95 for ext2 partitions (for programmers forced to deal with both) (pharos)
Re: Working on it! (ibaird)
Re: revision (ibarid)
Keywords: ext2 for Win95, WinNT, and maybe DOS
Date: Fri, 14 Nov 1997 05:33:58 GMT
From: Olaf <in5y003@public.rrz.uni-hamburg.de>

I have tried to find information about the IFS before, actually for doing the same thing (an ext2fs driver for Win95, WinNT and DOS). These are the best references I found:

```
- Inside the Windows 95 File System by Stan Mitchell,
  O'Reilly in May '97
  Gives many details (esp. on VFAT and IFSMgr) and includes
  Multimon, a tool to sniff on INTs, VXDs and so on.

- Systems Programming for Windows 95 by Walter Oney,
  Microsoft Press in '96 (he claims to have no affiliation
  with MS other that they published his book).
  Is more general than the above but has a very interesting
  chapter about the IFS. Describes how to write a VXD.
```

There are other interesting books about DOS/Windows internals, especially those written by Andrew Schulman and Geoff Chapell. (Write me if you need references).

I stopped working on this due to lack of time. But my approach started out as this: I was looking at the ext2tools, a package for DOS providing rudimetary ext2 access through special commands (like e2dir, e2cat and so on), without providing a drive letter. They were build from a snapshot of the ext2fs kernel sources, glued together with a library doing regular expressions (for filename matching) and getting a pointer to the partition through an environment variable. The disk accesses were done via the plain BIOS IRQ 13.

I wanted to make all of this into a drive letter based approach and wanted to put together the current ext2fs from the linux kernel, get a VXD running and answer IFS requests.

Someone else seems to have a read-only version of this running now. You should perhaps contact Peter van Sebille or read his page at http://www.globalxs.nl/home/p/pvs/. You can find the driver there as well.

setsockopt() error when triying to use ipfwadm for masquerading

Forum: The Linux Kernel Hackers' Guide
Date: Sat, 21 Jun 1997 20:05:37 GMT
From: <omaq@encomix.es>

```
Hi, I´m having such as a nasty problem...setsockopt(), I´d like  my LAN in internet
masquerading all the 192.168.1.X directions as the real one on my providers side. I
have compiled 2.0.27 with IP forwarding and IP firewalling....that´s correct ?????
Thanks in advenced for any clue.
```

Re: masquerading

Forum: The Linux Kernel Hackers' Guide
Re: setsockopt() error when triying to use ipfwadm for masquerading
Keywords: masquerade
Date: Mon, 23 Jun 1997 13:57:46 GMT
From: Charles Barrasso <charles@blitz.com>

I am assuming when you compiled the kernel you tured on masquerading too. Right?

you can check by doing

ls /proc/net/ and you should see

ip_masquerade if you don't then that could be why. I have never gotten that error so if it is not that I don't konw what to tell you.

Hope this helps,

Charles

reset the irq 0 timer after APM suspend

Forum: The Linux Kernel Hackers' Guide
Keywords: timer, APM
Date: Fri, 20 Jun 1997 21:34:04 GMT
From: Dong Chen <chen@ctp.mit.edu>

```
Hello,

On my AST J50 (P133) notebook, the timer on irq 0 resets itself from
interrupting at 100 Hz to 18.3 Hz (DOS default) after a suspend/resume.

What is the right way to re-initialize it back to 100 Hz without a reboot?
Thanks,

Dong Chen
chen@ctp.mit.edu
---------------------------------------------------------------------
BTW, I tried to modify

        linux/drivers/char/apm_bios.c

in function suspend() at line 639 (kernel 2.0.30) from

        err = apm_set_power_state(APM_STATE_SUSPEND);
        if (err)
                apm_error("suspend", err);
        set_time();

to
        err = apm_set_power_state(APM_STATE_SUSPEND);
        if (err)
                apm_error("suspend", err);

        save_flags(flags);
        cli();
        /* set the clock to 100 Hz */
        outb_p(0x34,0x43);              /* binary, mode 2, LSB/MSB, ch 0 */
        outb_p(LATCH & 0xff , 0x40);    /* LSB */
        outb(LATCH >> 8 , 0x40);        /* MSB */
        restore_flags(flags);

        set_time();

But this does not work.  All programs crash after suspend() is called.
```

Re: fixed, patch for kernel 2.0.30

Forum: The Linux Kernel Hackers' Guide
Re: reset the irq 0 timer after APM suspend (Dong Chen)
Keywords: timer, APM
Date: Fri, 27 Jun 1997 15:44:46 GMT
From: Dong Chen <chen@ctp.mit.edu>

```
(This is the message I sent to the linux-kernel mailing list)

Hi,

This is a patch for "drivers/char/apm_bios.c", it fixes the following
problems:

(1) On some notebooks (AST J series, for example), the timer on interrupt 0
is reset to DOS default: 18 Hz.  This patch re-initialize it to 100 Hz.
Thanks to Pavel (pavel@Elf.mj.gts.cz) for pointing out to me that I should
add some delays after the outb_p() and outb() calls.

(2) The clock is not correctly restored after a standby().

There are still some problems with not getting the correct time after APM
suspend or standby, namely before the first suspend() or standby()
call, if the clock is already slowed by CPU_IDLE call, then the estimate
time zone "clock_cmos_diff" would be wrong.  Ideally, "clock_cmos_diff"
should be setup at boot time after the time zone is set.  But that
will require changing code other than "apm_bios.c".  Also, APM will not
correct for the change between daylight savings time and normal time.

Dong Chen
chen@ctp.mit.edu

---------------------------CUT HERE------------------------------------
--- drivers/char/apm_bios.c.orig        Mon May 26 11:05:15 1997
+++ drivers/char/apm_bios.c     Tue Jun 24 12:09:06 1997
@@ -73,6 +73,18 @@
 #include <linux/miscdevice.h>
 #include <linux/apm_bios.h>

+/*
+ * INIT_TIMER_AFTER_SUSPEND: define to re-initialize the interrupt 0 timer
+ * to 100 Hz after a suspend.
+ */
+#define INIT_TIMER_AFTER_SUSPEND
+
+#ifdef INIT_TIMER_AFTER_SUSPEND
+#include <linux/timex.h>
```

```
+#include <asm/io.h>
+#include <linux/delay.h>
+#endif
+ static struct symbol_table     apm_syms = {
 #include <linux/symtab_begin.h>
        X(apm_register_callback),
@@ -627,28 +639,53 @@
        unsigned long   flags;
        int             err;

-                               /* Estimate time zone so that set_time can
-                                  update the clock */
-       save_flags(flags);
-       clock_cmos_diff = -get_cmos_time();
-       cli();
-       clock_cmos_diff += CURRENT_TIME;
-       got_clock_diff = 1;
-       restore_flags(flags);
+       if (!got_clock_diff) {
+                               /* Estimate time zone */
+               save_flags(flags);
+               clock_cmos_diff = -get_cmos_time();
+               cli();
+               clock_cmos_diff += CURRENT_TIME;
+               got_clock_diff = 1;
+               restore_flags(flags);
+       }

        err = apm_set_power_state(APM_STATE_SUSPEND);
        if (err)
                apm_error("suspend", err);
+
+#ifdef INIT_TIMER_AFTER_SUSPEND
+       cli();
+        /* set the clock to 100 Hz */
+        outb_p(0x34,0x43);              /* binary, mode 2, LSB/MSB, ch 0 */
+        udelay(10);
+        outb_p(LATCH & 0xff , 0x40);    /* LSB */
+        udelay(10);
+        outb(LATCH >> 8 , 0x40);        /* MSB */
+        udelay(10);
+#endif
+
        set_time();
 }

 static void standby(void)
 {
+       unsigned long   flags;
        int     err;

+       if (!got_clock_diff) {
+                               /* Estimate time zone */
+               save_flags(flags);
+               clock_cmos_diff = -get_cmos_time();
+               cli();
+               clock_cmos_diff += CURRENT_TIME;
+               got_clock_diff = 1;
+               restore_flags(flags);
+       }
+
        err = apm_set_power_state(APM_STATE_STANDBY);
        if (err)
                apm_error("standby", err);
+       set_time();
 }

 static apm_event_t get_event(void)
```

Source Code in C for make Linux partitions.

Forum: The Linux Kernel Hackers' Guide
Keywords: source code, partition
Date: Mon, 09 Jun 1997 07:24:11 GMT
From: Limbert Sanabria <limbert@bo.net>

Where I can get the Source Code in C for make Linux partitions?, Any idea about where, I can get information about this?

Please reply cc to limb@hotmail.com or limbert@bo.net

Thanks.

Limbert

Messages

1. Untitled *by lolley* NEW

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: Source Code in C for make Linux partitions. (Limbert Sanabria)
Keywords: source code, partition
Date: Tue, 10 Jun 1997 03:09:08 GMT
From: Wu Min <wumin@sunny.bjnet.edu.cn>

I think you can see fdisk's source code. Maybe helpful.

How can I "cheat" and change the IP address (src,dest) in the sent socket?

Forum: The Linux Kernel Hackers' Guide
Date: Sat, 07 Jun 1997 17:18:36 GMT
From: Rami <s2623500@cs.technion.ac.il>

Hi, Me and my partner are trying to implement a LocalDirector (as a project in network course), so we want to change the IP address in the socket to be sent to a local server from the LocalDirector.

Thanks

Messages

1. Do it in the kernel *by Michael K. Johnson* NEW

Do it in the kernel

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Date: Sat, 14 Jun 1997 01:31:25 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

The only way to do this in user space is to do something like diald, where a program talks SLIP (or you might choose PPP) to the kernel over a pty or two, and routes traffic back and forth through itself, making modifications.

The more reasonable way to do this is to put it in the generic network filtering. You can either do simple rewrites with the existing firewall tools or write your own firewall modules and drop them into the stack. That way you can give yourself the option of making arbitrary modifications to packets on their way in and/or out of the system.

Read Network Buffers And Memory Management first to learn about how the networking stack works, then read the ipfwadm code and the relevant kernel code. Good luck.

Transparent Proxy

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Keywords: ip network address transparent proxy masquerade
Date: Mon, 22 Jun 1998 22:16:01 GMT
From: Zygo Blaxell <zblaxell@furryterror.org>

Linux Transparent proxy support (part of the firewalling stuff) is designed to do exactly this.

There are basically two "halves" to transparent proxy:

1. You can `bind` to any address you like, instead of choosing from the addresses of interfaces on the machine.
2. You can collect SYN packets (generated by clients doing `connect`) on a port of your choice. You can do a `getsockname` to find out what address+port number the client *thinks* it connected to, and there are more fields in the "from" parameter of `recvfrom` that you can use to find out where a datagram was destined.

So if you want to connect to a server while pretending to have some other IP address, you simply do a `bind` system call on the socket before `connect`ing. The address you bind to is the address you want to appear to be. This is just like doing a `bind` with a specific IP address or port number when you want a specific network interface or when you want a port number below 1024 for `rcmd`-based services, except that now you specify an IP address other than your own.

If you're doing UDP, then you might want to do this with the `sendto` and `recvfrom` system calls, in which case the source address is specified in the second 8 bytes of the socket address for the destination address in `sendto` and vice-versa for the source address in `recvfrom`.

Put another way, when you do a `sendto`, you put the destination address in the "to" parameter as usual, but you also put the desired source address (which is not the "usual" one) in the "to" parameter + 8 bytes. Note that you must OR in `MSG_PROXY` to the flags parameter for `sendto`/`recvfrom`.

Note that in order to use any of the transparent proxy features you must be `root`. Generally this is most useful when the host doing transparent proxy is a gateway or router of some kind, because impersonating host A when connecting to host B will only work if host B will normally try to send packets to host A through your host.

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Date: Tue, 17 Mar 1998 08:56:35 GMT
From: <qwzhang@public2.bta.net.cn>

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Date: Sun, 14 Dec 1997 11:35:47 GMT
From: <navin97@hotmail.com>

how to change IP number? when I chat in IRC, I don't want somone know where am I from? then, wanna change IP number... can I? could you please let's me know?... thanks!

Changing your IP address is easy, but...

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Re: Untitled
Keywords: ip firewall router socket network address
Date: Mon, 22 Jun 1998 21:58:15 GMT
From: Zygo Blaxell <zblaxell@furryterror.org>

If you change your IP address on your client's socket to an "anonymous" IP address (one on a different physical subnet than your own assigned address), you will not be able to receive replies sent to that IP address unless you also manipulate the routing tables of all of the routers between the IRC server and your "anonymous" client. You probably can't do that, so it's not actually useful to know how.

Note that if your machine is physically on an ethernet segment with a subnet, you could just change your machine's IP address to a different address *within the same subnet*, which would obscure your identity with that of another user on the same subnet (i.e. "they" will know what company or ISP you're from but not which particular user, unless they have some other information to identify you). Cable modems are good for this, as they often have little security or accounting and lots of spare addresses to choose from.

Kids, don't try this at home. People who can afford lawyers get seriously offended if you steal their vacant IP addresses.

You have to know a bit of C (if u wanna learn) ;)

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Re: Untitled
Date: Thu, 15 Jan 1998 00:12:45 GMT
From: Lorenzo Cavallaro <lc529863@silab.dsi.unimi.it>

```
Yo,
You have either to write your own client  IRC, or to modify
the source Code
one of the most famous client IRC: ircII (I dunno the actual
release ... maybe 2.9.x).

If u dont have ircII, well it's not hard to find it on the
Net ...
For further info, check out www.2600.com and look for the
FAQ (even old ones)

Bye
```

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: How can I "cheat" and change the IP address (src,dest) in the sent socket? (Rami)
Date: Thu, 10 Jul 1997 01:53:57 GMT
From: <unknown>

Where is the source file for accept()

Forum: The Linux Kernel Hackers' Guide
Keywords: Networking Systemcall
Date: Wed, 04 Jun 1997 19:12:37 GMT
From: <kaixu@hocpa.ho.lucent.com>

Can anybody point me to the Linux source file that contains the implementation of accept() system call ? Thanx

Messages

1. Here, in /usr/src/linux/net/socket.c *by wumin@netchina.co.cn*

Here, in /usr/src/linux/net/socket.c

Forum: The Linux Kernel Hackers' Guide
Re: Where is the source file for accept()
Keywords: Networking Systemcall
Date: Thu, 05 Jun 1997 02:04:39 GMT
From: Wu Min <wumin@netchina.co.cn>

asmlinkage int sys_accept(.....

How can I use RAW SOCKETS in UNIX?

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 02 Jun 1997 15:34:59 GMT
From: Rami <s2623500@cs.technion.ac.il>

Messages

1. Re: Raw sockets *by genie@risq.belcaf.minsk.by* NEW

Re: Raw sockets

Forum: The Linux Kernel Hackers' Guide
Re: How can I use RAW SOCKETS in UNIX? (Rami)
Date: Wed, 04 Jun 1997 13:15:33 GMT
From: <genie@risq.belcaf.minsk.by>

To use RAW sockets in Unix it it mandatory that one be a root . To create RAW socket just write: s=socket(AF_INET,SOCK_RAW,<protocol>). Then you can do anything you want with it (sending, receiving). However you have to perform all necessary operations, according to the protocol you use (create headers (IP+TCP(UDP,ICMP,...)) and make all neccessary negotiations (TCP: SYN->ACK->....RST....ACK....).

the KHG in spanish?

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 27 May 1997 03:28:04 GMT
From: Jorge Alvarado Revatta <j_alvarado@geocities.com>

```
   where it's ?
                        Jorge Alvarado Revata
Universidad Nacional de San Marcos Lima Peru
```

Messages

1. No esta aqui! Pero... *by Michael K. Johnson* NEW

No esta aqui! Pero...

Forum: The Linux Kernel Hackers' Guide
Re: the KHG in spanish? (Jorge Alvarado Revatta)
Date: Sat, 14 Jun 1997 01:42:51 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

There is no one translating the KHG into Spanish. However, if you want to start, I'll be glad to put a pointer into the "Other Sources of Information" section, or even at the top level. Just let me know...

Si tenga preguntas, quisa yo pueda ayudarte.

Forum: The Linux Kernel Hackers' Guide
Re: the KHG in spanish? (Jorge Alvarado Revatta)
Keywords: spanish translation
Date: Tue, 09 Sep 1997 03:18:41 GMT
From: <KernelJock>

```
Yo tengo poco experiencia en traduscas technicas, pero
necesito practicarla.  Hace mucho tiempo que lo he
practicado.

Si tenga preguntas, quisa yo pueda ayudarle.

ciao
```

Tengo una pregunta

Forum: The Linux Kernel Hackers' Guide
Re: the KHG in spanish? (Jorge Alvarado Revatta)
Re: Si tenga preguntas, quisa yo pueda ayudarte.
Keywords: spanish translation
Date: Thu, 04 Jun 1998 16:05:56 GMT
From: <riderghost@rocketmail.com>

puedo echar a un cerdo de Internet, desde mi computadora?

Español

Forum: The Linux Kernel Hackers' Guide
Re: the KHG in spanish? (Jorge Alvarado Revatta)
Re: Si tenga preguntas, quisa yo pueda ayudarte.
Keywords: spanish translation
Date: Thu, 09 Oct 1997 21:38:05 GMT
From: LL2 <jluis@itzcoatl.fi-c.unam.mx>

Si bajo la información.. ¿Cómo puedo utilizarla?

LLL

How to get a Memory snapshot ?

Forum: The Linux Kernel Hackers' Guide
Date: Sat, 24 May 1997 14:42:34 GMT
From: Manuel Porras Brand <guporras@calvin.univalle.edu.co>

Hi

In my course of Operating Systems my final consists of the following:

If i'm using Linux as OS and suddenly the power shuts down, how can i now what was in the system runnig at that moment (process, jobs, users, etc). Maybe getting a snapshot of the memory blocks where Linux puts this info. If this is the solution , how can i do it? If not, which one could it be ? Or where in the web can i find docs or any information about this ?

Thanks.

PS : Sorry for my English.

Messages

1. Why not to get a memory snapshot? *by Jukka Santala* NEW

Why not to get a memory snapshot?

Forum: The Linux Kernel Hackers' Guide
Re: How to get a Memory snapshot ? (Manuel Porras Brand)
Keywords: UPS memory-image
Date: Fri, 30 May 1997 21:37:28 GMT
From: Jukka Santala <e75644@uwasa.fi>

Altough I normally disagree with study assignments "done" thru Usenet etc. (check any resource on netiquette etc. :P) that assignment sounds weird enough to warrant a quick note.

That being that "technically", there's little you can do once the power goes down... the first warnign you get is, well, err, when the power goes down and the processor stops executing instructions ;) Incidentally (and luckily, for many) there's a thing called Uninterruptable Power Supply, or UPS for short, which can warn the operating-system once the power goes bad, and feed emergency power to the system until it has managed a controlled shut-down or the power goes up again.

Ofcourse, if you "simply" need to know _what_ the OS was doing when the power went down, that may be a bit of an overkill. This is how we finalyl get into slightly kernel-related stuff. One approach might be to let kernel log all program executions/forks into syslog thru syslogd - however, since all disk-access is cached this will be slightly out of date if the computer just suddenly gets switched off. A bit of clever coding might be used to avoid the cache in writing (Or does syslog already do that? Would make sense) or another solution might be to use battery-backed bubble-ram or something similiar with very short access times if one is worried about performance.

But shortly put, memory-images once the power actually goes down are out of question ;) However, if the issue is simply about kernel bug-tracking... well, that's another issue (and more appropriate for this place, I might add :P) indeed.

Why you would want to get a memory snapshot

Forum: The Linux Kernel Hackers' Guide
Re: How to get a Memory snapshot ? (Manuel Porras Brand)
Re: Why not to get a memory snapshot? (Jukka Santala)
Keywords: UPS memory-image, Power-Fail Interrupt, power-fail recovery
Date: Tue, 11 Aug 1998 17:07:18 GMT
From: Dave M. <DGMDGM@INAME.COM>

If I understand the question correctly, one reason you would want to get a memory snapshot before a power failure would be in the case where you are designing a system that does not have a UPS but does have a Power Fail Interrupt.

This is a concept that was used back in the mini-computer days. When the power supply sensed a drop in input voltage it would generate an interrupt that would notify the system that power was about to be lost. The power supply was designed to continue to provide power for a number of milliseconds (typically 60-120) after the loss of line power. This would give the OS just enough time to stash memory and the state of all registers in an image that could be retrieved upon power-up. When power was restored, this image could be used to restart the system where it left off. It could also be analyzed during the boot process and used to direct recovery operations upon restart.

The need for this type of power-fail restart may not be immediately obvious to a PC user but if you are using your computer to perform some type of machine control or instrumentation monitoring, where power-fail recovery is critical, then knowing what the OS was doing at the time of power loss is very important.

Dave M.

resources hard limits

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 22 May 1997 19:29:43 GMT
From: <castejon@kbw350.chem.yale.edu>

```
How the resources limits (filesize, cpu,etc) can be changed?
```

Messages

1. Setting resource limits *by Jukka Santala* NEW

Setting resource limits

Forum: The Linux Kernel Hackers' Guide
Re: resources hard limits
Keywords: getrlimit setrlimit limit ulimit hard resource limits
Date: Fri, 30 May 1997 21:59:26 GMT
From: Jukka Santala <e75644@uwasa.fi>

Do try man limit and man ulimit; well, my current installation has neither manpage, but at least ulimit's in /usr/bin by default. limit is a built-in command in some shells for the same purpose. As for soft/hard limits, ulimit -S vs. ulimit -H ought to do that. Incidentally, if you specify neither, both should get changed at the same time. csh is an excemption; here you need to use limit -h for hard limits. As for C programs, use getrlimit() and setrlimit() calls - but then you really ought to get the manpages, and I have no clue what this has to do with kernel hacking... *shakes head* ;)

How to invalidate a chache page

Forum: The Linux Kernel Hackers' Guide
Keywords: Chache Problem
Date: Wed, 21 May 1997 17:47:36 GMT
From: Gerhard Uttenthaler <u7x62bt@sun1.lrz-muenchen.de>

Hello everybody,

I have a problem with a char device driver, which I think is related to a chaching problem.

In my interrupt routine where I receive data from my device I have to handle a specific protocol depending on my device.

So I have to poll a specific address until a change at this address occurs. Something like this:

```
do{
  status = readb(0xc0000);
   timeout++;
}while(!(status & 0x80) && (timeout < 2000));
```

Sometimes I get timeouts, which can only occur if I read chache memory which is not up to date. The device is fast and works pretty well under DOS and Win311.

The situation is quite similiar on write access, where I must be sure to write directly to the device, not into the chache waiting for a flush.

The question is: Does anyone know how to invalidate the chache on my address or how to tell the chache not to chache my device at all???

Thank you very much!

G. Uttenthaler u7x62bt@sun1.lrz-muenchen.de

Messages

1. Read the rest of the KHG! *by Michael K. Johnson*

Read the rest of the KHG!

Forum: The Linux Kernel Hackers' Guide
Re: How to invalidate a chache page (Gerhard Uttenthaler)
Keywords: Cache Problem
Date: Sat, 14 Jun 1997 01:49:44 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

You clearly want to read The Linux Cache Flush Architecture

Where are the tunable parameters of the kernel?

Forum: The Linux Kernel Hackers' Guide
Date: Thu, 15 May 1997 08:10:29 GMT
From: <demiguel@robot3.cps.unizar.es>

I need to find the tunable parameters of the linux kernel, bue i cant find them (filesystems, processes, swapping,...)

I would like to know if there is any files such as mtune, or dtune is System V where you can change the main tunable parameters.

Thank you everybody!!

Messages

1. Kernel tunable parameters *by Jukka Santala*

Kernel tunable parameters

Forum: The Linux Kernel Hackers' Guide
Re: Where are the tunable parameters of the kernel?
Date: Thu, 15 May 1997 14:47:31 GMT
From: Jukka Santala <e75644@uwasa.fi>

Try out rdev (man rdev if you have it installed; must be on sunsite etc. if not). In short rdev sets root device, swapdev sets swapping, ramsize sets ramdisk-size, rootflags for root-fs mounting parameters and vidmode for ... well, you guessed it ;)

How can my device driver access data structures in user space?

Forum: The Linux Kernel Hackers' Guide
Keywords: device driver
Date: Tue, 06 May 1997 15:58:47 GMT
From: Stephan Theil <theil@zarm.uni-bremen.de>

My device driver needs to access data structs in user space. The functions get/put_user are the only way - I think - to transfer data between kernel and user space. But both function only allow manipulating of scalar types. How can I get/put data structs from/to user space???

Thanx!

Stephan

Forced Cast data type

Forum: The Linux Kernel Hackers' Guide
Re: How can my device driver access data structures in user space? (Stephan Theil)
Keywords: device driver
Date: Fri, 05 Jun 1998 02:45:35 GMT
From: Wang Ju <wangju@ics.ict.ac.cn>

Since the device driver know the type of data structure, you can cast the point to user space to the structure you needed.

Problem in doing RAW SOCKET Programming

Forum: The Linux Kernel Hackers' Guide
Keywords: Client server program using Raw sockets.
Date: Wed, 30 Apr 1997 15:39:27 GMT
From: anjali sharma <asharm@acadcomp.cmp.ilstu.edu>

I have to write a client server program using raw socket. I have written the code for client as well as server but when ever I run it my server hangs up. So I have to reboot the server. I think there is problem with my send and receive. I am sending the code for server. Hope you would be able to help me.

@@@@@@@@@@@@@@@@@@@@@@@@ code @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

#include <stdio.h>

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
```

u_short portbase = 0; long time();

```
#define qlen 6
#define protocol "raw"
#ifdef REALLY_RAW
#define FIX(x) htons(x)
#else
#define FIX(x) (x)
#endif

main(int argc, char **argv)
{
    int msock, ssock;
    int alen;
    char buf[] = "asdfgh";
    char recv_buffer[20];
    struct servent *pse;
    struct protoent *ppe;
    struct sockaddr_in dst;
    struct hostent *hp;
    struct ip *ip = (struct ip *)buf;
    struct icmp *icmp = (struct icmp *)(ip +1);

    int s, type, dstL;
    int q, bind1, lis;
    int sockopt;
    int on = 1, address;
    int offset;
    int sendbuff;
    int n;
    bzero((char *)&dst, sizeof(dst));
    dst.sin_family = AF_INET;

    dst.sin_port = 6000;
    ppe = getprotobyname("raw");
    setbuf(stdout,NULL);
    s = socket(AF_INET, SOCK_RAW, 0);
    printf("\n%d value of s in servsock",s);
    if (s < 0)
        printf("\nCann't creat socket");

    setbuf(stdout,NULL);

    sockopt = setsockopt(s, 0, IP_HDRINCL, &on, sizeof(on));
    printf("\n%d value of sockopt", sockopt);
    if (sockopt < 0)
          exit(0);

    if(( hp = gethostbyname(argv[1])) == NULL){
        if(ip->ip_dst.s_addr = inet_addr(argv[1]) == -1)
             printf("\nERROR: UNKNOWN HOST");
    }
    else
        bcopy(hp->h_addr_list[0], &ip->ip_dst.s_addr,                            hp-
>h_length);
    printf("\nSending to %s\n", inet_ntoa(ip->ip_dst));

    ip->ip_v = 4;
    fflush(stdin);
    ip->ip_hl = sizeof *ip >> 2;
    ip>ip_tos = 0;
    ip->ip_len = sizeof buf;
    ip->ip_id = htons(4321);
    ip->ip_off = 0;
    ip->ip_ttl = 255;
    ip->ip_p = 1;
    ip->ip_sum = 0;
    ip->ip_src.s_addr = 0;
    dst.sin_addr = ip->ip_dst;
    dst.sin_family = AF_INET;

    icmp->icmp_type = ICMP_ECHO;
    icmp->icmp_code = 0;
    sendbuff = sendto(s, buf, sizeof buf, 0, (struct sockaddr
*) &dst, sizeof dst);
        if(sendbuff < 0)
                printf(" ERROR sending ");
        if ( sendbuff != sizeof buf)
                printf("ERROR packet size");
        printf("\n buf is %s value of send is %d ", buf, sendbuff);

        dstL = sizeof dst;
        n = recvfrom(s, recv_buffer, sizeof(recv_buffer), 0,
         (struct sockaddr *) &dst,&dstL);
```

printf("recv buffer is%s value of n is %d\n", recv_buffer,n); close(s); exit(0); }
@@

Problem with ICMP echo-request/reply

Forum: The Linux Kernel Hackers' Guide
Re: Problem in doing RAW SOCKET Programming (anjali sharma)
Keywords: Client server program using Raw sockets.
Date: Thu, 09 Jul 1998 12:17:09 GMT
From: Raghavendra Bhat <raghu@swec.xko.dec.com>

```
I have the following program to simulate "ping". I am sending an ICMP echo
request to a specified host and expect the host to respond with ICMP echo
response. I understand from TCP/IP books that the kernel of the target host
will send the "echo response" as soon as it receives the "echo request". But
"recvfrom()" in my program waits forever since it doesn't receive any
"echo response". While "myping" is waiting in recvfrom(), if I do a
"ping" to the target host from my host, "myping" comes out successfully
with correct packet size also. This means that IF my current host receives
the "echo response", it is properly delivering the packet to my program. That
is,
there should not be any problem with my recvfrom(). But I am not able to make
out why my targethost's kernel is not responding to my "echo request". Could
it be because my "echo request" is not reaching the target machine or the
target machine is not able to identify the received packet as "echo request"?
Can anybody tell me the reason for this? What could be the possible reasons
and
what is the solution?

Thanks in advance,
Raghu
```

```
/* myping.c */
/*
 * This program simulates the "ping" program. But it doesn't bother about
 * checksum, unique sequence id, etc.
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netdb.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>

/* My own ICMP structure */
struct myicmp
```

```
{
    unsigned char icmp_type;
    unsigned char icmp_code;
    unsigned short icmp_cksum;
    unsigned short icmp_id1;
    unsigned short icmp_seq1;
};

/* argv[1] can be either hostname or IP address */
main(int argc, char *argv[])
{
  struct sockaddr_in address;
  u_char sendpack[32];
  struct myicmp *icp;
  int Nbytes, sock , status, buf_len;
  struct sockaddr_in    Current_Sockaddr;
  char buffer[100];
  struct hostent *host;

  if (argc != 2)
  {
        printf("usage : %s <hostname|IP address>\n", argv[0]);
        exit(1);
  }

  if ((host=gethostbyname((const char*)argv[1])) == NULL)
  {
        if ((address.sin_addr.s_addr = inet_addr(argv[1])) == -1)
        {
          printf("%s: unknown host\n", argv[1]);
          exit(1);
        }
  }
  else
  {
        bcopy(host->h_addr_list[0], &(address.sin_addr.s_addr),
host->h_length);
  }

  memset(sendpack, 0x0, sizeof(sendpack));
  memset(buffer, 0x0, sizeof(buffer));

  icp=(struct myicmp*)sendpack;

  icp->icmp_type=ICMP_ECHO;
  icp->icmp_code=0;
  icp->icmp_seq1=1; /* any abritrary sequence number */
  icp->icmp_id1=123; /* any arbitrary id */

  address.sin_family = AF_INET;

  /* 1 is for ICMP protocol : from /etc/protocols */
  sock = socket(AF_INET,SOCK_RAW, 1);

  buf_len = sizeof(buffer);
  Nbytes= sendto(sock,  (const void *)sendpack, sizeof(sendpack), 0,
        (struct sockaddr *)&address,sizeof(address));

  printf ("Data is sent to %s\n", inet_ntoa(address.sin_addr));
  printf ("No of bytes sent = %d\n", Nbytes);

  buf_len = sizeof(Current_Sockaddr );

  Nbytes= recvfrom(sock, buffer,  buf_len, 0,
              (struct sockaddr *)&Current_Sockaddr, &buf_len  );

  printf ("Data received from %s\n", inet_ntoa(Current_Sockaddr.sin_addr));

  /* get the sequence-id : First 20 bytes are for IP header :
     21t byte is the sequence-id */
  icp=(struct myicmp*)&buffer[20];
  printf("Received sequence id is %d\n", icp->icmp_seq1);

  printf ("No of bytes recd = %d\n", Nbytes);
}
```

Tunable Kernel Parameters?

Forum: The Linux Kernel Hackers' Guide
Keywords: kernel tuning, file handles, Bus error
Date: Sat, 19 Apr 1997 13:07:40 GMT
From: <dennyf@bmn.net>

In a recent comp.os.linux. article I saw a reference to tunable kernel parameters, which I think may help me with a problem I'm having with expireover on a Debian 1.2, Inn 1.4 unoff4 news server.

The gist of the article was that the other fellow's problem may have been caused by running out of file handles, and that file handles were a compile time option.

I'm runnming 2.0.27, 64mb ram, raid 0 on three 2gb drives for /var/spool/news.

News expires fine, but expireover causes errors like:

no memory for named no memory for expireover

then finally:

Bus error

At this point serveral drivers usually die or go zombie and the system has to be restarted.

Monitoring the system with top while expireover is running shows plenty of available memory, with hardly any swap being used out of the 64mb swap space. By available memory, I mean there is about 30 odd mb in the buffer cache.

I've been looking for some docs on tuning kernel parameters like file handles, and can't seem to find it. Can someone please point me in the right direction?

Any other suggestions would be greatly appreciated.

Thanks,

Denny Fox, dennyf@bmn.net

Messages

1. sysctl in Linux *by Jukka Santala*

sysctl in Linux

Forum: The Linux Kernel Hackers' Guide
Re: Tunable Kernel Parameters?
Keywords: kernel tuning, file handles, Bus error
Date: Thu, 15 May 1997 15:14:33 GMT
From: Jukka Santala <e75644@uwasa.fi>

The util you're looking for is sysctl (Or a system-call by the same name). However, as far as I know this isn't quite fully implemented in Linux as of yet (I just saw it on a "wishlist" for 2.2 kernels). Certainly I haven't been able to find anything meaningful on sysctl in Linux, so perhaps that post you was referring to was abotu the *BSD's which I seem to remember use sysctl rather heavily. Ofcourse there's the option to choose whether or not to compile sysctl in to the kernel at least on 2.1.37; if anybody knows for sure if working sysctl utils can be had anywhere, drop a line.

However, on the grand scale, I don't think sysctl would do it- Linux (Ok, again, at least in 2.1.37) comes with all defaults compiled to 1024fd's, and changing that would require at least increasing _FD_SETSIZE, NR_FILE and NR_OPEN. You'd have to change these in the kernel headers and recompile everything to get any changes of it working anyway.

On the grand scale, though, I somehow doubt this is your problem - running out of file descriptors rarely results in kernel crashes.

Increasing number of files in system

Forum: The Linux Kernel Hackers' Guide
Re: Tunable Kernel Parameters?
Keywords: kernel tuning, file handles, Bus error
Date: Sat, 25 Oct 1997 11:19:38 GMT
From: Simon Cooper <simon.cooper@snc-net.demon.co.uk>

Increasing number of open file handles in kernel.

I have successfully increased the number of files on my system (Linux 2.0.30) by editing the following file:

/usr/src/linux/include/linux/fs.h

I increased NR_OPEN to 1024, NB. Don't make it any higher than this as it will break other code in the kernel and your new kernel wont boot !

I also increased NR_INODE to 5000, and NR_FILE to 4096.

I then rebuilt the Kernel (and modules for good measure) and installed the new image.

Note: Its always a good idea to have your original Kernel image available. I installed a boot block in my MBR using lilo, and added an entry for my original kernel I named vmlinux.safe.

For my root system on /dev/hda2 I added the following lines to my existing /etc/lilo.conf

image=/vmlinux.safe image=linuxS root=/dev/hda2

Messages

1. Increasing number of open files parameter *by Simon Cooper* NEW

Increasing number of open files parameter

Forum: The Linux Kernel Hackers' Guide
Re: Tunable Kernel Parameters?
Re: Increasing number of files in system (Simon Cooper)
Keywords: kernel tuning, file handles, Bus error
Date: Sat, 25 Oct 1997 11:25:44 GMT
From: Simon Cooper <simon.cooper@snc-net.demon.co.uk>

Sorry for the typo, my (Linux loader) lilo.conf entries are:

image=/vmlinux.safe [Newline] label=LinuxS [Newline] root=/dev/hda2 [Newline]

Setting and getting kernel vars

Forum: The Linux Kernel Hackers' Guide
Re: Tunable Kernel Parameters?
Re: sysctl in Linux (Jukka Santala)
Keywords: kernel vars
Date: Thu, 16 Jul 1998 22:57:00 GMT
From: <kbrown@csuhayward.edu>

If sysctl is not implemented properly yet, then what is a good method for setting and getting kernel variables?

I used sysctl to access some varibale I added under NetBSD and I was hoping to do the same now that I'm using Linux instead.

Has anyone done this successfully?

ELF matters

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 15 Apr 1997 09:09:51 GMT
From: Carlos Munoz <cmunoz@dsic.upv.es>

Hi again!!

Something bizarre is happening in my Linux 1.2.13:

I have created a silly program that executes just after a fork() for the init is done in the function start_kernel():

```
.........
if (!fork())
        init();
..........
```

void init() { ..... THIS IS MY CHANGE:

```
        if (!fork())
                execve("silly", NULL, NULL);
.....
}
```

If the format of the executable silly is ELF then everything works okay, but if I use the old a.out (compiled with gcc silly.c -o silly -b i486-linuxaout) the program doesn't execute at all and I cannot find it anywhere (top and ps -x fail to do it).

Am I going nuts or are Linux developers trying to move us to ELF?

Secondly, I would like to disable demand loading in ELF for some real-time tasks which I don't want them to produce page faults while they're actually running. Is it possible?

Thirdly, can you tell me where can I find information about ELF internals? I get sick when I try to understand the function load_elf_binary!

Thanks a lot in advance!!

CARLOS AKA SLACKER

Messages

1. Information about ELF Internals *by Pat Ekman*

Information about ELF Internals

Forum: The Linux Kernel Hackers' Guide
Re: ELF matters (Carlos Munoz)
Keywords: ELF format
Date: Thu, 08 May 1997 18:52:42 GMT
From: Pat Ekman <ekman@cs.byu.edu>

Documentation for ELF internals can be found at sunsite.unc.edu (or a mirror), in /pub/Linux/GCC/ELF.doc.tar.gz.

Droping Packets

Forum: The Linux Kernel Hackers' Guide
Keywords: packets
Date: Mon, 31 Mar 1997 01:27:55 GMT
From: Charles Barrasso <charles@blitz.com>

Is there any way to tell the kernel to drop packets from certian hosts. Like drop packets from all the hosts in a file? If not I would be verry greatfull to the person who writes that program if it could be written.

charles

Messages

1. [Selectively] Droping Packets *by Jose R. cordones*

[Selectively] Droping Packets

Forum: The Linux Kernel Hackers' Guide
Re: Droping Packets (Charles Barrasso)
Keywords: ICMP packet Firewall
Date: Fri, 11 Apr 1997 19:38:11 GMT
From: Jose R. cordones <cord2403@cslab.engr.ccny.cuny.edu>

Sure you can.

Look into compiling support for "Firewall," etc. code in the kernel. Then you get the "ipfwadm" (IP Firewall Admin.) package (available wherever fine free software is sold^H^H^H^Hgiven away.)

You then add rules on which traffic to allow your host to accept, which to reject (implies that the host attempting a connection receives feedback in the form of a ICMP error message) and which to ignore (no ICMP error sent.)

If such a machine is additionally forwarding traffic between several networks, then the marketing people call this a "Firewall." But you can also be using it just to protect the host itself.

There are other solutions that are not so low-level such as the tcpd daemon and configuring **well** any daemon/service you run on your machine, something to which no firewall can be a substitute.

Cheers,
José R. Cordones <cord2403@cslab.engr.ccny.cuny.edu>
http://www.engr.ccny.cuny.edu/~cordones

The /proc/profile

Forum: The Linux Kernel Hackers' Guide
Keywords: /proc/profile, kernel hacking
Date: Mon, 31 Mar 1997 00:15:34 GMT
From: Charles Barrasso <charles@blitz.com>

I recently upgraded the kernel to 2.0.29 and included kernel hacking support. Now I have a /proc/profile file that I want to read. Supposedly it contains info on the kernel. I know I need to have softwhere to read what is in the file. where would I get that? Also what else can I do now that I have kernel Hacking support?

thanks charles

Messages

1. readprofile systool *by Jukka Santala*

readprofile systool

Forum: The Linux Kernel Hackers' Guide
Re: The /proc/profile (Charles Barrasso)
Keywords: /proc/profile, kernel hacking
Date: Thu, 15 May 1997 15:24:43 GMT
From: Jukka Santala <e75644@uwasa.fi>

RTFM ;) If you enter '?' at the prompt (or menu-choice) about kernel-hacking support, it explains all it does for now is allow you to get profiles on where exactly the kernel is spending it's time into the /proc - filesystem. Further it's noted that to read that information you need a tool called readprofile, which is available from ftp://sunsite.unc.edu/pub/Linux/kernel/readprofile-2.0.tar.gz (What a surprise;). Using the mirror-site closest to you is preferable.

It's actually pretty useful for profiling the kernel for optimization purposes, however worth remembering is that since the kernel-source is heavily optimized, there's no direct connection between the results and the actual code in the way one could expect (Ie. a lot of functions get inlined etc; if the entry you're looking for doesn't show even on readprofile -a, it's probably made part of the calling function(s)).

Can you block or ignore ICMP packets?

Forum: The Linux Kernel Hackers' Guide
Keywords: ICMP ping Internet echo flood
Date: Sat, 22 Mar 1997 02:46:28 GMT
From: <HackOps@nutnet.com>

I would like to know if you could block or ignore ICMP packets. If there is no way to block reception, then is there a way to prevent the kernel from replying? The reason is that I want to prevent other people from abusing the "ping" command and flooding me with ICMP packets. In particular, I want to block only when there is an excessive amount of packets being recieved. (ie. 25+ in a 10 sec period)

Messages

4. ICMP send rate limit / ignoring *by Jukka Santala*
    1. Omission in earlier rate-limit... *by Jukka Santala*
    -> Patch worked... *by Jukka Santala*
3. Using ipfwadm *by Charles Barrasso*
1. Icmp.c and kernal ping replies *by Don Thomas*

ICMP send rate limit / ignoring

Forum: The Linux Kernel Hackers' Guide
Re: Can you block or ignore ICMP packets?
Keywords: ICMP ping Internet echo flood
Date: Thu, 15 May 1997 14:30:48 GMT
From: Jukka Santala <e75644@uwasa.fi>

While adding that #define CONFIG_IP_IGNORE_ECHO_REQUESTS into linux/net/ipv4/icmp.c will work fine for now, I'd suggest putting it into the configuration-headers so it doesn't tangle up with further patches, or, should that define later move into different file(s), lose it's efficiency. This is also the easiest way to make sure all future versions of the kernel you compile get that setting defined.

Unfortunately, I'm not quite sure where you can stick it without messing up the kernel autoconfig ;) If anybody has any input on this, it would be most welcome.

Meanwhile, if you're worried that ignoring _all_ echo-requests may be a bit too rough move, there's a way to make the kernel ignore them selectively. This is available at least in the 2.1.X series, unfortunately I don't know if it's elsewhere.

While browsing the net earlier I came upon a site with cross- referenced kernel sources for all major Linux distributions, so I thought I'd check it out from there, but naturally I didn't save the URL anywhere, typical, so if somebody knows that site I'd appreciate to know too ;)

But back on track... so how do you make that selective ignore? Simple, first make sure CONFIG_NO_ICMP_LIMIT _isn't_ defined - don't worry how, it won't be ;) Next, in linux/net/ipv4/icmp.c go to the end of the file where there is a table of ICMP definitions - the first entry is after /* ECHO REPLY (0) */ This is, incidentally, what you need to change. Change the NULL on that line to &xrl_generic. So what does that do? I suggest you look at the source and try to figure that out yourself - it's not that hard, and allows you better diddle with it. (However, the limit-code seems pretty inefficient to me, and is no use against spoofed ICMP-floods, so I suggest relying on it with caution)

Messages

1. Omission in earlier rate-limit... *by* *Jukka Santala*
-> Patch worked... *by* *Jukka Santala*

Omission in earlier rate-limit...

Forum: The Linux Kernel Hackers' Guide
Re: Can you block or ignore ICMP packets?
Re: ICMP send rate limit / ignoring (Jukka Santala)
Keywords: ICMP ping Internet echo flood
Date: Thu, 15 May 1997 22:44:33 GMT
From: Jukka Santala <e75644@uwasa.fi>

```
Oops, what a mistake. I missed the fact that icmp_send()
isn't actually used for replying to ICMP_ECHO_REQUEST's etc.
so no matter how you change the table in question, none
of the replies are going to be limited... so what you need
to do is add a call to the check in question to icmp_reply()
as well, which is something that can already be called real
kernel hacking. Here's how I'm doing it; however...
1) I haven't yet rebooted with this code... wish me luck ;)
2) Am I missing something? ping -f and ping -l get mostly ignored
Here's the bit of code, in icmp_reply() right at the beginning (after local varable
definitions) :
  #ifndef CONFIG_NO_ICMP_LIMIT
          if(!xrlim_allow(icmp_param->icmph.type, skb->nh.iph->saddr))
                 return;
  #endif
I'll let you know how my tests with the thing proceed ;)
(Sorry for bad formatting, I managed to break my PPP thingy playing around with
filedescriptors, it seems, and this remote lynx doesn't quite handle text-fields
properly, it seems... :P)
```

Messages

1. Patch worked... *by Jukka Santala*

Patch worked...

Forum: The Linux Kernel Hackers' Guide
Re: Can you block or ignore ICMP packets?
Re: ICMP send rate limit / ignoring (Jukka Santala)
Re: Omission in earlier rate-limit... (Jukka Santala)
Keywords: ICMP ping Internet echo flood
Date: Thu, 15 May 1997 23:04:28 GMT
From: Jukka Santala <e75644@uwasa.fi>

Just a quick note to report success on that patch ;) Now, doing ping -l 1000 -c 1000 host (Not suggested to test willy-nilly; very effective flood where supported;) only replies to 30 first ping-packets, ignoring the rest (Before the patch I got about 180 replies - does similiar code to tune already exist elsewhere?). Another ping-flood right before earns only two replies, though (Is this correct?). A normal ping with one-second delay goes thu with 0% packet loss. I'd be interested to hear results if anybody dares to try this patching out on a "real" configuration. (I have a very limited PPP account, basically conducting tests over local loopback - oh, and by the way, that PPP breakage wasn't because of my filehandle playing, it was because I had removed resolv.conf for who knows what reasons... increasing fd's up to 4k seem to have worked without problem at least for now;)

Using ipfwadm

Forum: The Linux Kernel Hackers' Guide
Re: Can you block or ignore ICMP packets?
Keywords: ICMP ping Internet echo flood
Date: Sun, 11 May 1997 21:50:25 GMT
From: Charles Barrasso <charles@blitz.com>

If you compile the kernel with FireWall support then you could do:

ipfwadm -I -P icmp -a reject(or deny)

that would make it so your computer wouldn't reply to the pings from any host.

But lets say that you wanted to be able to be pinged by brigia.blitz.com but not by anyone else well then you would

ipfwadm -I -P icmp -a accept -S brigia.blitz.com ipfwadm -I -P icmp -a reject(or deny)

make sure you put the accepts first then the deny's or rejects.

Hope this wasn't too confusing,

Charles

Icmp.c and kernal ping replies

Forum: The Linux Kernel Hackers' Guide
Re: Can you block or ignore ICMP packets?
Keywords: ICMP ping Internet echo flood
Date: Sat, 22 Mar 1997 15:55:26 GMT
From: Don Thomas <dgt@multiline.com.au>

Hi,

I'm not sure if you can ignore ICMP requests, but I have been able to modify icmp.c to stop the kernal replying to ping requests. This halves the amount of traffic if you are flood pinged, plus the person pinging you, may well believe that you are down because of the absence of replies. I added this to icmp.c in the /usr/src/linux/net/ipv4 directory, and then re-compiled. Seems to work okay on 2.0.28.

```
#define CONFIG_IP_IGNORE_ECHO_REQUESTS
```

Regards,

Don

ipfwadm configuration utility

Forum: The Linux Kernel Hackers' Guide
Re: Can you block or ignore ICMP packets?
Re: Using ipfwadm (Charles Barrasso)
Keywords: IPFWADM
Date: Mon, 20 Jul 1998 22:44:31 GMT
From: Sonny Parlin <sonny@ejj.net>

FREE SOFTWARE

I have written an ipfwadm GUI configuration utility. It's GUI via Netscape... it creates a shell script to be used as a firewall based on the criteria you choose during the configuration. It can also install the firewall rules, uninstall them, check firewall status, and watch network traffic from the masqueraded connections. If anyone is interested in this, Check out:

http://www.ejj.net/~sonny/fwconfig/fwconfig.html

Sonny

encaps documentation

Forum: The Linux Kernel Hackers' Guide
Keywords: encaps doc docs
Date: Thu, 13 Mar 1997 11:28:50 GMT
From: Kuang-chun Cheng <kccheng@hycppc01.fnal.gov>

Hi,

```
   Could someone tell me where can find the documentation
or man pages of command 'encaps' which used in elf kernel
building.  I knew it came from binutils but just can't find
any word about in binutils' info page.  Thanks.

                      Kuang-chun Cheng
                      kccheng@hycppc01.fnal.gov
```

Mounting Caldrea OpenDOS formatted fs's

Forum: The Linux Kernel Hackers' Guide
Keywords: filesystems Caldera OpenDOS mount
Date: Mon, 17 Feb 1997 18:17:19 GMT
From: Trey Childs <tchilds@paraengr.com>

Anyone have a patch to allow mounting an OpenDOS formatted partition as 'msdos'? I think it's complaining about a signature that it expects. If so, that sounds easy. I took a look in the filesystems code and didn't see anything like it.

Thanks, Trey Childs

finding the address that caused a SIGSEGV.

Forum: The Linux Kernel Hackers' Guide
Keywords: SIGSEGV, signal, handler
Date: Sun, 09 Feb 1997 08:46:09 GMT
From: Ben Shelef <meekg@pobox.com>

one way, of course, is to backtrack up the stack, beyond the sigreturn frame. for that, i'd need the stack layout. but this address should also be available from the kernel - does anyone know where to find it? thanks, ben.

sti() called too late.

Forum: The Linux Kernel Hackers' Guide
Keywords: cli() IRQ rtc realtime timer lost
Date: Sat, 18 Jan 1997 11:00:12 GMT
From: Erik Thiele <erik@unterland.de>

Hello

i read the /dev/rtc (Realtime clock) driver, mutated it and wrote my own little module, that get's called 8192 times per second via SA_INTERRUPT (highest possible priority) IRQ 8. ok. if harddisk interrupts occur, some timer interrupts get lost. this is also mentioned in the original rtc driver. this happens, because cli is "active" too long. WHY doesn't the harddisk driver do a sti() as early as possible ???

i don't think this is good code.

:)

byebye Erik

Messages

1. sti() called too late. *by Gadi Oxman*

sti() called too late.

Forum: The Linux Kernel Hackers' Guide
Re: sti() called too late. (Erik Thiele)
Keywords: cli() IRQ rtc realtime timer lost
Date: Fri, 24 Jan 1997 19:07:40 GMT
From: Gadi Oxman <gadio@netvision.net.il>

```
Unfortunately, some combinations of chipsets, IDE
interfaces and drives will not work reliably when
interrupts are allowed to occur while we are servicing
a disk interrupt.

The IDE driver can optionally unmask interrupts during
the data transfer, by using "hdparm -u1 /dev/hdx".

This is a typical problem; on the one hand, we want to
achieve the best possible performance, but once we do
this, we might find out that we are no longer compatible
with marginal hardware which could have worked if only
we didn't try to push it so hard.

Gadi Oxman
gadio@netvision.net.il
```

Module Development Info?

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 18 Dec 1996 04:12:10 GMT
From: Mark S. Mathews <mark@absoval.com>

Is there a 'single source' for information about developing modules?

Thanks in Advance! MSM

Messages

1. Needed here too *by ajay*
2. Help needed here too! *by ajay*

Needed here too

Forum: The Linux Kernel Hackers' Guide
Re: Module Development Info? (Mark S. Mathews)
Date: Tue, 11 Feb 1997 09:48:15 GMT
From: ajay <aftek@giaspn01.vsnl.net.in>

I also need information about module development and the only help I can get are from the module programs given with the source. Also need kernel functions and I don't have any info other than the ksyms file. Thanx !!

Help needed here too!

Forum: The Linux Kernel Hackers' Guide
Re: Module Development Info? (Mark S. Mathews)
Date: Tue, 11 Feb 1997 09:52:25 GMT
From: ajay <aftek@giaspn01.vsnl.net.in>

I also need information about module development and the only help I can get are from the module programs given with the source. Also need kernel functions and I don't have any info other than the ksyms file. Thanx !!

Need quicker timer than 100 ms in kernel-module

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 06 Dec 1996 19:14:31 GMT
From: Erik Thiele <erik@unterland.de>

hello

i am just writing a kernel module (2.0.27kernel), that shall turn engines.. :) you know, those electronic engines that have 4 statuses, and by selecting 1 2 3 4 1 2 3 4 etc. they turn. i just don't know the english word... :)

i use a little external hardware that needs to be triggered in order to let the engine perform a step. the module uses an add_timer(...) to be called 100 times per second. this means 50 pulses per second. this means 50 steps of the engine per second. this is not enough. :-(so. as i read above there are many real-time approaches to linux, but they aren't yet really implemented ??? :) well somebody told me about /dev/rtc i could use this one. but i still want to use a kernel module for my purpose, so:

Is it possible to use open() write() read() close() ioctl() in a kernel-module or aren't those libc functions overloaded? (i want to use /dev/rtc IN my kernel module)

byebye&thanks :) Erik

Messages

1. 10 ms timer patch *by Reinhold J. Gerharz*
-> Please send me the patch *by Jin Hwang*

10 ms timer patch

Forum: The Linux Kernel Hackers' Guide
Re: Need quicker timer than 100 ms in kernel-module (Erik Thiele)
Keywords: TIMER
Date: Thu, 09 Jan 1997 03:55:29 GMT
From: Reinhold J. Gerharz <rgerharz@erols.com>

The "little engines" are call stepper-motors.

I have a kernel patch that allows a module to insert a high-priority hook function on the timer interrupt. It's very simple, actually. I can email it, as I don't have a web or ftp site of my own.

Messages

1. Please send me the patch *by Jin Hwang*

Please send me the patch

Forum: The Linux Kernel Hackers' Guide
Re: Need quicker timer than 100 ms in kernel-module (Erik Thiele)
Re: 10 ms timer patch (Reinhold J. Gerharz)
Keywords: TIMER
Date: Fri, 10 Jan 1997 22:43:18 GMT
From: Jin Hwang <hwang@ilt.com>

I have been looking for a way to use timer interrupt (30 msec) in module. I would appreciate if you send me that patch or tell me how to do it. I am using kernel 1.2.3. my e-mail address is hwang@ilt.com

---

please send me 10 ms timer patch

Forum: The Linux Kernel Hackers' Guide
Re: Need quicker timer than 100 ms in kernel-module (Erik Thiele)
Re: 10 ms timer patch (Reinhold J. Gerharz)
Keywords: TIMER
Date: Sat, 29 Nov 1997 15:08:26 GMT
From: Tolga Ayav <enko@raksnet.com.tr>

I am working on PC based control systems and I really need to use timer interrupt as fast as possible. I would be pleased if you could send me your patch. Thank you.

UTIME: Microsecond Resolution Timers

Forum: The Linux Kernel Hackers' Guide
Re: Need quicker timer than 100 ms in kernel-module (Erik Thiele)
Re: 10 ms timer patch (Reinhold J. Gerharz)
Re: Please send me the patch (Jin Hwang)
Keywords: TIMER Microsecond Resolution
Date: Mon, 13 Oct 1997 12:01:30 GMT
From: <BalajiSrinivasan>

We at the University of Kansas have been working on providing facility to add microsecond resolution timers to linux. Please see http://hegel.ittc.ukans.edu/projects/utime for more details.
Balaji

Need help with finding the linked list of unacked sk_buffs in TCP

Forum: The Linux Kernel Hackers' Guide
Keywords: sk_buff linked list
Date: Thu, 05 Dec 1996 02:40:41 GMT
From: Vijay Gupta <vijay@crhc.uiuc.edu>

Hi,

```
        I have been trying to build a protocol
similar to TCP within the Linux kernel for a
mobile computing environment.

        For that, I need to look at the list of
unacknowledged sk_buffs in order to change their
headers so that I can send them through a new
interface.

        I have tried sk->send_head, sk->send_tail
but without any success. In spite of there being
around 36-37 unacknowledged segments, there is
still nothing which I get by following the next
or prev pointers of send_head and send_tail.

        Can anyone help me tell which other place
I can look into ?

        Any help will be greatly appreciated. If
you can also cc the reply to vsgupta@cs.uiuc.edu,
then that would be great.
```

Thanks a lot, Vijay

❓ Partition Type

Forum: The Linux Kernel Hackers' Guide
Date: Tue, 12 Nov 1996 21:46:59 GMT
From: Suman Ball <sb241@columbia.edu>

Does anyone know which file in the kernel source contains partition type information? It appears that windows95 has a different partion type number for extended and logical partitions, so I need to add it to see whether I can read them. Thanks, Suman.

New document on exception handling

Forum: The Linux Kernel Hackers' Guide
Date: Mon, 11 Nov 1996 23:16:40 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Joerg Pommnitz wrote a nice document on Kernel-Level Exception Handling which he posted to the linux-kernel@vger.rutgers.edu list, and which he has graciously allowed me to format in HTML and include here in the KHG.

Thanks, Joerg!

How to make paralelism in to the kernel?

Forum: The Linux Kernel Hackers' Guide
Date: Wed, 30 Oct 1996 06:28:27 GMT
From: Delian Dlechev <delian@wfpa.acad.bg>

I need to know how to make paralel programs in to the kernel. May be I need to make modules? Or any process like kerneld and nfsiod?

readv/writev & other sock funcs

Forum: The Linux Kernel Hackers' Guide
Keywords: socket programming readv
Date: Mon, 21 Oct 1996 19:58:50 GMT
From: Dave Wreski <dwreski@ultrix.ramapo.edu>

I'm having a problem writing code to make use of readv() and the iovec struct. I'm pretty sure I'm doing it correctly, as I have spent countless hours troubleshooting (seasoned newbie here)

Should I post to linux-c-programming, or is it ok to ask kernel/C/socket questions here?

I'm using 2.0.20 or so, and I have one writev() that writes two iovec structs to a socket. The readv() on the other end requires two reads (well, readv's), to gather the data, and it doesn't seem to even be placed back together in my header properly.

Any ideas on pointers, or directions? I have most of Stevens' books, and have scoured them. They seem to be more interested in fd passing, which I don't need. (should I be using sendmsg()/recvmsg()?)

Thanks so much, Dave Wreski dwreski@ultrix.ramapo.edu

I'd like to see the scheduler chapter

Forum: The Linux Kernel Hackers' Guide
Date: Fri, 13 Sep 1996 02:29:56 GMT
From: Tim Bird <tbird@caldera.com>

The alpha 0.6 KHG included a chapter on the scheduler.

I suppose it's mostly out of date, but it would be nice to have it posted so we could start updating it.

I apologize in advance if it's here somewhere and I just didn't see it.

Messages

1. Untitled *by Vijay Gupta*
3. Go ahead! *by Michael K. Johnson*

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: I'd like to see the scheduler chapter (Tim Bird)
Date: Fri, 13 Sep 1996 09:19:00 GMT
From: Vijay Gupta <vijay@crhc.uiuc.edu>

```
You can refer to pages 46-49 of
Linux Kernel Internals (English version).
Authors M. Beck, H. Bohme, M. Dziadzka et. al.
Publisher Addison Wesley
Year 1996

Hope this helps,
Vijay
```

Go ahead!

Forum: The Linux Kernel Hackers' Guide
Re: I'd like to see the scheduler chapter (Tim Bird)
Date: Sun, 29 Sep 1996 21:11:38 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

If someone will research the topic sufficiently and write up a basic docoument, I'll put it up as a base document to which to attach comments and questions.

Unable to access KHG, port 8080 giving problem.

Forum: The Linux Kernel Hackers' Guide
Keywords: KHG port 80
Date: Sun, 18 Aug 1996 22:11:48 GMT
From: Srihari Nelakuditi <srihari@cs.umn.edu>

Hi,

I am not able to access KHG from my work place. I guess packets with port number greater than 1024 are filtered. So could you please suggest a mirror KHG site which serves on the regular port 80.

Thanks a Lot,

Srihari.

Messages

1. Get a proxy *by Michael K. Johnson*

Get a proxy

Forum: The Linux Kernel Hackers' Guide
Re: Unable to access KHG, port 8080 giving problem. (Srihari Nelakuditi)
Keywords: KHG port 80
Date: Thu, 29 Aug 1996 19:47:55 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Either have your workplace add a WWW proxy, or access the KHG from home.

proc fs docs?

Forum: The Linux Kernel Hackers' Guide
Keywords: proc
Date: Thu, 15 Aug 1996 07:53:40 GMT
From: David Woodruff <david.woodruff@tellurian.com.au>

Anyone know where to get information on how to write programs which use the proc fs? Or should I find and use sample code?

ta muchly, dave

Examples code as documentation

Forum: The Linux Kernel Hackers' Guide
Re: proc fs docs? (David Woodruff)
Keywords: proc
Date: Sun, 07 Dec 1997 02:43:13 GMT
From: Jeremy Impson <jdimpson@syr.edu>

I know this question was asked over two years ago, but I have a partial answer...

At http://web.syr.edu/~jdimpson/proj/fib-0.1.tgz you can get the source code to a loadable module that creates a file called /proc/fib. Writing a number to it will seed it, reading from it will read the the fibonacci value of that seed.

I wrote it as an exercise in understanding how to read and write to /proc files. Reade http://web.syr.edu/~jdimpson/proj/fib-0.1/README for more details.

--Jeremy

What is SOCK_RAW and how do I use it?

Forum: The Linux Kernel Hackers' Guide
Keywords: SOCK_RAW socket
Date: Tue, 06 Aug 1996 20:02:18 GMT
From: arkane <cat@iol.unh.edu>

What is SOCK_RAW type of socket. I know it requires root access but I don't know how to use it or what it does.

TIA

Messages

1. What raw sockets are for. *by Cameron MacKinnon*

What raw sockets are for.

Forum: The Linux Kernel Hackers' Guide
Re: What is SOCK_RAW and how do I use it? (arkane)
Keywords: SOCK_RAW socket
Date: Thu, 08 Aug 1996 14:59:11 GMT
From: Cameron MacKinnon <mackin@interlog.com>

Well, there are several types of sockets: TCP and UDP go over the wire formatted as TCP or UDP packets, unix-domain sockets don't generally go over the wire (they're used for interprocess communication). These are some of the built-in socket types that the kernel understands (i.e. it will handle the connection management stuff at the front of each of these packet types). Raw sockets are used to generate/receive packets of a type that the kernel doesn't explicitly support.

An easy example that you're probably familiar with is PING. Ping works by sending out an ICMP (internet control message protocol - another IP protocol distinct from TCP or UDP) echo packet. The kernel has built-in code to respond to echo/ping packets; it has to in order to comply with the TCP/IP spec. It doesn't have code to generate these packets, because it isn't required. So, rather than create another system call with associated code in the kernel to accomplish this, the "ping packet generator" is a program in user space. It formats an ICMP echo packet and sends it out over a SOCK_RAW, waiting for a response. That's why ping runs as set-uid root.

Linux kernel debugging

Forum: The Linux Kernel Hackers' Guide
Keywords: Linux kernel debugging
Date: Sat, 29 Jun 1996 09:52:11 GMT
From: <yylai@hk.net>

Does anyone get experiences, standard methods/setup, tips etc. in debugging the Linux kernel ?¡@I think it is a great topic that can be added to the Linux Hackers' guide.

Jeremy Y.Y.Lai

Messages

1. Device debugging *by alombard©iiic.ethz.ch*
2. GDB for Linux *by David Grothe*
    1. gdb debugging of kernel now available *by David Grothe*
    2. Another kernel debugging tool *by David Hinds*

Device debugging

Forum: The Linux Kernel Hackers' Guide
Re: Linux kernel debugging
Keywords: Linux kernel debugging
Date: Mon, 19 Aug 1996 10:12:27 GMT
From: <alombard©iiic.ethz.ch>

I have the same problem. I need to debug a network driver, but I can't figure out how to do it. It would be nice if I could make it write a kind of log file. Is that possible?

GDB for Linux

Forum: The Linux Kernel Hackers' Guide
Re: Linux kernel debugging
Keywords: Linux kernel debugging
Date: Fri, 20 Sep 1996 16:26:04 GMT
From: David Grothe <dave@gcom.com>

I have GDB running between two Linux boxes with a serial interface cable between them. I can set breakpoints, single step and do source level debugging in the kernel from one machine to the other.

Unfortunately I have not been able to make this generally available for two reasons. 1) I have been unable to do any Linux work at all for several months due to other pressing needs. 2) I am a beginner at the Linux kernel and toolsets and do not yet know how to use the "patch" facility.

I hope to be able to get this info out there before too long.

Messages

1. gdb debugging of kernel now available *by David Grothe*
2. Another kernel debugging tool *by David Hinds*

gdb debugging of kernel now available

Forum: The Linux Kernel Hackers' Guide
Re: Linux kernel debugging
Re: GDB for Linux (David Grothe)
Keywords: Linux kernel debugging
Date: Mon, 09 Dec 1996 23:40:07 GMT
From: David Grothe <dave@gcom.com>

A package that implements kernel debugging between two machines using gdb is now available.

Connect to ftp.gcom.com and download /pub/linux/src/gdbstub/gdbstub-2.0.24.tar.gz. This package is intended for the 2.0.24 kernel but will fit pretty easily into other (older) kernels as well.

Another kernel debugging tool

Forum: The Linux Kernel Hackers' Guide
Re: Linux kernel debugging
Re: GDB for Linux (David Grothe)
Keywords: Linux kernel debugging
Date: Fri, 20 Dec 1996 21:04:49 GMT
From: David Hinds <dhinds@hyper.stanford.edu>

I wrote a tool that lets you run gdb on the same system as the kernel you're debugging. It supports viewing and modifying kernel data structures, viewing stack traces for processes in the kernel, interpreting trap reports, and calling kernel functions. It isn't as flexible as a remote debugger; in particular, there are no breakpoints. But I've still found it to be very useful, and if you don't have a spare system to use for remote debugging, it is the next best thing.

You can find the "kdebug" package at <ftp://hyper.stanford.edu/pub/pcmcia/extras/kdebug-1.6.tar.gz>.

Kernel debugging with breakpoints

Forum: The Linux Kernel Hackers' Guide
Re: Linux kernel debugging
Re: GDB for Linux (David Grothe)
Re: Another kernel debugging tool (David Hinds)
Keywords: Linux kernel debugging
Date: Wed, 17 Sep 1997 11:59:33 GMT
From: Keith Owens <kaos@ocs.com.au>

John Heidemann <johnh@isi.edu> modified kdebug to support single-stepping and breakpointing, calling it xkdebug. In turn I have upgraded John's code from 1.3.30 to 2.1.55 and added code to detect some deadlock conditions and automatically avoid them.

ftp://ftp.ocs.com.au/pub/xkdebug-2.1.55.tgz

Need help for debugging

Forum: The Linux Kernel Hackers' Guide
Re: Linux kernel debugging
Re: GDB for Linux (David Grothe)
Re: Another kernel debugging tool (David Hinds)
Re: Kernel debugging with breakpoints (Keith Owens)
Keywords: Linux kernel debugging
Date: Thu, 23 Oct 1997 11:05:51 GMT
From: C.H.Gopinath <gopich@cse.iitb.ernet.in>

```
hi,
     I have downloaded the xkdebug_for_2.1.55. I tried to install it. It has
generated the Makefile.rej.
I am new to linux. So can u please suggest me how to deal with that file.
By the by I am using redhat-4.2 kernel 2.0.30. Can I use this
debugger or not?

Here i am giveing the rej file.

***************
*** 88,97 ****
  # standard CFLAGS
  #

- CFLAGS = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer

  ifdef CONFIG_CPP
  CFLAGS := $(CFLAGS) -x c++
  endif

  ifdef SMP
--- 88,103 ----
  # standard CFLAGS
  #

+ CFLAGS = -Wall -Wstrict-prototypes -O2

  ifdef CONFIG_CPP
  CFLAGS := $(CFLAGS) -x c++
+ endif
+
+ ifdef CONFIG_XKDEBUG
+ CFLAGS := $(CFLAGS) -g
+ else
+ CFLAGS := $(CFLAGS) -fomit-frame-pointer
  endif
  ifdef SMP
_XKDEBUG
```

Realtime mods anyone?

Forum: The Linux Kernel Hackers' Guide
Keywords: realtime round-robin task scheduling
Date: Sat, 01 Jun 1996 19:20:04 GMT
From: bill duncan <bduncan@beachnet.org>

I have a requirement for a realtime system doing process control, and I'd like to see if Linux can do it. I believe that the timing constraints are relaxed enough that Linux can do it straight out of the box, but wonder if anyone else has done enhancements for realtime.

The timing constraints are less than 100ms response times for a few external events. Since it will be a single purpose machine, and I will configure it without swap, I doubt that there will be a problem anyway. Nevertheless, if there are mods out there for the scheduling algorithms (like round-robin instead of the Unix-style socialist policy scheduling) I'd appreciate finding out.

Thanks.

```
--
bill duncan, bduncan@beachnet.org
```

Messages

1. 100 ms real time should be easy *by jeff millar*
2. Realtime is already done(!) *by Kai Harrekilde-Petersen*
4. POSIX.4 scheduler *by Peter Monta*
    1. cli()/sti() latency, hard numbers *by Ingo Molnar*
5. found some hacks ?!? *by Mayk Langer*
6. Hard real-time now available *by Michael K. Johnson*
7. Summary of Linux Real-Time Status *by Markus Kuhn*

100 ms real time should be easy

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Keywords: realtime round-robin task scheduling
Date: Mon, 03 Jun 1996 03:07:35 GMT
From: jeff millar <jeff@wa1hco.mv.com>

Linux 1.3.(some high numbers) kernels have fairly good real time performance. Applications can use POSIX real time scheduling with absolute priorities higher than any process. I ran the realtime test programs associated with some program (don't remember the name) for POSIX realtime process testing and noted that the longest time that the kernal locked out the realtime application never exceeded 135 microseconds on my Pentium 100. I assume this means that the longest kernel call tested didn't exceed that number...some other cases might go longer.

I would like to run a test where a realtime process ran on a precision timed interrupt at the same time the overall Linux kernel performed it full range of functions. This realtime process sole job would be to measure interrupt latency and histogram them, probably through the /proc filesyste. My learning curve for this task would be quite steep but if someone would like to take on this task for a little education, I'd be interested in the results.

jeff

Realtime is already done(!)

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Keywords: realtime round-robin task scheduling
Date: Tue, 04 Jun 1996 12:46:00 GMT
From: Kai Harrekilde-Petersen <khp@dolphinics.no>

I remember that someone implemented a POSIX.4 (aka Real-Time) scheduler for Linux, perhaps a year ago. However, I don't remember who. You probably need to grep through the collected kernel mailing list archives to find it.

Kai

POSIX.4 scheduler

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Keywords: realtime round-robin task scheduling
Date: Tue, 09 Jul 1996 00:47:18 GMT
From: Peter Monta <pmonta@gi.com>

The author of the POSIX.4 scheduler mods is Markus Kuhn; the archives or dejanews will have the announcements and performance utilities. I assume everything made it into the 2.0 kernel.

I did have occasion to compare the dispatch latency with real (microsecond-resolution) hardware timers. Once it's running under SCHED_FIFO and everything is locked down, latency is quite stable, though there were a few spikes up to a few milliseconds. I think this might have been some network code.

In general I don't think there's heavy emphasis on the part of kernel-driver authors to be careful about disabling interrupts for a long time. Your mileage will depend on what mix of kernel code gets run. Some sort of monitoring is a very good idea; I'm told the Pentium has a cycle counter on-chip, which is ideal.

Cheers, Peter Monta

Messages

1. cli()/sti() latency, hard numbers *by Ingo Molnar*

cli()/sti() latency, hard numbers

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Re: POSIX.4 scheduler (Peter Monta)
Keywords: realtime round-robin task scheduling
Date: Sat, 17 Aug 1996 10:45:38 GMT
From: Ingo Molnar <mingo@pc5829.hil.siemens.at>

i've done some timings on cli()/sti() latency, on IP basis. Most parts of the kernel are OK, they have less than 100 usecs of max latency. There is one thing why device driver writers take care of cli()/sti() latencies, it's the serial interrupt. If the latency is too high, then we loose serial data quite easily. Some hard data: on a 100 MHz Neptun dual CPU system, hardware interrupt latency is 10+-1 usecs, typical cli()/sti() latencies are on the order of 10 usecs. Some code like the IDE driver has latencies up to 100 usecs, occasionally higher. The IDE driver latency can be minimized by using the hdparm utility: multiple mode and irq masking should be turned off.

found some hacks ?!?

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Keywords: realtime round-robin task scheduling
Date: Wed, 24 Jul 1996 18:04:53 GMT
From: Mayk Langer <langer@vsys.de>

i have found some hack files on

`http://www.dur.ac.uk/~dph0www2/martini/WFS/linux-rtx.html`

i dont know how to use the real-timer from simple programs

please send email with ideas

Hard real-time now available

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Keywords: realtime round-robin task scheduling
Date: Sun, 29 Sep 1996 20:51:50 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Victor Yodaiken has announced RT-Linux, a hard-real-time-capable Linux-based OS.

Summary of Linux Real-Time Status

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Keywords: real-time round-robin task scheduling POSIX.1b SCHED_FIFO SCHED_RR
Date: Tue, 01 Oct 1996 17:32:30 GMT
From: Markus Kuhn <mskuhn@cip.informatik.uni-erlangen.de>

A detailed summary of the POSIX.1b real-time support system call interface, the current implementation status of POSIX.1b under Linux, a discussion of various real-time related problems, and many links to related ressources can be found in

`ftp://ftp.informatik.uni-erlangen.de/local/cip/mskuhn/misc/linux-posix.1b`

If you look for information about real-time applications under Linux, start there! Feel free to copy information from there into the Linux KHG.

Shortcomings of RT-Linux

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Re: Hard real-time now available (Michael K. Johnson)
Keywords: realtime round-robin task scheduling RT-Linux KURT KU Real-Time Linux
Date: Sun, 08 Feb 1998 23:25:41 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

Though Victor Yodaikens RT-Linux is great for developing hard real-time applications using Linux, it does not allow real-time tasks to use any of Linux's features (like networking, etc...) To write a real-time application that uses Linux's features, you need to split it into two parts. A part that does not need such features (the real-time part) and a part that needs to use these features (the non-real-time part). These two parts can communicate by using FIFOs (i think). Note that the non real-time part is not given any real-time guarantees. If the real-time application cannot be split into two parts, then you cannot use RT-Linux. See http://hegel.ittc.ukans.edu/projects/kurt for further details. balaji

Firm Realtime available

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Re: Hard real-time now available (Michael K. Johnson)
Keywords: realtime round-robin task scheduling
Date: Mon, 13 Oct 1997 11:57:46 GMT
From: Balaji Srinivasan <balaji@hegel.ittc.ukans.edu>

We at the university of kansas have been working on a firm realtime linux. This version of realtime linux allows you to use standard linux features in realtime tasks and trades off some deadline guarantees. For more information see http://hegel.ittc.ukans.edu/projects/kurt balaji

I want to know how to hack Red Hat Linux Release 5.0

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Re: found some hacks ?!? (Mayk Langer)
Keywords: Red Hat Linux hacking hack hacker
Date: Wed, 01 Jul 1998 06:01:59 GMT
From: Kevin <hack_the_earth@hotmail.com>

Please help

anyone that know how to hack **Red Hat Linux release 5.0** please E-mail me (Kevin) and tell me how, i would really like your help. hack_the_earth@hotmail.com **thanks**

Real-Time Applications with Linux POSIX.4 Scheduling

Forum: The Linux Kernel Hackers' Guide
Re: Realtime mods anyone? (bill duncan)
Re: 100 ms real time should be easy (jeff millar)
Keywords: realtime FIFO scheduling
Date: Wed, 22 Oct 1997 18:08:55 GMT
From: P. Woolley <woolley@cs.york.ac.uk>

I have recently been working on system to implement Real-Time computer control using Linux 1.2.13 with the POSIX.4 (should that be .1b ?) FIFO scheduler.

When I run my RT progs. I also have the usual system running doing various things, and have not experieneced any unpredictability down to sampling intervals of 10ms. This includes doing things like AD/DA device driver access.

I have also, though less deterministically, had intervals as low as 500us to 600us going. (I have been using multiple processes, but in a single process (threaded maybe) fast, deterministic intervals should go fine?)

Why can't we incorporate new changes in linux kernel in KHG ?

Forum: The Linux Kernel Hackers' Guide
Keywords: KHG should reflect current changes in the linux kernel
Date: Fri, 24 May 1996 11:05:22 GMT
From: Praveen Kumar Dwivedi <pkd@wipro.wipsys.soft.net>

I have read version 0.6 completely but so much has changed in the 1.3.xx kernels that for experienced linux kernel hackers KHG no longer is all that useful.

```
   I thinks this is high time we should do a complete overhaul
of KHG so that it reflects current changes in the linux kernel
especially in the areas such as memory management.

  One last thing I am grateful to all those who contributed
 to KHG.
```

Messages

1. You can! by *Michael K. Johnson*

You can!

Forum: The Linux Kernel Hackers' Guide
Re: Why can't we incorporate new changes in linux kernel in KHG ? (Praveen Kumar Dwivedi)
Keywords: KHG should reflect current changes in the linux kernel
Date: Sun, 26 May 1996 15:48:35 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

> *I have read version 0.6 completely but so much has changed in the 1.3.xx kernels that for experienced linux kernel hackers KHG no longer is all that useful.*

Was the KHG ever *really* useful for *experienced* Linux kernel hackers? I would guess not.

> *I thinks this is high time we should do a complete overhaul of KHG so that it reflects current changes in the linux kernel especially in the areas such as memory management.*

I'm glad you phrased it that way. **We** should indeed do a complete overhaul of the KHG. Here's how to do this: as you find disparities, please post them as responses. Those responses will be used to update the KHG. Without those responses, the updates **will not** happen except as I happen to notice the disparities, which doesn't happen much. Without **your** help, the KHG will remain hopelessly mired in the past.

If you look at the pages, you will see that kind readers have already started this process. To everyone who has contributed a change, fix, elucidation, update, or whatever, thank you very much! Keep up the good work, everyone, and we'll have a document worth reading.

Kernel source code

Forum: The Linux Kernel Hackers' Guide
Date: Sat, 18 May 1996 15:45:45 GMT
From: Gabor J.Toth <jtoth@princeton.edu>

I like the new setup of KHG a lot! One thing I miss is that the source of the kernel is not available along with KHG. I often browse the net from work where I don't have linux available and I guess I'm not alone. Putting a version of the kernel (the latest my be a good idea but pretty much any would do) onto an ftp server and making a link to it from least the main page (but eventually from all relevant pages) might be of great help for many.

Comments, anyone?

Messages

1. The sounds of silence... *by Gabor J.Toth*
    1. Breaking the silence :) *by Kyle Ferrio*
        1. Scribbling in the margins *by Michael K. Johnson*
    2. It requires thought... *by Michael K. Johnson*
2. Kernel source is already browsable online *by Axel Boldt*

The sounds of silence...

Forum: The Linux Kernel Hackers' Guide
Re: Kernel source code (Gabor J.Toth)
Date: Thu, 23 May 1996 01:23:35 GMT
From: Gabor J.Toth <jtoth@princeton.edu>

I didn't expect to get very much response but, boy, I thought I would get some. Does this silence mean that

```
 * everybody likes the idea and has nothing to add;

 * everybody found the idea dumb and boring and didn't care to
   comment;

 * no one could figure out what I meant?
```

Comments, this time?

Messages

1. Breaking the silence :) *by Kyle Ferrio*
   1. Scribbling in the margins *by Michael K. Johnson*
2. It requires thought... *by Michael K. Johnson*

Breaking the silence :)

Forum: The Linux Kernel Hackers' Guide
Re: Kernel source code (Gabor J.Toth)
Re: The sounds of silence... (Gabor J.Toth)
Keywords: source links hardcopy
Date: Thu, 23 May 1996 17:24:37 GMT
From: Kyle Ferrio <kylef@engin.umich.edu>

Hypertext links to germane sections of the kernel source would be great, especially for those like me who are just starting to hack a path through the woods.

Links to the code would also go a long way toward convincing me that a fully on-line, interactive khg is a Good Thing. At present, I still prefer having pulp and carbon. It's just too hard to scribble in the margins of a Web page. :)

```
 Respectfully Submitted,
 Kyle Ferrio
```

Messages

1. Scribbling in the margins *by Michael K. Johnson*

Scribbling in the margins

Forum: The Linux Kernel Hackers' Guide
Re: Kernel source code (Gabor J.Toth)
Re: The sounds of silence... (Gabor J.Toth)
Re: Breaking the silence :) (Kyle Ferrio)
Keywords: source links hardcopy
Date: Sun, 26 May 1996 15:32:04 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

You wrote:

> *It's just too hard to scribble in the margins of a Web page. :)*

You just scribbled in the margin. The difference is that here other people can benefit from your scriblings.

Broken URL: http://www.redhat.com:8080/HyperNews/get/khg/3/1/1/www-personal.engin.umich.edu/~kylef

Try: http://www.redhat.com:8080/HyperNews/get/khg/3/1/1.html

It requires thought...

Forum: The Linux Kernel Hackers' Guide
Re: Kernel source code (Gabor J.Toth)
Re: The sounds of silence... (Gabor J.Toth)
Date: Sun, 26 May 1996 15:37:49 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

I didn't respond because I wanted to think about how it could be done, and done in such a way that it would actually be helpful...

I just saw a pointer to cxref posted to comp.os.linux.announce. I don't think that we would have to use its extra documentation features to use its cross-referencing features. I would appreciate it if someone would retrieve and evaluate cxref for this purpose.

Kernel source is already browsable online

Forum: The Linux Kernel Hackers' Guide
Re: Kernel source code (Gabor J.Toth)
Keywords: Linux kernel source code browsing WWW online
Date: Wed, 16 Oct 1996 22:03:05 GMT
From: Axel Boldt <boldt@math.ucsb.edu>

There's already an excellent online Linux kernel source code browser at http://sunsite.unc.edu/linux-source/

I think we should have a pointer to it somewhere near the top of the KHG.

```
Thanks,
  Axel
```

Need easy way to download whole KHG

Forum: The Linux Kernel Hackers' Guide
Keywords: administrivia
Date: Sat, 18 May 1996 07:04:31 GMT
From: *<unknown>*

```
Those of us who have slow dialup links, and who consequently
use suck (or equivalent) to download news quickly and read
it offline, would similarly like to be able to download the
KHG and read it offline.  It would be nice if there were a
packaged version to make this simpler.  (Doing a tree
traversal with Lynx is no fun at all if one needs to be
sure one has obtained the whole thing.)
```

Messages

1. Mirror packages are available, but that's not really enough *by Michael K. Johnson*
    4. Mirror whole KHG package, off line reading and Post to this site *by Kim In-Sung*
        2. Untitled *by Jim Van Zandt* NEW
        1. That works. (using it now). Two tips: *by Richard Braakman*
            1. Anyone willing to put the whole lot up for FTP *by Richard Braakman*
            -> Probably... *by Michael K. Johnson*
    1. Pointer to an HTTP mirror package *by Amos Shapira*
2. postscript version of these documents? *by Michael Stiller*
    1. Sure! *by Michael K. Johnson*
    -> Not so Sure! *by jeff millar*
    -> Enough already! *by Michael K. Johnson*
3. an iterator query might not be hard... *by Mark Eichin*
    1. Might be harder here... *by Michael K. Johnson*

Mirror packages are available, but that's not really enough

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Keywords: administrivia
Date: Sat, 18 May 1996 15:41:36 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

I also have a slow dialup link; I use a 14.4 modem and I'm currently about 20 network hops away from the home site. So I understand how annoying it can be.

Nevertheless, I'm not going to offer a tar file of the KHG at this time. The primary reason for going to a web-based presentation was to make it interactive, and pushing it off-line would make it less interactive.

For now, you can use web mirroring packages to create your own local mirror and read from there. Several such packages are available, and since I don't use any of them, I don't have details like names: can someone else who knows please post details?

For the future, it would be nice if someone is interested enough in mirroring to write scripts for the KHG that allow you to read it off-line, but also allow you to respond. However, I will only give my imprimatur to a system which:

- Mirrors everything, including responses. The responses are just as much a part of the new KHG as the bodies of the articles.
- Provides a reasonable method of posting responses from the off-line state; preferably it will queue them up, and the process of "mirroring" the KHG will involve two steps:
  1. Post any queued responses back to the original site
  2. Mirror the whole thing again.

  They need to happen in that order, obviously.

If anyone is interested in working on such a system, they are welcome to. I won't be working on it; I have too many other things on my plate and I wouldn't use it, so I'd be a lousy choice for building it. The person who writes the system really ought to be someone who will use it...

I would include a pointer to those scripts from within the KHG if anyone wrote them to my satisfaction.

Messages

4. Mirror whole KHG package, off line reading and Post to this site *by Kim In-Sung*
    2. Untitled *by Jim Van Zandt* NEW
    1. That works. (using it now). Two tips: *by Richard Braakman*
        1. Anyone willing to put the whole lot up for FTP *by Richard Braakman*
        -> Probably... *by Michael K. Johnson*
1. Pointer to an HTTP mirror package *by Amos Shapira*

Mirror whole KHG package, off line reading and Post to this site

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Keywords: administrivia
Date: Sun, 02 Mar 1997 18:00:02 GMT
From: Kim In-Sung <kisskiss@soback.kornet.nm.kr>

(My English is not perfect. Sorry)

Try this package

`ftp://sunsite.unc.edu/pub/Linux/Incoming/getwww-1.4.tar.gz`

I write this program. :-P

I mirror KHG, open this KHG locally and post this article locally. But this article is posted KHG original site.

Getwww translate absolute URL to relative links between local files. and don't touch <FORM ACTION...>.

So you can get KHG in batch mode and read local file reading mode in Netscape and post your article to this site.

KHG HTML links are some mingle(is it correct? I mean "not simple" Hmmmm... maybe "complex links") so you can use Getwww options like this

```
 getwww http://www.redhat.com:8080/HyperNews/khg.html \
 -S embed frame outline show=all admin Response \
 -D HyperNews/SECURED HyperNews/thread.pl \
    HyperNews/edit-subscribe.pl
 -l login:passwd
```

```
-S means  don't get files have [string] in file name
   the same html file are included normal mode, frame mode,
   embed mode, admin mode .....
   So such URL is useless.
-D means don't get files in [dir] directory
   files in those directory are not meaning in local mode.
-l Getwww uuencode your password and user name, but Redhat
   Web server not accept such simple encoding, so this option
   is useless, but I use this option for your understanding.
```

Next time you want get new KHG version, use this option

```
 getwww http://www.redhat.com:8080/HyperNews/khg.html \
 -S embed frame outline show=all admin Response \
 -D HyperNews/SECURED HyperNews/thread.pl \
    HyperNews/edit-subscribe.pl
 -l login:passwd  \
 -i
```

With -i option, Getwww get updated files from KHG site.

I write README file for Getwww in Korean, I found someone translate it in English. Maybe you can help me.

Thanks

Messages

2. Untitled *by Jim Van Zandt* NEW
1. That works. (using it now). Two tips: *by Richard Braakman*
    1. Anyone willing to put the whole lot up for FTP *by Richard Braakman*
    -> Probably... *by Michael K. Johnson*

Untitled

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Re: Mirror whole KHG package, off line reading and Post to this site (Kim In-Sung)
Keywords: getwww mirror
Date: Sun, 08 Jun 1997 01:08:53 GMT
From: Jim Van Zandt <jrv@vanzandt.mv.com>

The sunsite archives have apparently been reorganized. The getwww application has been moved to: ftp://sunsite.unc.edu/pub/Linux/apps/www/mirroring/getwww-1.4.tar.gz

That works. (using it now). Two tips:

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Re: Mirror whole KHG package, off line reading and Post to this site (Kim In-Sung)
Keywords: administrivia
Date: Wed, 19 Mar 1997 00:49:48 GMT
From: Richard Braakman <dark@xs4all.nl>

First, the package is no longer in incoming. I found it in ftp://sunsite.unc.edu/pub/Linux/apps/www/getwww-1.4.tar.gz (Not hard to find, but I thought I'd save people the trouble) Second, it's a good idea to put ~johnsonm/index.html among the URLs to avoid. It contains further links to scanned books, and getwww will happily suck down 2 MB of prose :-) I haven't tried the "upgrade" flag yet.

Messages

1. Anyone willing to put the whole lot up for FTP *by Richard Braakman*
-> Probably... *by Michael K. Johnson*

Anyone willing to put the whole lot up for FTP

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Re: Mirror whole KHG package, off line reading and Post to this site (Kim In-Sung)
Re: That works. (using it now). Two tips: (Richard Braakman)
Keywords: administrivia
Date: Wed, 19 Mar 1997 00:49:48 GMT
From: Richard Braakman <dark@xs4all.nl>

Is anyone interested in putting all of their mirror up for FTP in one file, to save load on the Hypernews servers and speed up the downloading a bit?

Messages

1. Probably... *by Michael K. Johnson*

Probably...

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Re: Mirror whole KHG package, off line reading and Post to this site (Kim In-Sung)
Re: That works. (using it now). Two tips: (Richard Braakman)
Re: Anyone willing to put the whole lot up for FTP (Richard Braakman)
Keywords: administrivia
Date: Tue, 08 Apr 1997 00:39:17 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

I'll probably do that at some point. I only ask that if anyone else wants to do this, they make **AT LEAST** a nightly mirror so that downloaders are up-to-date.

Pointer to an HTTP mirror package

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Keywords: administrivia
Date: Sun, 19 May 1996 08:16:33 GMT
From: Amos Shapira <amoss@cs.huji.ac.il>

A little dig into the archives of comp.lang.perl provided the following link:

`http://www.cs.trinity.edu/~nyarrow/MirrorTools/`

This is a beta release, and I haven't run it yet, but it looks promising.

postscript version of these documents?

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Keywords: administrivia
Date: Wed, 22 May 1996 21:33:33 GMT
From: Michael Stiller <michael@toyland.ping.de>

like the other 'old' khg, a postscript version would be good, it's nice to have a carbon copy at hand if you try to figure things out.

Messages

1. Sure! *by Michael K. Johnson*
-> Not so Sure! *by jeff millar*
-> Enough already! *by Michael K. Johnson*

Sure!

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: postscript version of these documents? (Michael Stiller)
Keywords: administrivia
Date: Sun, 26 May 1996 16:58:49 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Most browsers allow you to print the page you are looking at. Just print the page that you are interested in, and any responses to that page the bear on what you are doing. That way, you will get the latest possible information; it will be better than the old KHG where you could hardly know if the information you were reading was two, three, or four years old.

Messages

1. Not so Sure! *by jeff millar*
-> Enough already! *by Michael K. Johnson*

Not so Sure!

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: postscript version of these documents? (Michael Stiller)
Re: Sure! (Michael K. Johnson)
Keywords: administrivia
Date: Mon, 03 Jun 1996 02:52:39 GMT
From: jeff millar <jeff@wa1hco.mv.com>

```
You suggested printing each page...sound painful to me given
several hundred pages.  Having an assembled downloadable
format as an option seems desireable.

jeff
```

Messages

1. Enough already! *by Michael K. Johnson*

Enough already!

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: postscript version of these documents? (Michael Stiller)
Re: Sure! (Michael K. Johnson)
Re: Not so Sure! (jeff millar)
Keywords: administrivia
Date: Mon, 03 Jun 1996 18:13:16 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

You suggested printing each page...sound painful to me given several hundred pages.

Web page, not paper page.

- Only print the pages you want to read **right away**.
- If you want to print all responses with the page, click on the **All** item offered under ``**[Embed Depth: 1 2 3 All]**'' above the response list on each HyperNews page and see what you get.
- There aren't hundreds of web pages here, and if there ever are, that would equal many thousands of paper pages, and would be yet another argument against making it easy to print the whole thing out.

End of discussion. (Yes, that's a hint...)

Messages

an iterator query might not be hard...

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Keywords: server administrivia
Date: Fri, 24 May 1996 23:14:34 GMT
From: Mark Eichin <eichin@kitten.gen.ma.us>

```
Something I implemented in kittenweb (still just something I'm playing
with, not widely published yet) was a "report" query that let you grab
the tree at some point and have the server hand you everything below
it. It also did some useful massaging of links (expecting that it
would be in print form.) Kittenweb is inspired by wikiwikiweb,
http://c2.com/ and is also written in perl.

If the sources to the server used here are available, I'd take a look
and see how hard it would be to add...

                               _Mark_ <eichin@kitten.gen.ma.us>
```

Messages

1. Might be harder here... *by Michael K. Johnson*

Might be harder here...

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: an iterator query might not be hard... (Mark Eichin)
Keywords: server administrivia
Date: Sun, 26 May 1996 16:30:17 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

> *...a "report" query that let you grab the tree at some point and have the server hand you everything below it.*

That would be harder with the KHG because each page is assembled on-request by CGI scripts. We might want to add the indexed kernel source to the KHG at some point, so it might be nice to be able to choose whether or not to include that in the response to the request. :-)

So I don't think it is a trivial operation.

> *If the sources to the server used here are available, I'd take a look and see how hard it would be to add...*

HyperNews feel very free to take a look at it and implement similar functionality for it. I'm sure that it would be useful for far more than just the KHG; HyperNews is used to run a lot of other sites as well.

KHG being mirrored nightly for download!

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Keywords: administrivia
Date: Sat, 14 Jun 1997 15:10:06 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

Please download ftp://ftp.redhat.com/johnsonm/khg.tar.gz and try it out. There are a few broken links in it, but it seems to work. You'll need a connection to post, but not to read. Thanks to Kim In-Sung for getwww which has enabled this!

Appears to be a bug in getwww, though...

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Re: Mirror whole KHG package, off line reading and Post to this site (Kim In-Sung)
Re: That works. (using it now). Two tips: (Richard Braakman)
Keywords: administrivia
Date: Sat, 14 Jun 1997 14:51:50 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

getwww doesn't seem to understand different port numbers. It would be fine if there were a configuration option that said "do follow links that are on the same site but have different port numbers" or "don't follow links that are on the same site but have different port numbers", but getwww doesn't understand either...

When getting the KHG from port 8080, getwww sees absolute links without a port number specified, and assumes that they should come from port 8080 instead of port 80.

There's another bug, but I don't know whether its a but in getwww or in my. Richard says that "it's a good idea to put ~johnsonm/index.html among the URLs to avoid" but I can't make it actually avoid that, and I've tried a lot of command-line arguments by now. Has anyone made that work? I managed to make it not suck down my home page by explicitly telling the server on port 8080 not to serve public_html pages, but that means that the link to the device driver paper it still tries to download and leaves as a broken link (because of the port number bug). It would make more sense for it to leave it as a remote link, I'd think.

Sucking up to the wrong site... ;)

Forum: The Linux Kernel Hackers' Guide
Re: Need easy way to download whole KHG
Re: Mirror packages are available, but that's not really enough (Michael K. Johnson)
Re: Mirror whole KHG package, off line reading and Post to this site (Kim In-Sung)
Re: That works. (using it now). Two tips: (Richard Braakman)
Re: Appears to be a bug in getwww, though... (Michael K. Johnson)
Keywords: administrivia
Date: Tue, 17 Jun 1997 14:44:11 GMT
From: Jukka Santala <e75644@uwasa.fi>

I haven't been able to avoid the ~johnsonm homepage either - I'm not sure what's the reason, either there could be some mesh with absolute WWW-pagenames or then I just don't know how to quote/escape the character properly... hmm, I'm not sure I tried that, reating it as a mask with \~...

But anyway, I avoided that page using another, altough a bit rough method - I ignored index.html pages altogether. This is because to the best I'm able to tell, none of the actual stuff on these pages uses that filename. This should also keep the thing from running rampant on any other possible future index.html references.

Help make the new KHG a success

Forum: The Linux Kernel Hackers' Guide
Keywords: contributing
Date: Wed, 15 May 1996 16:08:04 GMT
From: Michael K. Johnson <johnsonm@redhat.com>

I can't write and maintain the KHG by myself. In order to make this a success, I need help from readers:

- If you see something wrong, **respond** with a correction.
- If you see something missing, either respond to say that it is missing, or respond with the missing piece.
- If something is unclear, try to find out what it is saying and post a clarification. Otherwise, post a request for clarification.
- If you can answer a request for clarification, do so.
- In order to encourage me to continue providing this service, **subscribe** to the pages in which you are most interested.

Responses **do not** have to be perfect, or in perfect english. Think of this like a mailing list, not a book. When I incorporate responses into the the main text (body) of an article, I'll edit it. I'm good at editing, and I enjoy doing it, especially when I don't have a deadline.

If the KHG remains a simply personal effort, it will become less and less relevant. With your help, it can become more and more worth reading.

Thanks!

Messages