Machine Translated by Google

SYSTEM AND LIBRARY CALLS EVERY PROGRAMMER NEEDS TO KNOW

LINUX SYSTEM PROGRAMMING

O'REILLY®

ROBERT LOVE

www.it-ebooks.info

Machine Translated by Google

TALKING DIRECTLY TO THE KERNEL AND C LIBRARY

LINUX SYSTEM PROGRAMMING

Nearly all programmers at some point have to deal with the system calls and libraries of the operating system on which their programs run. This book is about writing system software for Linux—code that lives at a low level, and talks directly to the kernel and core system libraries. *Linux System Programming* describes the functions and performance trade-offs in using standard interfaces, including advanced Linux-only interfaces.

This book is also an insider's guide to writing smarter, faster code. Author and kernel hacker Robert Love explains not only how system interfaces *should* work, but also how they *actually* work, and how to use them safely and efficiently. *Linux System Programming* contains tricks to help you write better code at any level.

Topics include:

- Reading from and writing to files and other file I/O operations, including how the Linux kernel implements and manages file I/O, memory mappings, and optimization techniques
- System calls for process management, including real-time processes
- File and directories—creating, moving, copying, deleting, and managing them
- Memory management—interfaces for allocating memory, managing memory, and optimizing memory access
- Signals and their role on a Unix system, plus basic and advanced signal interfaces
- Time, sleeping, and clock management, starting with the basics, and covering POSIX clocks and high-resolution timers

With *Linux System Programming*, you will able to take an in-depth look at Linux from a theoretical and applied perspective to make the most effective use of your system.

Robert Love has been a Linux user and hacker since the early days. He is active in—and passionate about—the Linux kernel and GNOME desktop communities. His recent contributions to the Linux kernel include work on the kernel event layer and inotify. GNOME-related contributions include Beagle, GNOME Volume Manager, NetworkManager, and Project Utopia. Currently, Robert works in the Open Source Program Office at Google.

O'REILLY®
www.oreilly.com

US $49.99 CAN $59.99
ISBN-10: 0-596-00958-5
ISBN-13: 978-0-596-00958-8

Safari Books Online
Free online edition
with purchase of this book.
Details on last page.

www.it-ebooks.info

Machine Translated by Google

LINUX

Lập trình hệ thống

www.it-ebooks.info

Machine Translated by Google

Các tài nguyên Linux khác từ O'Reilly

Tiêu đề liên quan Xây dựng Linux nhúng Hệ thống

Thiết kế nhúng Phần cứng

Trình điều khiển thiết bị Linux

Tóm tắt về hạt nhân Linux

Lập trình nhúng Hệ thống

Chạy Linux

Hiểu về Linux Nội bộ mạng

Hiểu về Linux Hạt nhân

Sách Linux
Trung tâm tài nguyên

linux.oreilly.com là danh mục đầy đủ các cuốn sách của O'Reilly về Linux và Unix và các công nghệ liên quan, bao gồm các chư ơ ng mẫu và ví dụ mã.

O'REILLY ONLamp.com LINUX • APACHE • MYSQL • PHP • PYTHON • PERL

ONLamp.com là trang web hàng đầu cho nền tảng web nguồn mở: Linux, Apache, MySQL và Perl, Python hoặc PHP.

Hội nghị O'Reilly tập hợp những nhà đổi mới đa dạng để nuôi dư ỡng những ý tư ởng tạo nên các ngành công nghiệp mang tính cách mạng. Chúng tôi chuyên ghi chép lại các công cụ và hệ thống mới nhất, chuyển đổi kiến thức của nhà đổi mới thành các kỹ năng hữu ích cho những ngư ời trong cuộc. Truy cập con-ferences.oreilly.com để biết các sự kiện sắp tới của chúng tôi.

O'REILLY NETWORK Safari Bookshelf.

Safari Bookshelf (safari.oreilly.com) là thư viện tham khảo trực tuyến hàng đầu dành cho lập trình viên và chuyên gia CNTT. Thực hiện tìm kiếm trên hơ n 1.000 cuốn sách. Ngư ời đăng ký có thể tập trung vào câu trả lời cho các câu hỏi quan trọng về thời gian chỉ trong vài giây. Đọc sách trên Giá sách của bạn từ đầu đến cuối hoặc chỉ cần lật đến trang bạn cần. Hãy dùng thử miễn phí ngay hôm nay.

www.it-ebooks.info

Machine Translated by Google

LINUX

Lập trình hệ thống

Robert tình yêu

O'REILLY®

Bắc Kinh • Cambridge • Farnham • Köln • Paris • Sebastopol • Đài Bắc • Tokyo

www.it-ebooks.info

Machine Translated by Google

Lập trình hệ thống Linux của
Robert Love

Bản quyền © 2007 O'Reilly Media, Inc. Bảo lư u mọi quyền.
Đư ợc in tại Hoa Kỳ.

Đư ợc xuất bản bởi O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Sách O'Reilly có thể đư ợc mua để sử dụng cho mục đích giáo dục, kinh doanh hoặc quảng cáo bán hàng. Phiên bản trực tuyến cũng có sẵn cho hầu hết các tựa sách (safari.oreilly.com). Để biết thêm thông tin, hãy liên hệ với bộ phận bán hàng của công ty/tổ chức của chúng tôi: (800) 998-9938 hoặc corporate@oreilly.com.

Biên tập: Andy Oram
Biên tập sản xuất: Sumita Mukherji
Biên tập viên: Rachel Head
Ngư ời hiệu đính: Sumita Mukherji
Ngư ời lập chỉ mục: John Bickelhaupt
Nhà thiết kế bìa: Karen Montgomery
Nhà thiết kế nội thất: David Futato
Minh họa: Jessamyn Read

Lịch sử in ấn:

Tháng 9 năm 2007: Phiên bản đầu tiên.

Nutshell Handbook, logo Nutshell Handbook và logo O'Reilly là các thư ơ ng hiệu đã đăng ký của O'Reilly Media, Inc. Các tên gọi của dòng Linux , Linux System Programming, hình ảnh ngư ời đàn ông trong cỗ máy biết bay và hình ảnh thư ơ ng mại liên quan là các thư ơ ng hiệu của O'Reilly Media, Inc.

Nhiều tên gọi đư ợc nhà sản xuất và ngư ời bán sử dụng để phân biệt sản phẩm của họ đư ợc coi là nhãn hiệu. Khi những tên gọi đó xuất hiện trong cuốn sách này và O'Reilly Media, Inc. biết về khiếu nại về nhãn hiệu, thì những tên gọi đó đư ợc in bằng chữ hoa hoặc chữ viết tắt.

Mặc dù đã thực hiện mọi biện pháp phòng ngừa trong quá trình chuẩn bị cuốn sách này, nhà xuất bản và tác giả không chịu trách nhiệm về các lỗi hoặc thiếu sót, hoặc về các thiệt hại phát sinh do sử dụng thông tin có trong tài liệu này.

Cuốn sách này sử dụng RepKover™ , một loại bìa cứng phẳng bền và linh hoạt.

Mã số ISBN-10: 0-596-00958-5
Mã số ISBN-13: 978-0-596-00958-8

[Nam]

www.it-ebooks.info

Machine Translated by Google

Mục lục

www.it-ebooks.info

Machine Translated by Google

www.it-ebooks.info

Machine Translated by Google

www.it-ebooks.info

Machine Translated by Google

www.it-ebooks.info

Machine Translated by Google

Lời nói đầu

Có một câu nói cũ mà các nhà phát triển hạt nhân Linux thích nói ra khi họ cảm thấy cáu kỉnh: "Không gian ngư ời dùng chỉ là tải thử nghiệm cho hạt nhân".

Bằng cách lẩm bẩm câu này, các nhà phát triển kernel muốn rũ bỏ mọi trách nhiệm đối với bất kỳ lỗi nào trong việc chạy mã không gian ngư ời dùng tốt nhất có thể. Đối với họ, các nhà phát triển không gian ngư ời dùng chỉ nên đi và sửa mã của riêng họ, vì bất kỳ vấn đề nào chắc chắn không phải là lỗi của kernel.

Để chứng minh rằng thư ờng thì không phải là lỗi của kernel, một nhà phát triển kernel Linux hàng đầu đã có bài phát biểu “Tại sao không gian ngư ời dùng tệ hại” tại các phòng hội nghị chật kín ngư ời trong hơ n ba năm nay, chỉ ra những ví dụ thực tế về mã không gian ngư ời dùng tệ hại mà mọi ngư ời đều dựa vào hàng ngày. Các nhà phát triển kernel khác đã tạo ra các công cụ cho thấy các chư ơ ng trình không gian ngư ời dùng đang lạm dụng phần cứng và làm cạn kiệt pin của những chiếc máy tính xách tay không hề hay biết như thế nào.

Như ng trong khi mã không gian ngư ời dùng có thể chỉ là "tải thử nghiệm" để các nhà phát triển kernel chế giễu, thì hóa ra tất cả các nhà phát triển kernel này cũng phụ thuộc vào mã không gian ngư ời dùng đó hàng ngày. Nếu không có nó, tất cả những gì kernel có thể làm là in ra các mẫu ABABAB xen kẽ trên màn hình.

Hiện tại, Linux là hệ điều hành linh hoạt và mạnh mẽ nhất từng đư ợc tạo ra, chạy mọi thứ từ điện thoại di động nhỏ nhất và các thiết bị nhúng đến hơ n 70 phần trăm trong số 500 siêu máy tính hàng đầu thế giới. Không có hệ điều hành nào khác có thể mở rộng tốt như vậy và đáp ứng đư ợc những thách thức của tất cả các loại phần cứng và môi trư ờng khác nhau này.

Cùng với hạt nhân, mã chạy trong không gian ngư ời dùng trên Linux cũng có thể hoạt động trên tất cả các nền tảng đó, cung cấp cho thế giới các ứng dụng và tiện ích thực sự mà mọi ngư ời tin cậy.

Trong cuốn sách này, Robert Love đã đảm nhiệm nhiệm vụ không đáng ghen tị là dạy cho ngư ời đọc về hầu hết mọi lệnh gọi hệ thống trên hệ thống Linux. Khi làm như vậy, ông đã tạo ra một cuốn sách cho phép bạn hiểu đầy đủ về cách hạt nhân Linux hoạt động theo góc nhìn không gian ngư ời dùng và cách khai thác sức mạnh của hệ thống này.

www.it-ebooks.info

Machine Translated by Google

Thông tin trong cuốn sách này sẽ chỉ cho bạn cách tạo mã chạy trên tất cả các bản phân phối Linux và loại phần cứng khác nhau. Nó sẽ cho phép bạn hiểu cách Linux hoạt động và cách tận dụng tính linh hoạt của nó.

Cuối cùng, cuốn sách này dạy bạn cách viết code sao cho hay, đó chính là điều tuyệt vời nhất.

—Greg Kroah-Hartman

www.it-ebooks.info

Machine Translated by Google

Cuốn sách này nói về lập trình hệ thống—cụ thể là lập trình hệ thống trên Linux. Lập trình hệ thống là hoạt động viết phần mềm hệ thống, tức là mã nằm ở cấp độ thấp, giao tiếp trực tiếp với nhân và các thư viện hệ thống lõi. Nói cách khác, chủ đề của cuốn sách là các lệnh gọi hệ thống Linux và các hàm cấp thấp khác, chẳng hạn như các hàm được định nghĩa bởi thư viện C.

Trong khi nhiều cuốn sách đề cập đến lập trình hệ thống cho các hệ thống Unix, thì rất ít cuốn đề cập đến chủ đề này chỉ tập trung vào Linux, và thậm chí còn ít hơn nữa (nếu có) đề cập đến các bản phát hành Linux mới nhất và các giao diện chỉ dành cho Linux nâng cao. Hơn nữa, cuốn sách này được hưởng lợi từ một nét đặc biệt: Tôi đã viết rất nhiều mã cho Linux, cho cả hạt nhân và phần mềm hệ thống được xây dựng trên đó. Trên thực tế, tôi đã triển khai một số lệnh gọi hệ thống và các tính năng khác được đề cập trong cuốn sách này. Do đó, cuốn sách này chứa đựng rất nhiều kiến thức chuyên sâu, không chỉ đề cập đến cách các giao diện hệ thống nên hoạt động mà còn đề cập đến cách chúng thực sự hoạt động và cách bạn (lập trình viên) có thể sử dụng chúng hiệu quả nhất. Do đó, cuốn sách này kết hợp trong một tác phẩm duy nhất một hướng dẫn về lập trình hệ thống Linux, một hướng dẫn tham khảo về các lệnh gọi hệ thống Linux và hướng dẫn dành cho người trong cuộc để viết mã thông minh hơn, nhanh hơn. Văn bản rất thú vị và dễ hiểu, và bất kể bạn có viết mã ở cấp độ hệ thống hàng ngày hay không, cuốn sách này sẽ dạy bạn các thủ thuật giúp bạn viết mã tốt hơn.

Khán giả và Giả định

Các trang sau đây giả định rằng người đọc đã quen thuộc với lập trình C và môi trường lập trình Linux—không nhất thiết phải thông thạo các chủ đề này, nhưng ít nhất là quen thuộc với chúng. Nếu bạn chưa đọc bất kỳ cuốn sách nào về ngôn ngữ lập trình C, chẳng hạn như tác phẩm kinh điển của Brian W. Kernighan và Dennis M. Ritchie Ngôn ngữ lập trình C (Prentice Hall; cuốn sách thường được gọi là K&R), tôi thực sự khuyên bạn nên xem qua một cuốn. Nếu bạn không thoải mái với trình soạn thảo văn bản Unix—Emacs và vim là những trình soạn thảo phổ biến và được đánh giá cao nhất—hãy bắt đầu chơi

www.it-ebooks.info

Machine Translated by Google

với một. Bạn cũng sẽ muốn làm quen với những điều cơ bản khi sử dụng gcc, gdb, make, v.v. Có rất nhiều sách khác về các công cụ và thực hành lập trình Linux; phần tài liệu tham khảo ở cuối sách này liệt kê một số tài liệu tham khảo hữu ích.

Tôi đã đưa ra một vài giả định về kiến thức của người đọc về lập trình hệ thống Unix hoặc Linux. Cuốn sách này sẽ bắt đầu từ cơ bản, bắt đầu với những điều cơ bản và dần dần đi lên các giao diện và thủ thuật tối ưu hóa tiên tiến nhất.
Tôi hy vọng độc giả ở mọi trình độ sẽ thấy tác phẩm này đáng giá và học được điều gì đó mới mẻ. Trong quá trình viết cuốn sách, tôi chắc chắn đã làm được điều đó.

Tôi cũng không đưa ra giả định về sức thuyết phục hay động lực của người đọc.
Các kỹ sư muốn lập trình (tốt hơn) ở cấp độ thấp rõ ràng là mục tiêu, nhưng các lập trình viên cấp cao hơn đang tìm kiếm một vị thế vững chắc hơn trên nền tảng mà họ dựa vào cũng sẽ tìm thấy nhiều điều khiến họ quan tâm. Những hacker chỉ tò mò cũng được chào đón, vì cuốn sách này cũng sẽ thỏa mãn cơn đói của họ. Bất kể độc giả muốn và cần gì, cuốn sách này sẽ tung lưới đủ rộng—ít nhất là đối với lập trình hệ thống Linux—để thỏa mãn họ.

Bất kể động cơ của bạn là gì, trên hết, hãy vui vẻ.

Nội dung của cuốn sách này

Cuốn sách này được chia thành 10 chương, một phần phụ lục và một phần tài liệu tham khảo.

Chương 1, Giới thiệu và các khái niệm thiết yếu

Chương này đóng vai trò là phần giới thiệu, cung cấp tổng quan về Linux, lập trình hệ thống, hạt nhân, thư viện C và trình biên dịch C. Ngay cả người dùng nâng cao cũng nên xem chương này—tin tôi đi.

Chương 2, Tệp I/O

Chương này giới thiệu tệp, khái niệm trừu tượng quan trọng nhất trong môi trường Unix, và tệp I/O, cơ sở của chế độ lập trình Linux. Chương này đề cập đến việc đọc và ghi vào tệp, cùng với các hoạt động tệp I/O cơ bản khác.
Chương này kết thúc bằng cuộc thảo luận về cách hạt nhân Linux triển khai và quản lý các tệp.

Chương 3, I/O đệm Chương

này thảo luận về một vấn đề liên quan đến giao diện I/O tệp cơ bản—quản lý kích thước bộ đệm—và giới thiệu I/O đệm nói chung và I/O chuẩn nói riêng như là các giải pháp.

Chương 4, Nhập/xuất tệp nâng cao

Chương này hoàn thành bộ ba I/O với cách xử lý các giao diện I/O nâng cao, ánh xạ bộ nhớ và các kỹ thuật tối ưu hóa. Chương này được kết thúc bằng một cuộc thảo luận về việc tránh tìm kiếm và vai trò của trình lập lịch I/O của hạt nhân Linux.

www.it-ebooks.info

Machine Translated by Google

Chư ơ ng 5, Quản lý quy trình Chư ơ ng
này giới thiệu về khái niệm trừu tư ợng quan trọng thứ hai của Unix, quy trình và họ các lệnh gọi hệ thống để quản lý quy trình cơ bản, bao gồm cả nhánh fork đáng kính.

Chư ơ ng 6, Quản lý quy trình nâng cao
Chư ơ ng này tiếp tục trình bày bằng cách thảo luận về quản lý quy trình nâng cao, bao gồm các quy trình thời gian thực.

Chư ơ ng 7, Quản lý tập tin và thư mục
Chư ơ ng này thảo luận về việc tạo, di chuyển, sao chép, xóa và quản lý các tệp và thư mục.

Chư ơ ng 8, Quản lý bộ nhớ Chư ơ ng này
đề cập đến quản lý bộ nhớ. Chư ơ ng này bắt đầu bằng việc giới thiệu các khái niệm Unix về bộ nhớ, chẳng hạn như không gian địa chỉ tiến trình và trang, và tiếp tục bằng cách thảo luận về các giao diện để lấy bộ nhớ từ và trả lại bộ nhớ cho hạt nhân. Chư ơ ng này kết thúc bằng cách xử lý các giao diện liên quan đến bộ nhớ nâng cao.

Chư ơ ng 9, Tín hiệu
Chư ơ ng này đề cập đến tín hiệu. Chư ơ ng này bắt đầu bằng thảo luận về tín hiệu và vai trò của chúng trên hệ thống Unix. Sau đó, chư ơ ng này đề cập đến giao diện tín hiệu, bắt đầu từ cơ bản và kết thúc bằng nâng cao.

Chư ơ ng 10, Thời
gian Chư ơ ng này thảo luận về thời gian, chế độ ngủ và quản lý đồng hồ. Chư ơ ng này đề cập đến các giao diện cơ bản cho đến đồng hồ POSIX và bộ đếm thời gian có độ phân giải cao.

Phụ lục, Phần mở rộng của GCC cho Ngôn ngữ C Phụ lục
xem xét nhiều tối ư u hóa do gcc và GNUC cung cấp, chẳng hạn như các thuộc tính để đánh dấu hàm hằng số, thuần túy và nội tuyến.

Cuốn sách kết thúc bằng một danh mục sách đọc đư ợc khuyến nghị, liệt kê cả những tài liệu bổ sung hữu ích cho tác phẩm này và những cuốn sách giải quyết các chủ đề tiên quyết không đư ợc đề cập ở đây.

Các phiên bản đư ợc đề cập trong cuốn sách này

Giao diện hệ thống Linux có thể đư ợc định nghĩa là giao diện nhị phân ứng dụng và giao diện lập trình ứng dụng đư ợc cung cấp bởi bộ ba hạt nhân Linux (trái tim của hệ điều hành), thư viện GNUC (glibc) và GNUC Compiler (gcc – hiện đư ợc gọi chính thức là GNUCompiler Collection, như ng chúng ta chỉ quan tâm đến C). Cuốn sách này đề cập đến giao diện hệ thống đư ợc định nghĩa bởi hạt nhân Linux phiên bản 2.6.22, glibc phiên bản 2.5 và gcc phiên bản 4.2. Các giao diện trong cuốn sách này phải tư ơ ng thích ngư ợc với các phiên bản cũ hơ n (trừ các giao diện mới) và tư ơ ng thích hư ớng tới các phiên bản mới hơ n.

www.it-ebooks.info

Machine Translated by Google

Nếu bất kỳ hệ điều hành nào đang tiến hóa là mục tiêu di động thì Linux chính là một con báo gêpa hung dữ. Tiến trình đư ợc đo bằng ngày chứ không phải năm, và việc phát hành thư ờng xuyên hạt nhân và các thành phần khác liên tục làm thay đổi sân chơ i. Không có cuốn sách nào có thể hy vọng nắm bắt đư ợc một con thú năng động như vậy theo cách vư ợt thời gian.

Tuy nhiên, môi trư ờng lập trình đư ợc định nghĩa bởi lập trình hệ thống đã đư ợc thiết lập cố định. Các nhà phát triển Kernel rất cẩn thận để không phá vỡ các lệnh gọi hệ thống, các nhà phát triển glibc đánh giá cao khả năng tư ơ ng thích ngư ợc và tiến, và chuỗi công cụ Linux tạo ra mã tư ơ ng thích trên nhiều phiên bản (đặc biệt là đối với ngôn ngữ C). Do đó, trong khi Linux có thể liên tục thay đổi, lập trình hệ thống Linux vẫn ổn định và một cuốn sách dựa trên ảnh chụp nhanh của hệ thống, đặc biệt là tại thời điểm này trong quá trình phát triển Linux, có sức mạnh bền bỉ to lớn. Điều tôi đang cố nói rất đơ n giản: đừng lo lắng về việc giao diện hệ thống thay đổi và hãy mua cuốn sách này!

Các quy ư ớc đư ợc sử dụng trong cuốn sách này

Cuốn sách này sử dụng các quy ư ớc đánh máy sau:

In

nghiêng Đư ợc sử dụng để nhấn mạnh, thuật ngữ mới, URL, cụm từ nư ớc ngoài, lệnh và tiện ích Unix, tên tệp, tên thư mục và tên đư ờng dẫn.

Chiều rộng hằng số

Chỉ ra các tệp tiêu đề, biến, thuộc tính, hàm, kiểu, tham số, đối tư ợng, macro và các cấu trúc lập trình khác.

Chiều rộng cố định nghiêng

Chỉ định văn bản (ví dụ: thành phần đư ờng dẫn) đư ợc thay thế bằng giá trị do ngư ời dùng cung cấp.

Biểu tư ợng này biểu thị lời khuyên, gợi ý hoặc ghi chú chung.

Hầu hết mã trong cuốn sách này đều ở dạng các đoạn mã ngắn như ng hữu ích. Chúng trông như thế này:

```
trong khi (1)
          { int ret;

          ret = nĩa ( ); if
          (ret == -1) perror
                    ("ngã ba");
}
```

Rất nhiều công sức đã đư ợc bỏ ra để cung cấp các đoạn mã ngắn gọn như ng hữu ích. Không cần các tệp tiêu đề đặc biệt, đầy các macro điên rồ và các phím tắt khó đọc. Thay vì xây dựng một vài chư ơ ng trình khổng lồ, cuốn sách này chứa đầy nhiều ví dụ đơ n giản.

www.it-ebooks.info

Machine Translated by Google

Vì các ví dụ mang tính mô tả và hoàn toàn dễ sử dụng, nhưng lại nhỏ và rõ ràng, tôi hy vọng chúng sẽ cung cấp hướng dẫn hữu ích khi đọc lần đầu và vẫn là tài liệu tham khảo tốt cho những lần đọc tiếp theo.

Hầu như tất cả các ví dụ trong cuốn sách này đều là độc lập. Điều này có nghĩa là bạn có thể dễ dàng sao chép chúng vào trình soạn thảo văn bản của mình và đưa chúng vào sử dụng thực tế. Trừ khi được đề cập khác, tất cả các đoạn mã đều phải được xây dựng mà không cần bất kỳ cờ biên dịch đặc biệt nào. (Trong một số trường hợp, bạn cần liên kết với một thư viện đặc biệt.) Tôi khuyên bạn nên sử dụng lệnh sau để biên dịch tệp nguồn:

```
$ gcc -Wall -Wextra -O2 -g -o đoạn trích snippet.c
```

Điều này biên dịch tệp nguồn snippet.c thành đoạn mã nhị phân có thể thực thi, cho phép nhiều lần kiểm tra cảnh báo, tối ưu hóa quan trọng nhưng hợp lý và gỡ lỗi. Mã trong cuốn sách này sẽ biên dịch bằng lệnh này mà không có lỗi hoặc cảnh báo—mặc dù tất nhiên, trước tiên bạn có thể phải xây dựng một chương trình khung xung quanh đoạn mã.

Khi một phần giới thiệu một chức năng mới, nó sẽ ở định dạng trang hướng dẫn Unix thông thường với phông chữ nhấn mạnh đặc biệt, trông như thế này:

```
#include <fcntl.h>

int posix_fadvise (int fd, off_t pos, off_t len, int lời khuyên);
```

Tiêu đề bắt buộc và mọi định nghĩa cần thiết đều nằm ở trên cùng, theo sau là nguyên mẫu đầy đủ của cuộc gọi.

Sách Safari® trực tuyến

Khi bạn nhìn thấy biểu tượng Safari® Books Online trên bìa cuốn sách công nghệ yêu thích của mình, điều đó có nghĩa là cuốn sách đó có sẵn trực tuyến thông qua O'Reilly Network Safari Bookshelf.

Safari cung cấp một giải pháp tốt hơn so với sách điện tử. Đây là một thư viện ảo cho phép bạn dễ dàng tìm kiếm hàng ngàn cuốn sách công nghệ hàng đầu, cắt và dán các mẫu mã, tải xuống các chương và tìm câu trả lời nhanh chóng khi bạn cần thông tin chính xác và mới nhất. Hãy dùng thử miễn phí tại http://safari.oreilly.com.

Sử dụng ví dụ mã

Cuốn sách này ở đây để giúp bạn hoàn thành công việc của mình. Nhìn chung, bạn có thể sử dụng mã trong cuốn sách này trong các chương trình và tài liệu của mình. Bạn không cần phải liên hệ với chúng tôi để xin phép trừ khi bạn đang sao chép một phần đáng kể của mã. Ví dụ, việc viết một chương trình sử dụng một số đoạn mã từ cuốn sách này không yêu cầu phải xin phép. Việc bán hoặc phân phối một đĩa CD-ROM gồm các ví dụ từ sách O'Reilly yêu cầu phải xin phép. Trả lời một câu hỏi bằng cách trích dẫn cuốn sách này và trích dẫn

www.it-ebooks.info

Machine Translated by Google

Mã ví dụ không cần xin phép. Việc kết hợp một lư ợng lớn mã ví dụ từ cuốn sách này vào tài liệu sản phẩm của bạn cần phải xin phép.

Chúng tôi đánh giá cao việc ghi nguồn. Việc ghi nguồn thư ờng bao gồm tiêu đề, tác giả, nhà xuất bản và ISBN. Ví dụ: “Lập trình hệ thống Linux của Robert Love. Bản quyền 2007 O'Reilly Media, Inc., 978-0-596-00958-8.”

Nếu bạn tin rằng việc bạn sử dụng các ví dụ mã nằm ngoài phạm vi sử dụng hợp lý hoặc quyền đư ợc cấp ở trên, vui lòng liên hệ với chúng tôi theo địa chỉ permissions@oreilly.com.

Làm thế nào để liên hệ với chúng tôi

Vui lòng gửi ý kiến và câu hỏi liên quan đến cuốn sách này tới nhà xuất bản:

Công ty truyền thông O'Reilly
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (tại Hoa Kỳ hoặc Canada) 707-829-0515
(quốc tế hoặc địa phư ơ ng) 707-829-0104 (fax)

Chúng tôi có một trang web cho cuốn sách này, nơ i chúng tôi liệt kê các lỗi chính tả, ví dụ và bất kỳ thông tin bổ sung nào. Bạn có thể truy cập trang này tại địa chỉ này:

http://www.oreilly.com/catalog/9780596009588/

Để bình luận hoặc đặt câu hỏi kỹ thuật về cuốn sách này, bạn có thể gửi email đến địa chỉ sau: bookquestions@oreilly.com

Để biết thêm thông tin về sách, hội nghị, Trung tâm tài nguyên và O'Reilly Network, hãy xem trang web của chúng tôi tại địa chỉ này:

http://www.oreilly.com

Lời cảm ơ n

Nhiều trái tim và khối óc đã đóng góp vào việc hoàn thành bản thảo này. Mặc dù không có danh sách nào là đầy đủ, như ng tôi rất vui khi đư ợc ghi nhận sự hỗ trợ và tình bạn của những cá nhân đã động viên, cung cấp kiến thức và hỗ trợ trong suốt quá trình.

Andy Oram là một biên tập viên và con ngư ời phi thư ờng. Nỗ lực này sẽ không thể thực hiện đư ợc nếu không có sự chăm chỉ của anh ấy. Là một ngư ời hiếm có, Andy kết hợp kiến thức chuyên môn sâu rộng với khả năng sử dụng tiếng Anh một cách đầy chất thơ .

www.it-ebooks.info

Machine Translated by Google

Brian Jepson đã đảm nhiệm vai trò biên tập viên một cách xuất sắc trong một thời gian, và những nỗ lực tuyệt vời của ông vẫn tiếp tục đư ợc thể hiện trong suốt tác phẩm này.

Cuốn sách này may mắn có đư ợc những nhà phê bình kỹ thuật phi thư ờng, những bậc thầy thực sự trong nghề của họ, nếu không có họ, tác phẩm này sẽ trở nên nhạt nhòa so với sản phẩm cuối cùng mà bạn đang đọc. Những nhà phê bình kỹ thuật là Robert Day, Jim Lieb, Chris Rivera, Joey Shaw và Alain Williams. Bất chấp những nỗ lực của họ, bất kỳ lỗi nào cũng là của tôi.

Rachel Head đã hoàn thành xuất sắc vai trò biên tập viên. Sau đó, mực đó đã tô điểm cho những từ ngữ tôi viết—độc giả chắc chắn sẽ đánh giá cao những chỉnh sửa của cô ấy.

Vì nhiều lý do, xin cảm ơ n và trân trọng Paul Amici, Mikey Babbitt, Keith Bar-bag, Jacob Berkman, Dave Camp, Chris DiBona, Larry Ewing, Nat Friedman, Albert Gator, Dustin Hall, Joyce Hawkins, Miguel de Icaza, Jimmy Krehl, Greg Kroah-Hartman, Doris Love, Jonathan Love, Linda Love, Tim O'Reilly, Aaron Matthews, John McCain, Randy O'Dowd, Salvatore Ribaudo và gia đình, Chris Rivera, Joey Shaw, Sarah Stewart, Peter Teichman, Linus Torvalds, Jon Trowbridge, Jeremy Van-Doren và gia đình, Luis Villa, Steve Weisberg và gia đình, và Helen Whisnant.

Lời cảm ơ n cuối cùng xin gửi tới bố mẹ tôi, Bob và Elaine.

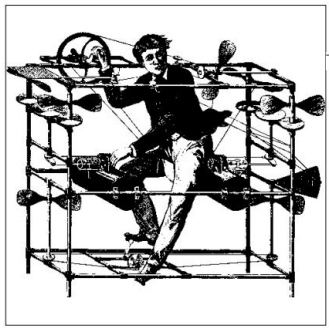
—Robert Tình yêu
Boston

www.it-ebooks.info

Machine Translated by Google

www.it-ebooks.info

Machine Translated by Google



CHƯƠNG 1

Giới thiệu và thiết yếu

Các khái niệm

Cuốn sách này nói về lập trình hệ thống, tức nghệ thuật viết phần mềm hệ thống. Phần mềm hệ thống tồn tại ở cấp độ thấp, giao tiếp trực tiếp với nhân và các thư viện hệ thống lõi. Phần mềm hệ thống bao gồm shell và trình soạn thảo văn bản, trình biên dịch và trình gỡ lỗi, các tiện ích cốt lõi và daemon hệ thống. Các thành phần này hoàn toàn là phần mềm hệ thống, dựa trên nhân và thư viện C. Nhiều phần mềm khác (như các ứng dụng GUI cấp cao) chủ yếu tồn tại ở các cấp độ cao hơn, chỉ thỉnh thoảng mới đi sâu vào cấp độ thấp, nếu có. Một số lập trình viên dành cả ngày để viết phần mềm hệ thống; những người khác chỉ dành một phần thời gian của họ cho nhiệm vụ này. Tuy nhiên, không có lập trình viên nào không được hưởng lợi từ việc hiểu biết về lập trình hệ thống. Cho dù đó là lý do tồn tại của lập trình viên hay chỉ là nền tảng cho các khái niệm cấp cao hơn, thì lập trình hệ thống là cốt lõi của tất cả các phần mềm mà chúng ta viết.

Đặc biệt, cuốn sách này nói về lập trình hệ thống trên Linux. Linux là một hệ thống hiện đại giống Unix, được Linus Torvalds viết từ đầu và là một cộng đồng tin tặc lỏng lẻo trên toàn cầu. Mặc dù Linux chia sẻ các mục tiêu và hệ tư tưởng của Unix, nhưng Linux không phải là Unix. Thay vào đó, Linux đi theo con đường riêng của mình, tách ra khi cần và chỉ hội tụ khi thực tế. Nhìn chung, cốt lõi của lập trình hệ thống Linux giống như trên bất kỳ hệ thống Unix nào khác. Tuy nhiên, ngoài những điều cơ bản, Linux cũng tạo nên sự khác biệt - so với các hệ thống Unix truyền thống, Linux tràn ngập các lệnh gọi hệ thống bổ sung, hành vi khác biệt và các tính năng mới.

Lập trình hệ thống

Theo truyền thống, tất cả lập trình Unix đều là lập trình cấp hệ thống. Về mặt lịch sử, các hệ thống Unix không bao gồm nhiều trừu tượng cấp cao hơn. Ngay cả lập trình trong môi trường phát triển như Hệ thống X Window cũng phơi bày toàn bộ API hệ thống Unix cốt lõi. Do đó, có thể nói rằng cuốn sách này là một cuốn sách về Linux

www.it-ebooks.info

Machine Translated by Google

lập trình nói chung. Như ng lư u ý rằng cuốn sách này không đề cập đến môi trư ờng lập trình Linux – không có hư ớng dẫn nào về make trong các trang này. Nội dung đư ợc đề cập là API lập trình hệ thống đư ợc trình bày trên máy Linux hiện đại.

Lập trình hệ thống thư ờng đư ợc so sánh với lập trình ứng dụng.
Lập trình cấp hệ thống và cấp ứng dụng khác nhau ở một số khía cạnh, như ng không khác nhau ở những khía cạnh khác. Lập trình hệ thống khác biệt ở chỗ các lập trình viên hệ thống phải có nhận thức sâu sắc về phần cứng và hệ điều hành mà họ đang làm việc.
Tất nhiên, cũng có sự khác biệt giữa các thư viện đư ợc sử dụng và các lệnh gọi đư ợc thực hiện. Tùy thuộc vào "mức" của ngăn xếp mà ứng dụng đư ợc viết, hai thứ này có thể không thực sự có thể hoán đổi cho nhau, như ng nói chung, việc chuyển từ lập trình ứng dụng sang lập trình hệ thống (hoặc ngư ợc lại) không khó. Ngay cả khi ứng dụng nằm ở vị trí rất cao trong ngăn xếp, cách xa các cấp thấp nhất của hệ thống, thì kiến thức về lập trình hệ thống vẫn rất quan trọng. Và các thông lệ tốt tư ơ ng tự đư ợc áp dụng trong mọi hình thức lập trình.

Vài năm trở lại đây đã chứng kiến xu hư ớng trong lập trình ứng dụng chuyển từ lập trình cấp hệ thống sang phát triển cấp rất cao, thông qua phần mềm web (như JavaScript hoặc PHP) hoặc thông qua mã đư ợc quản lý (như C# hoặc Java). Tuy nhiên, sự phát triển này không báo trư ớc cái chết của lập trình hệ thống. Thật vậy, vẫn có ngư ời phải viết trình thông dịch JavaScript và thời gian chạy C#, bản thân nó là lập trình hệ thống. Hơ n nữa, các nhà phát triển viết PHP hoặc Java vẫn có thể hư ởng lợi từ kiến thức về lập trình hệ thống, vì hiểu biết về các thành phần cốt lõi bên trong cho phép viết mã tốt hơ n bất kể mã đư ợc viết ở đâu trong ngăn xếp.

Mặc dù có xu hư ớng này trong lập trình ứng dụng, phần lớn mã Unix và Linux vẫn đư ợc viết ở cấp hệ thống. Phần lớn là C và chủ yếu dựa trên các giao diện do thư viện C và hạt nhân cung cấp. Đây là lập trình hệ thống truyền thống—Apache, bash, cp, Emacs, init, gcc , gdb, glibc, ls, mv, vim và X. Những ứng dụng này sẽ không biến mất trong thời gian tới.

Phạm vi của lập trình hệ thống thư ờng bao gồm phát triển hạt nhân, hoặc ít nhất là viết trình điều khiển thiết bị. Như ng cuốn sách này, giống như hầu hết các văn bản về lập trình hệ thống, không liên quan đến phát triển hạt nhân. Thay vào đó, nó tập trung vào lập trình cấp hệ thống không gian ngư ời dùng; nghĩa là, mọi thứ trên hạt nhân (mặc dù kiến thức về nội bộ hạt nhân là một phần bổ sung hữu ích cho văn bản này). Tư ơ ng tự như vậy, lập trình mạng—socket và các thứ tư ơ ng tự—không đư ợc đề cập trong cuốn sách này. Viết trình điều khiển thiết bị và lập trình mạng là những chủ đề lớn, mở rộng, tốt nhất nên giải quyết trong các cuốn sách dành riêng cho chủ đề này.

Giao diện cấp hệ thống là gì và làm thế nào để tôi viết các ứng dụng cấp hệ thống trong Linux? Chính xác thì hạt nhân và thư viện C cung cấp những gì? Làm thế nào để tôi viết mã tối ư u và Linux cung cấp những thủ thuật nào? Những lệnh gọi hệ thống gọn gàng nào đư ợc cung cấp trong Linux so với các biến thể Unix khác? Tất cả hoạt động như thế nào? Những câu hỏi đó là trọng tâm của cuốn sách này.

Có ba nền tảng cho lập trình hệ thống trong Linux: lệnh gọi hệ thống, thư viện C và trình biên dịch C. Mỗi nền tảng đều đáng đư ợc giới thiệu.

www.it-ebooks.info

Machine Translated by Google

System Calls Lập

trình hệ thống bắt đầu bằng các lệnh gọi hệ thống. Các lệnh gọi hệ thống (thư ờng đư ợc viết tắt là syscall) là các lệnh gọi hàm đư ợc tạo từ không gian ngư ời dùng—trình soạn thảo văn bản, trò chơ i yêu thích của bạn, v.v.—vào kernel (phần bên trong cốt lõi của hệ thống) để yêu cầu một số dịch vụ hoặc tài nguyên từ hệ điều hành. Các lệnh gọi hệ thống bao gồm từ quen thuộc, chẳng hạn như read() và write( ), đến kỳ lạ, chẳng hạn như get_thread_area() và set_tid_address( ).

Linux triển khai ít lệnh gọi hệ thống hơ n nhiều so với hầu hết các hạt nhân hệ điều hành khác. Ví dụ, số lư ợng các lệnh gọi hệ thống của kiến trúc i386 vào khoảng 300, so với hàng nghìn lệnh gọi hệ thống đư ợc cho là trên Microsoft Windows. Trong nhân Linux, mỗi kiến trúc máy (như Alpha, i386 hoặc PowerPC) triển khai danh sách các lệnh gọi hệ thống khả dụng của riêng mình. Do đó, các lệnh gọi hệ thống khả dụng trên một kiến trúc có thể khác với các lệnh gọi hệ thống khả dụng trên một kiến trúc khác. Tuy nhiên, một tập hợp con rất lớn các lệnh gọi hệ thống—hơ n 90 phần trăm—đư ợc triển khai bởi tất cả các kiến trúc. Chính tập hợp con đư ợc chia sẻ này, các giao diện chung này, là những gì tôi đề cập trong cuốn sách này.

Gọi lệnh gọi hệ thống

Không thể liên kết trực tiếp các ứng dụng không gian ngư ời dùng với không gian hạt nhân. Vì lý do bảo mật và độ tin cậy, các ứng dụng không gian ngư ời dùng không đư ợc phép thực thi trực tiếp mã hạt nhân hoặc thao tác dữ liệu hạt nhân. Thay vào đó, hạt nhân phải cung cấp một cơ chế mà ứng dụng không gian ngư ời dùng có thể "báo hiệu" cho hạt nhân rằng nó muốn gọi lệnh gọi hệ thống. Sau đó, ứng dụng có thể bẫy vào hạt nhân thông qua cơ chế đư ợc xác định rõ này và chỉ thực thi mã mà hạt nhân cho phép thực thi. Cơ chế chính xác khác nhau tùy theo kiến trúc. Ví dụ, trên i386, ứng dụng không gian ngư ời dùng thực thi lệnh ngắt phần mềm, int, với giá trị 0x80. Lệnh này gây ra sự chuyển đổi vào không gian hạt nhân, miền đư ợc bảo vệ của hạt nhân, nơ i hạt nhân thực thi trình xử lý ngắt phần mềm—và trình xử lý cho ngắt 0x80 là gì? Không gì khác ngoài trình xử lý lệnh gọi hệ thống!

Ứng dụng cho kernel biết lệnh gọi hệ thống nào cần thực thi và với tham số nào thông qua các thanh ghi máy. Các lệnh gọi hệ thống đư ợc biểu thị bằng số, bắt đầu từ 0. Trên kiến trúc i386, để yêu cầu lệnh gọi hệ thống 5 (là open( )), ứng dụng không gian ngư ời dùng sẽ nhồi 5 vào thanh ghi eax trư ớc khi đư a ra lệnh int .

Việc truyền tham số đư ợc xử lý theo cách tư ơ ng tự. Ví dụ, trên i386, một thanh ghi đư ợc sử dụng cho mỗi tham số có thể có—các thanh ghi ebx, ecx, edx, esi và edi chứa, theo thứ tự, năm tham số đầu tiên. Trong trư ờng hợp hiếm hoi của một lệnh gọi hệ thống có hơ n năm tham số, một thanh ghi duy nhất đư ợc sử dụng để trỏ đến một bộ đệm trong không gian ngư ời dùng nơ i lư u giữ tất cả các tham số. Tất nhiên, hầu hết các lệnh gọi hệ thống chỉ có một vài tham số.

Các kiến trúc khác xử lý lệnh gọi hệ thống theo cách khác nhau, mặc dù tinh thần là như nhau. Là một lập trình viên hệ thống, bạn thư ờng không cần bất kỳ kiến thức nào về cách hạt nhân xử lý lệnh gọi hệ thống. Kiến thức đó đư ợc mã hóa thành các quy ư ớc gọi chuẩn cho kiến trúc và đư ợc trình biên dịch và thư viện C xử lý tự động.

www.it-ebooks.info

Machine Translated by Google

Thư viện C Thư

viện C (libc) là trung tâm của các ứng dụng Unix. Ngay cả khi bạn lập trình bằng ngôn ngữ khác, thư viện C rất có thể vẫn đang hoạt động, được bao bọc bởi các thư viện cấp cao hơn, cung cấp các dịch vụ cốt lõi và tạo điều kiện cho việc gọi lệnh hệ thống. Trên các hệ thống Linux hiện đại, thư viện C được cung cấp bởi GNU libc, viết tắt là glibc và phát âm là gee-lib-see hoặc ít phổ biến hơn là glib-see.

Thư viện GNUC cung cấp nhiều hơn tên gọi của nó. Ngoài việc triển khai thư viện C chuẩn, glibc còn cung cấp các trình bao bọc cho các lệnh gọi hệ thống, hỗ trợ luồng và các tiện ích ứng dụng cơ bản.

Trình biên dịch C

Trong Linux, trình biên dịch C chuẩn được cung cấp bởi GNU Compiler Collection (gcc).
Ban đầu, gcc là phiên bản GNU của cc, C Compiler. Do đó, gcc là viết tắt của GNU C Compiler.
Theo thời gian, hỗ trợ được thêm vào cho ngày càng nhiều ngôn ngữ. Do đó, ngày nay gcc được sử dụng làm tên chung cho họ GNUcompiler.
Tuy nhiên, gcc cũng là nhị phân được sử dụng để gọi trình biên dịch C. Trong cuốn sách này, khi tôi nói về gcc, tôi thường có nghĩa là chương trình gcc, trừ khi ngữ cảnh gợi ý khác.

Trình biên dịch được sử dụng trong hệ thống Unix—bao gồm cả Linux—có liên quan mật thiết đến lập trình hệ thống, vì trình biên dịch giúp triển khai chuẩn C (xem “Tiêu chuẩn ngôn ngữ C”) và ABI hệ thống (xem “API và ABI”), cả hai đều ở phần sau của chương này.

API và ABI

Các lập trình viên thường quan tâm đến việc đảm bảo chương trình của họ chạy trên tất cả các hệ thống mà họ đã hứa sẽ hỗ trợ, hiện tại và trong tương lai. Họ muốn cảm thấy an toàn rằng các chương trình họ viết trên bản phân phối Linux của mình sẽ chạy trên các bản phân phối Linux khác, cũng như trên các kiến trúc Linux được hỗ trợ khác và các phiên bản Linux mới hơn (hoặc cũ hơn).

Ở cấp độ hệ thống, có hai bộ định nghĩa và mô tả riêng biệt tác động đến tính di động. Một là giao diện lập trình ứng dụng (API) và bộ còn lại là giao diện nhị phân ứng dụng (ABI). Cả hai đều định nghĩa và mô tả các giao diện giữa các phần mềm máy tính khác nhau.

API

API định nghĩa các giao diện mà một phần mềm giao tiếp với phần mềm khác ở cấp độ nguồn. API cung cấp khả năng trừu tượng hóa bằng cách cung cấp một tập hợp các giao diện chuẩn—thường là các hàm—mà một phần mềm (thường là, mặc dù không

www.it-ebooks.info

Machine Translated by Google

nhất thiết, một phần cấp cao hơn) có thể gọi từ một phần mềm khác (thường là một phần cấp thấp hơn). Ví dụ, một API có thể trừu tượng hóa khái niệm vẽ văn bản trên màn hình thông qua một họ các hàm cung cấp mọi thứ cần thiết để vẽ văn bản. API chỉ định nghĩa giao diện; phần mềm thực sự cung cấp API được gọi là triển khai API.

Người ta thường gọi API là "hợp đồng". Điều này không đúng, ít nhất là theo nghĩa pháp lý của thuật ngữ này, vì API không phải là thỏa thuận hai chiều. Người dùng API (nói chung là phần mềm cấp cao hơn) không có bất kỳ đầu vào nào đối với API và việc triển khai của nó. Người dùng có thể sử dụng API theo nguyên trạng hoặc không sử dụng nó: chấp nhận hoặc bỏ qua! API chỉ hoạt động để đảm bảo rằng nếu cả hai phần mềm đều tuân theo API, thì chúng tương thích với mã nguồn; nghĩa là, người dùng API sẽ biên dịch thành công so với việc triển khai API.

Một ví dụ thực tế là API được định nghĩa theo chuẩn C và được triển khai theo chuẩn thư viện C. API này định nghĩa một họ các hàm cơ bản và thiết yếu, chẳng hạn như các hàm xử lý chuỗi.

Trong suốt cuốn sách này, chúng ta sẽ dựa vào sự tồn tại của nhiều API khác nhau, chẳng hạn như thư viện I/O chuẩn được thảo luận trong Chương 3. Các API quan trọng nhất trong lập trình hệ thống Linux được thảo luận trong phần "Tiêu chuẩn" sau trong chương này.

ABI

Trong khi API định nghĩa một giao diện nguồn, ABI định nghĩa giao diện nhị phân cấp thấp giữa hai hoặc nhiều phần mềm trên một kiến trúc cụ thể. Nó định nghĩa cách một ứng dụng tương tác với chính nó, cách một ứng dụng tương tác với hạt nhân và cách một ứng dụng tương tác với các thư viện. ABI đảm bảo khả năng tương thích nhị phân, đảm bảo rằng một phần mã đối tượng sẽ hoạt động trên bất kỳ hệ thống nào có cùng ABI mà không cần biên dịch lại.

ABI liên quan đến các vấn đề như quy ước gọi, thứ tự byte, sử dụng thanh ghi, gọi lệnh hệ thống, liên kết, hành vi thư viện và định dạng đối tượng nhị phân.
Ví dụ, quy ước gọi sẽ xác định cách các hàm được gọi, cách các đối số được truyền cho các hàm, thanh ghi nào được giữ nguyên và thanh ghi nào bị làm sai lệch, và cách trình gọi truy xuất giá trị trả về.

Mặc dù đã có một số nỗ lực nhằm định nghĩa một ABI duy nhất cho một kiến trúc nhất định trên nhiều hệ điều hành (đặc biệt là đối với i386 trên các hệ thống Unix), nhưng những nỗ lực này không mấy thành công. Thay vào đó, các hệ điều hành—kể cả Linux—có xu hướng định nghĩa ABI của riêng chúng theo bất kỳ cách nào chúng thấy phù hợp. ABI gắn chặt với kiến trúc; phần lớn ABI đề cập đến các khái niệm cụ thể của máy, chẳng hạn như các thanh ghi cụ thể hoặc lệnh lắp ráp. Do đó, mỗi kiến trúc máy đều có ABI riêng trên Linux. Trên thực tế, chúng ta có xu hướng gọi một ABI cụ thể theo tên máy của nó, chẳng hạn như alpha hoặc x86-64.

www.it-ebooks.info

Machine Translated by Google

Các lập trình viên hệ thống cần phải biết về ABI, nhưng thường không cần phải ghi nhớ nó. ABI được thực thi bởi toolchain – trình biên dịch, trình liên kết, v.v.–và thường không xuất hiện. Tuy nhiên, kiến thức về ABI có thể dẫn đến lập trình tối ưu hơn và là bắt buộc nếu viết mã lắp ráp hoặc hack vào chính toolchain (sau cùng thì đó là lập trình hệ thống).

ABI cho một kiến trúc nhất định trên Linux có sẵn trên Internet và được triển khai bởi chuỗi công cụ và hạt nhân của kiến trúc đó.

Tiêu chuẩn

Lập trình hệ thống Unix là một nghệ thuật cũ. Những điều cơ bản của lập trình Unix đã tồn tại trong nhiều thập kỷ mà không bị ảnh hưởng. Tuy nhiên, các hệ thống Unix là những con thú năng động. Các thay đổi về hành vi và tính năng được thêm vào. Để giúp đưa trật tự vào sự hỗn loạn, các nhóm tiêu chuẩn mã hóa các giao diện hệ thống thành các tiêu chuẩn chính thức. Có rất nhiều tiêu chuẩn như vậy, nhưng về mặt kỹ thuật, Linux không chính thức tuân thủ bất kỳ tiêu chuẩn nào trong số chúng. Thay vào đó, Linux hướng đến việc tuân thủ hai tiêu chuẩn quan trọng và phổ biến nhất: POSIX và Single UNIX Specification (SUS).

POSIX và SUS ghi lại, trong số những thứ khác, C API cho giao diện hệ điều hành giống Unix. Trên thực tế, chúng định nghĩa lập trình hệ thống, hoặc ít nhất là một tập hợp con chung của nó, cho các hệ thống Unix tương thích.

Lịch sử POSIX và SUS Vào giữa

những năm 1980, Viện Kỹ sư Điện và Điện tử (IEEE) đã tiên phong trong nỗ lực chuẩn hóa các giao diện cấp hệ thống trên các hệ thống Unix. Richard Stallman, người sáng lập phong trào Phần mềm Tự do, đã đề xuất tiêu chuẩn này được đặt tên là POSIX (phát âm là pahz-icks), hiện là viết tắt của Giao diện Hệ điều hành Di động.

Kết quả đầu tiên của nỗ lực này, được ban hành vào năm 1988, là IEEE Std 1003.1-1988 (gọi tắt là POSIX 1988). Năm 1990, IEEE đã sửa đổi tiêu chuẩn POSIX thành IEEE Std 1003.1-1990 (POSIX 1990). Hỗ trợ thời gian thực và luồng tùy chọn đã được ghi lại trong IEEE Std 1003.1b-1993 (POSIX 1993 hoặc POSIX.1b) và IEEE Std 1003.1c-1995 (POSIX 1995 hoặc POSIX.1c). Năm 2001, các tiêu chuẩn tùy chọn đã được đưa vào cùng với POSIX 1990 cơ sở, tạo ra một tiêu chuẩn duy nhất: IEEE Std 1003.1-2001 (POSIX 2001).
Bản sửa đổi mới nhất, được phát hành vào tháng 4 năm 2004, là IEEE Std 1003.1-2004. Tất cả các tiêu chuẩn POSIX cốt lõi đều được viết tắt là POSIX.1, với bản sửa đổi năm 2004 là bản mới nhất.

Vào cuối những năm 1980 và đầu những năm 1990, các nhà cung cấp hệ thống Unix đã tham gia vào "Cuộc chiến Unix", mỗi bên đều đấu tranh để xác định biến thể Unix của mình là hệ điều hành Unix.
Một số nhà cung cấp Unix lớn đã tập hợp xung quanh The Open Group, một tập đoàn công nghiệp

www.it-ebooks.info

Machine Translated by Google

đư ợc hình thành từ sự hợp nhất của Open Software Foundation (OSF) và X/Open. Open Group cung cấp chứng nhận, sách trắng và thử nghiệm tuân thủ. Vào đầu những năm 1990, khi Unix Wars đang diễn ra, Open Group đã phát hành Single UNIX Specification. SUS nhanh chóng trở nên phổ biến, phần lớn là do chi phí (miễn phí) so với chi phí cao của tiêu chuẩn POSIX. Ngày nay, SUS kết hợp tiêu chuẩn POSIX mới nhất.

SUS đầu tiên đư ợc công bố vào năm 1994. Các hệ thống tuân thủ SUSv1 đư ợc đánh dấu là UNIX 95. SUS thứ hai đư ợc công bố vào năm 1997 và các hệ thống tuân thủ đư ợc đánh dấu là UNIX 98. SUS thứ ba và mới nhất, SUSv3, đư ợc công bố vào năm 2002. Các hệ thống tuân thủ đư ợc đánh dấu là UNIX 03. SUSv3 sửa đổi và kết hợp IEEE Std 1003.1-2001 và một số tiêu chuẩn khác. Trong suốt cuốn sách này, tôi sẽ đề cập đến thời điểm các lệnh gọi hệ thống và các giao diện khác đư ợc chuẩn hóa bởi POSIX. Tôi đề cập đến POSIX chứ không phải SUS vì POSIX bao hàm cả SUS.

Tiêu chuẩn ngôn ngữ C Cuốn sách

nổi tiếng của Dennis Ritchie và Brian Kernighan, Ngôn ngữ lập trình C (Prentice Hall), đã hoạt động như thông số kỹ thuật C không chính thức trong nhiều năm sau khi xuất bản năm 1978. Phiên bản C này đư ợc gọi là K&R C. C đã nhanh chóng thay thế BASIC và các ngôn ngữ khác để trở thành ngôn ngữ chung của lập trình máy vi tính. Do đó, để chuẩn hóa ngôn ngữ khá phổ biến vào thời điểm đó, vào năm 1983, Viện Tiêu chuẩn Quốc gia Hoa Kỳ (ANSI) đã thành lập một ủy ban để phát triển phiên bản chính thức của C, kết hợp các tính năng và cải tiến từ nhiều nhà cung cấp khác nhau và ngôn ngữ C++ mới. Quá trình này kéo dài và tốn nhiều công sức, như ng ANSI C đã đư ợc hoàn thành vào năm 1989. Năm 1990, Tổ chức Tiêu chuẩn hóa Quốc tế (ISO) đã phê chuẩn ISO C90, dựa trên ANSI C với một số ít sửa đổi.

Năm 1995, ISO đã phát hành phiên bản cập nhật (mặc dù hiếm khi đư ợc triển khai) của ngôn ngữ C, ISO C95. Tiếp theo là bản cập nhật lớn vào năm 1999 cho ngôn ngữ, ISO C99, giới thiệu nhiều tính năng mới, bao gồm các hàm nội tuyến, kiểu dữ liệu mới, mảng có độ dài thay đổi, chú thích theo kiểu C++ và các hàm thư viện mới.

Linux và các tiêu chuẩn

Như đã nêu ở trên, Linux hư ớng đến việc tuân thủ POSIX và SUS. Nó cung cấp các giao diện đư ợc ghi chép trong SUSv3 và POSIX.1, bao gồm hỗ trợ thời gian thực tùy chọn (POSIX.1b) và luồng tùy chọn (POSIX.1c). Quan trọng hơ n, Linux cố gắng cung cấp hành vi phù hợp với các yêu cầu của POSIX và SUS. Nhìn chung, việc không tuân thủ các tiêu chuẩn đư ợc coi là một lỗi. Linux đư ợc cho là tuân thủ POSIX.1 và SUSv3, như ng vì chư a có chứng nhận POSIX hoặc SUS chính thức nào đư ợc thực hiện (đặc biệt là trên mọi bản sửa đổi của Linux), tôi không thể nói rằng Linux chính thức tuân thủ POSIX hoặc SUS.

www.it-ebooks.info

Machine Translated by Google

Về tiêu chuẩn ngôn ngữ, Linux hoạt động tốt. Trình biên dịch C gcc hỗ trợ ISO C99. Ngoài ra, gcc cung cấp nhiều tiện ích mở rộng riêng cho ngôn ngữ C.
Các phần mở rộng này được gọi chung là GNU C và được ghi lại trong Phụ lục.

Linux không có lịch sử tốt về khả năng tương thích ngược,* mặc dù hiện nay nó đã tốt hơn nhiều. Các giao diện được ghi lại theo tiêu chuẩn, chẳng hạn như thư viện C chuẩn, rõ ràng sẽ luôn duy trì khả năng tương thích với mã nguồn. Khả năng tương thích nhị phân được duy trì trên một phiên bản chính nhất định của glibc, ít nhất là như vậy. Và vì C được chuẩn hóa, gcc sẽ luôn biên dịch C hợp lệ một cách chính xác, mặc dù các tiện ích mở rộng dành riêng cho gcc có thể bị loại bỏ và cuối cùng bị xóa bỏ với các bản phát hành gcc mới . Quan trọng nhất, hạt nhân Linux đảm bảo tính ổn định của các lệnh gọi hệ thống. Khi một lệnh gọi hệ thống được triển khai trong phiên bản ổn định của hạt nhân Linux, nó sẽ được thiết lập chắc chắn.

Trong số các bản phân phối Linux khác nhau, Linux Standard Base (LSB) chuẩn hóa phần lớn hệ thống Linux. LSB là một dự án chung của một số nhà cung cấp Linux dưới sự bảo trợ của Linux Foundation (trước đây là Free Standards Group). LSB mở rộng POSIX và SUS, và thêm một số tiêu chuẩn của riêng mình; nó cố gắng cung cấp một tiêu chuẩn nhị phân, cho phép mã đối tượng chạy mà không cần sửa đổi trên các hệ thống tuân thủ.
Hầu hết các nhà cung cấp Linux đều tuân thủ LSB ở một mức độ nào đó.

Cuốn sách này và các tiêu chuẩn

Cuốn sách này cố tình tránh nói suông về bất kỳ tiêu chuẩn nào. Quá thường xuyên, các sách lập trình hệ thống Unix phải dừng lại để giải thích cách một giao diện hoạt động trong một tiêu chuẩn này so với tiêu chuẩn khác, liệu một lệnh gọi hệ thống nhất định có được triển khai trên hệ thống này so với hệ thống kia hay không và những trang tương tự như vậy. Tuy nhiên, cuốn sách này đặc biệt nói về lập trình hệ thống trên hệ thống Linux hiện đại, được cung cấp bởi các phiên bản mới nhất của hạt nhân Linux (2.6), trình biên dịch gcc C (4.2) và thư viện C (2.5).

Vì giao diện hệ thống thường được thiết lập cố định—ví dụ, các nhà phát triển hạt nhân Linux rất nỗ lực để không bao giờ phá vỡ các giao diện lệnh gọi hệ thống—và cung cấp một số mức độ tương thích giữa mã nguồn và nhị phân, nên cách tiếp cận này cho phép chúng ta đi sâu vào chi tiết về giao diện hệ thống của Linux mà không bị ràng buộc bởi các mối quan tâm về khả năng tương thích với nhiều hệ thống và tiêu chuẩn Unix khác. Việc tập trung vào Linux cũng cho phép cuốn sách này cung cấp cách xử lý chuyên sâu về các giao diện dành riêng cho Linux tiên tiến sẽ vẫn phù hợp và có giá trị trong tương lai xa. Cuốn sách dựa trên kiến thức sâu sắc về Linux, đặc biệt là về cách triển khai và hành vi của các thành phần như gcc và hạt nhân, để cung cấp góc nhìn của người trong cuộc, đầy đủ các phương pháp hay nhất và mẹo tối ưu hóa của một cựu chiến binh giàu kinh nghiệm.

* Người dùng Linux có kinh nghiệm có thể nhớ việc chuyển đổi từ a.out sang ELF, chuyển đổi từ libc5 sang glibc, thay đổi gcc , v.v. Rất may là những ngày đó đã qua rồi.

www.it-ebooks.info

Machine Translated by Google

Các khái niệm về lập trình Linux

Phần này trình bày tổng quan ngắn gọn về các dịch vụ do hệ thống Linux cung cấp.
Tất cả các hệ thống Unix, bao gồm cả Linux, đều cung cấp một tập hợp các trừu tượng và giao diện
chung. Thật vậy, điểm chung này định nghĩa Unix. Các trừu tượng như tệp và
quy trình, giao diện để quản lý đường ống và ổ cắm, v.v., là cốt lõi của những gì
Hệ điều hành Unix.

Tổng quan này giả định rằng bạn đã quen thuộc với môi trường Linux: Tôi cho rằng
mà bạn có thể sử dụng trong shell, sử dụng các lệnh cơ bản và biên dịch một chương trình C đơn
giản. Đây không phải là tổng quan về Linux hoặc môi trường lập trình của nó, mà là
của "những thứ" tạo nên cơ sở của lập trình hệ thống Linux.

Các tập tin và hệ thống tập tin

Tệp là sự trừu tượng cơ bản và căn bản nhất trong Linux. Linux tuân theo
triết lý mọi thứ đều là một tập tin (mặc dù không nghiêm ngặt như một số hệ thống khác, chẳng hạn như
như Plan9*). Do đó, nhiều tương tác diễn ra thông qua việc đọc và viết cho
các tệp, ngay cả khi đối tượng đang đề cập không phải là thứ mà bạn cho là tệp tin hàng ngày của
mình.

Để có thể truy cập, trước tiên phải mở một tệp. Tệp có thể được mở để đọc,
viết, hoặc cả hai. Một tệp mở được tham chiếu thông qua một mô tả duy nhất, một ánh xạ từ
siêu dữ liệu liên quan đến tệp mở trở lại chính tệp cụ thể đó. Bên trong
Nhân Linux, mô tả này được xử lý bởi một số nguyên (kiểu C int) được gọi là
mô tả tập tin, viết tắt là fd. Mô tả tập tin được chia sẻ với không gian người dùng và
được sử dụng trực tiếp bởi các chương trình người dùng để truy cập các tệp. Một phần lớn của
chương trình hệ thống Linux bao gồm mở, thao tác, đóng và sử dụng các mô tả tệp.

Các tập tin thông thường

Những gì mà hầu hết chúng ta gọi là "tệp" là những gì Linux dán nhãn là các tệp thông thường. Một tệp thông
thường chứa các byte dữ liệu, được tổ chức thành một mảng tuyến tính gọi là luồng byte. Trong Linux, không có
tổ chức hoặc định dạng tiếp theo được chỉ định cho một tệp. Các byte có thể có bất kỳ giá trị nào
và chúng có thể được tổ chức trong tệp theo bất kỳ cách nào. Ở cấp độ hệ thống, Linux
không áp đặt cấu trúc lên các tệp ngoài luồng byte. Một số hệ điều hành, chẳng hạn như VMS, cung
cấp các tệp có cấu trúc cao, hỗ trợ các khái niệm như
bản ghi. Linux thì không.

Bất kỳ byte nào trong một tệp có thể được đọc từ hoặc ghi vào. Các hoạt động này bắt đầu
tại một byte cụ thể, đó là "vị trí" khái niệm của một người trong tệp. Vị trí này
được gọi là vị trí tệp hoặc độ lệch tệp. Vị trí tệp là một phần thiết yếu của

* Plan9, một hệ điều hành ra đời từ Bell Labs, thường được gọi là hệ điều hành kế thừa Unix. Nó có một
số ý tưởng sáng tạo và là người theo triết lý mọi thứ đều là một tệp.

www.it-ebooks.info

Machine Translated by Google

siêu dữ liệu mà hạt nhân liên kết với mỗi tệp mở. Khi tệp được mở lần đầu, vị trí tệp là số không. Thông thường, khi các byte trong tệp được đọc hoặc ghi vào, từng byte một, vị trí tệp tăng theo loại. Vị trí tệp cũng có thể được đặt thủ công thành một giá trị nhất định, thậm chí là giá trị nằm ngoài phần cuối của tệp. Ghi một byte vào vị trí tệp nằm ngoài phần cuối của tệp sẽ khiến các byte xen kẽ được đệm bằng số không. Mặc dù có thể ghi các byte theo cách này vào một vị trí nằm ngoài phần cuối của tệp, nhưng không thể ghi các byte vào vị trí trước phần đầu của tệp. Thực hành như vậy nghe có vẻ vô nghĩa và thực sự không có nhiều tác dụng. Vị trí tệp bắt đầu từ số không; không thể âm. Ghi một byte vào giữa tệp sẽ ghi đè lên byte trước đó nằm ở vị trí bù trừ đó. Do đó, không thể mở rộng tệp bằng cách ghi vào giữa tệp. Hầu hết việc ghi tệp diễn ra ở phần cuối của tệp. Giá trị tối đa của vị trí tệp chỉ bị giới hạn bởi kích thước của kiểu C được sử dụng để lưu trữ tệp đó, tức là 64 bit trong Linux hiện đại.

Kích thước của một tệp được đo bằng byte và được gọi là độ dài của tệp. Nói cách khác, độ dài chỉ đơn giản là số byte trong mảng tuyến tính tạo nên tệp. Độ dài của tệp có thể được thay đổi thông qua một hoạt động được gọi là cắt bớt. Tệp có thể được cắt bớt thành kích thước mới nhỏ hơn kích thước ban đầu, dẫn đến việc xóa các byte khỏi cuối tệp. Thật khó hiểu, với tên của hoạt động, tệp cũng có thể được "cắt bớt" thành kích thước mới lớn hơn kích thước ban đầu. Trong trường hợp đó, các byte mới (được thêm vào cuối tệp) được điền bằng số không. Tệp có thể trống (có độ dài bằng không) và do đó không chứa bất kỳ byte hợp lệ nào. Độ dài tệp tối đa, giống như vị trí tệp tối đa, chỉ bị giới hạn bởi các giới hạn về kích thước của các kiểu C mà hạt nhân Linux sử dụng để quản lý tệp. Tuy nhiên, các hệ thống tệp cụ thể có thể áp dụng các hạn chế riêng của chúng, đưa độ dài tối đa xuống một giá trị nhỏ hơn.

Một tập tin có thể được mở nhiều lần, bằng một quy trình khác nhau hoặc thậm chí là cùng một quy trình. Mỗi phiên bản mở của tệp được cấp một mô tả tệp duy nhất; các tiến trình có thể chia sẻ mô tả tệp của chúng, cho phép một mô tả duy nhất được sử dụng bởi nhiều tiến trình.
Hạt nhân không áp đặt bất kỳ hạn chế nào đối với việc truy cập tệp đồng thời. Nhiều quy trình có thể tự do đọc và ghi vào cùng một tệp cùng một lúc. Kết quả của các lần truy cập đồng thời như vậy phụ thuộc vào thứ tự của từng hoạt động riêng lẻ và thường không thể đoán trước. Các chương trình không gian người dùng thường phải phối hợp với nhau để đảm bảo rằng các lần truy cập tệp đồng thời được đồng bộ hóa đầy đủ.

Mặc dù các tệp thường được truy cập thông qua tên tệp, nhưng thực tế chúng không được liên kết trực tiếp với các tên như vậy. Thay vào đó, một tệp được tham chiếu bởi một inode (ban đầu là nút thông tin), được gán một giá trị số duy nhất. Giá trị này được gọi là số inode, thường được viết tắt là i-number hoặc ino. Một inode lưu trữ siêu dữ liệu được liên kết với một tệp, chẳng hạn như dấu thời gian sửa đổi, chủ sở hữu, loại, độ dài và vị trí dữ liệu của tệp—nhưng không có tên tệp! Inode vừa là một đối tượng vật lý, nằm trên đĩa trong các hệ thống tệp theo kiểu Unix, vừa là một thực thể khái niệm, được biểu diễn bằng một cấu trúc dữ liệu trong hạt nhân Linux.

www.it-ebooks.info

Machine Translated by Google

Thư mục và liên kết

Việc truy cập tệp thông qua số inode rất phức tạp (và cũng tiềm ẩn lỗ hổng bảo mật), do đó, tệp luôn được mở từ không gian người dùng theo tên chứ không phải số inode.
Thư mục được sử dụng để cung cấp tên để truy cập tệp. Thư mục hoạt động như một ánh xạ các tên có thể đọc được bằng con người thành số inode. Một cặp tên và inode được gọi là liên kết. Dạng vật lý trên đĩa của ánh xạ này—một bảng đơn giản, một hàm băm hoặc bất kỳ thứ gì—được triển khai và quản lý bởi mã hạt nhân hỗ trợ một hệ thống tệp nhất định. Về mặt khái niệm, thư mục được xem như bất kỳ tệp bình thường nào, với sự khác biệt là nó chỉ chứa ánh xạ các tên thành inode. Hạt nhân sử dụng trực tiếp ánh xạ này để thực hiện các giải pháp tên thành inode.

Khi một ứng dụng không gian người dùng yêu cầu mở một tên tệp nhất định, hạt nhân sẽ mở thư mục chứa tên tệp đó và tìm kiếm tên đã cho. Từ tên tệp, hạt nhân sẽ lấy được số inode. Từ số inode, sẽ tìm thấy inode. Inode chứa siêu dữ liệu liên quan đến tệp, bao gồm vị trí trên đĩa của dữ liệu tệp.

Ban đầu, chỉ có một thư mục trên đĩa, thư mục gốc. Thư mục này thường được ký hiệu bằng đường dẫn /. Nhưng, như chúng ta đều biết, thường có nhiều thư mục trên một hệ thống. Làm thế nào để hạt nhân biết thư mục nào cần tìm để tìm một tên tệp nhất định?

Như đã đề cập trước đó, các thư mục rất giống với các tệp thông thường. Thật vậy, chúng thậm chí còn có các inode liên kết. Do đó, các liên kết bên trong các thư mục có thể trỏ đến các inode của các thư mục khác. Điều này có nghĩa là các thư mục có thể lồng vào bên trong các thư mục khác, tạo thành một hệ thống phân cấp các thư mục. Điều này, đến lượt nó, cho phép sử dụng các tên đường dẫn mà tất cả người dùng Unix đều quen thuộc—ví dụ: /home/blackbeard/landscaping.txt.

Khi nhân được yêu cầu mở một đường dẫn như thế này, nó sẽ duyệt qua từng mục nhập thư mục (được gọi là dentry bên trong nhân) trong đường dẫn để tìm inode của mục nhập tiếp theo. Trong ví dụ trước, nhân bắt đầu tại /, lấy inode cho home, đến đó, lấy inode cho blackbeard, chạy ở đó và cuối cùng lấy inode cho landscape.txt. Hoạt động này được gọi là giải quyết thư mục hoặc đường dẫn. Nhân Linux cũng sử dụng bộ đệm, được gọi là bộ đệm dentry, để lưu trữ kết quả của các giải quyết thư mục, giúp tra cứu nhanh hơn trong tương lai tùy theo vị trí thời gian.* Một đường dẫn bắt đầu tại thư mục gốc được cho là đủ điều kiện và được gọi là đường dẫn tuyệt đối. Một số đường dẫn

không đủ điều kiện; thay vào đó, chúng được cung cấp tương đối với một số thư mục khác (ví dụ: todo/plunder). Các đường dẫn này được gọi là đường dẫn tương đối. Khi được cung cấp một đường dẫn tương đối, nhân bắt đầu giải quyết đường dẫn trong thư mục làm việc hiện tại. Từ thư mục làm việc hiện tại, nhân tra cứu thư mục todo. Từ đó, hạt nhân sẽ lấy inode để cướp bóc.

* Vị trí thời gian là khả năng cao của việc truy cập vào một tài nguyên cụ thể được theo sau bởi một truy cập khác vào cùng một tài nguyên. Nhiều tài nguyên trên máy tính thể hiện vị trí thời gian.

www.it-ebooks.info

Machine Translated by Google

Mặc dù các thư mục đư ợc xử lý như các tệp bình thư ờng, như ng hạt nhân không cho phép chúng đư ợc mở và thao tác như các tệp thông thư ờng. Thay vào đó, chúng phải đư ợc thao tác bằng một tập hợp các lệnh gọi hệ thống đặc biệt. Các lệnh gọi hệ thống này cho phép thêm và xóa các liên kết, dù sao thì đây cũng là hai thao tác hợp lý duy nhất. Nếu không gian ngư ời dùng đư ợc phép thao tác các thư mục mà không có sự trung gian của hạt nhân, thì một lỗi đơ n giản cũng có thể phá hỏng hệ thống tệp.

Liên kết cứng

Về mặt khái niệm, không có gì đư ợc đề cập đến cho đến nay sẽ ngăn cản nhiều tên phân giải thành cùng một inode. Thật vậy, điều này đư ợc phép. Khi nhiều liên kết ánh xạ các tên khác nhau thành cùng một inode, chúng tôi gọi chúng là liên kết cứng.

Liên kết cứng cho phép các cấu trúc hệ thống tệp phức tạp với nhiều tên đư ờng dẫn trỏ đến cùng một dữ liệu. Các liên kết cứng có thể nằm trong cùng một thư mục hoặc trong hai hoặc nhiều thư mục khác nhau. Trong cả hai trư ờng hợp, hạt nhân chỉ cần giải quyết tên đư ờng dẫn đến inode chính xác. Ví dụ, một inode cụ thể trỏ đến một khối dữ liệu cụ thể có thể đư ợc liên kết cứng từ /home/bluebeard/map.txt và /home/blackbeard/treasure.txt.

Xóa một tệp liên quan đến việc hủy liên kết tệp khỏi cấu trúc thư mục, đư ợc thực hiện đơ n giản bằng cách xóa cặp tên và inode của tệp khỏi thư mục. Tuy nhiên, vì Linux hỗ trợ liên kết cứng nên hệ thống tệp không thể hủy inode và dữ liệu liên quan của nó trong mọi thao tác hủy liên kết. Nếu một liên kết cứng khác tồn tại ở nơ i khác trong hệ thống tệp thì sao? Để đảm bảo rằng tệp không bị hủy cho đến khi tất cả các liên kết đến tệp đó bị xóa, mỗi inode chứa một số liên kết theo dõi số lư ợng liên kết trong hệ thống tệp trỏ đến tệp đó. Khi một tên đư ờng dẫn bị hủy liên kết, số liên kết sẽ giảm đi một; chỉ khi đạt đến số không thì inode và dữ liệu liên quan của nó mới thực sự bị xóa khỏi hệ thống tệp.

Liên kết tư ợng

trư ng Liên kết cứng không thể mở rộng hệ thống tập tin vì số inode không có ý nghĩa bên ngoài hệ thống tập tin của inode đó. Để cho phép các liên kết có thể mở rộng hệ thống tập tin và đơ n giản hơ n một chút và ít minh bạch hơ n, các hệ thống Unix cũng triển khai liên kết tư ợng trư ng (thư ờng đư ợc rút gọn thành liên kết tư ợng trư ng).

Liên kết tư ợng trư ng trông giống như các tệp thông thư ờng. Một liên kết tư ợng trư ng có inode và khối dữ liệu riêng, chứa tên đư ờng dẫn đầy đủ của tệp đư ợc liên kết đến. Điều này có nghĩa là các liên kết tư ợng trư ng có thể trỏ đến bất kỳ đâu, bao gồm cả các tệp và thư mục nằm trên các hệ thống tệp khác nhau và thậm chí đến các tệp và thư mục không tồn tại. Một liên kết tư ợng trư ng trỏ đến một tệp không tồn tại đư ợc gọi là liên kết bị hỏng.

Liên kết tư ợng trư ng gây ra nhiều chi phí hơ n liên kết cứng vì việc giải quyết liên kết tư ợng trư ng thực sự liên quan đến việc giải quyết hai tệp: liên kết tư ợng trư ng và sau đó là tệp đư ợc liên kết đến. Liên kết cứng không gây ra chi phí bổ sung này—không có sự khác biệt giữa việc truy cập tệp đư ợc liên kết vào hệ thống tệp nhiều hơ n một lần và tệp chỉ đư ợc liên kết một lần.
Chi phí cho liên kết tư ợng trư ng là rất nhỏ, như ng nó vẫn đư ợc coi là tiêu cực.

www.it-ebooks.info

Machine Translated by Google

Liên kết tư ợng trư ng cũng ít minh bạch hơ n liên kết cứng. Sử dụng liên kết cứng hoàn toàn minh bạch; trên thực tế, phải mất công sức để tìm ra một tệp đư ợc liên kết nhiều hơ n một lần!
Mặt khác, thao tác liên kết tư ợng trư ng đòi hỏi các lệnh gọi hệ thống đặc biệt. Sự thiếu minh bạch này thư ờng đư ợc coi là tích cực, với các liên kết tư ợng trư ng hoạt động như các phím tắt hơ n là các liên kết nội bộ hệ thống tệp.

Các tệp đặc

biệt Các tệp đặc biệt là các đối tư ợng hạt nhân đư ợc biểu diễn dư ới dạng tệp. Trong nhiều năm, các hệ thống Unix đã hỗ trợ một số tệp đặc biệt khác nhau. Linux hỗ trợ bốn tệp: tệp thiết bị khối, tệp thiết bị ký tự, đư ờng ống đư ợc đặt tên và ổ cắm miền Unix. Các tệp đặc biệt là một cách để cho phép một số trừu tư ợng nhất định phù hợp với hệ thống tệp, tham gia vào mô hình mọi thứ đều là tệp. Linux cung cấp lệnh gọi hệ thống để tạo tệp đặc biệt.

Truy cập thiết bị trong hệ thống Unix đư ợc thực hiện thông qua các tệp thiết bị, hoạt động và trông giống như các tệp bình thư ờng nằm trên hệ thống tệp. Các tệp thiết bị có thể đư ợc mở, đọc từ và ghi vào, cho phép không gian ngư ời dùng truy cập và thao tác các thiết bị (cả vật lý và ảo) trên hệ thống. Các thiết bị Unix thư ờng đư ợc chia thành hai nhóm: thiết bị ký tự và thiết bị khối. Mỗi loại thiết bị có tệp thiết bị đặc biệt riêng.

Thiết bị ký tự đư ợc truy cập như một hàng đợi tuyến tính của các byte. Trình điều khiển thiết bị đặt các byte vào hàng đợi, từng byte một, và không gian ngư ời dùng đọc các byte theo thứ tự mà chúng đư ợc đặt vào hàng đợi. Bàn phím là một ví dụ về thiết bị ký tự. Ví dụ, nếu ngư ời dùng nhập "peg", ứng dụng sẽ muốn đọc từ thiết bị bàn phím các ký tự p, e và cuối cùng là g. Khi không còn ký tự nào để đọc nữa, thiết bị sẽ trả về kết thúc tệp (EOF). Việc thiếu một ký tự hoặc đọc chúng theo bất kỳ thứ tự nào khác sẽ không có nhiều ý nghĩa. Thiết bị ký tự đư ợc truy cập thông qua các tệp thiết bị ký tự.

Ngư ợc lại, một thiết bị khối đư ợc truy cập như một mảng byte. Trình điều khiển thiết bị ánh xạ các byte trên một thiết bị có thể tìm kiếm và không gian ngư ời dùng đư ợc tự do truy cập bất kỳ byte hợp lệ nào trong mảng, theo bất kỳ thứ tự nào—nó có thể đọc byte 12, sau đó là byte 7 và sau đó là byte 12 một lần nữa. Thiết bị khối thư ờng là thiết bị lư u trữ. Đĩa cứng, ổ đĩa mềm, ổ đĩa CD-ROM và bộ nhớ flash đều là ví dụ về thiết bị khối. Chúng đư ợc truy cập thông qua các tệp thiết bị khối.

Ống dẫn có tên (thư ờng đư ợc gọi là FIFO, viết tắt của “vào trư ớc, ra trư ớc”) là một cơ chế giao tiếp giữa các tiến trình (IPC) cung cấp một kênh giao tiếp qua một mô tả tệp, đư ợc truy cập thông qua một tệp đặc biệt. Ống dẫn thông thư ờng là phư ơ ng pháp đư ợc sử dụng để “dẫn” đầu ra của một chư ơ ng trình vào đầu vào của một chư ơ ng trình khác; chúng đư ợc tạo trong bộ nhớ thông qua một lệnh gọi hệ thống và không tồn tại trên bất kỳ hệ thống tệp nào. Ống dẫn có tên hoạt động giống như ống dẫn thông thư ờng, như ng đư ợc truy cập thông qua một tệp, đư ợc gọi là tệp đặc biệt FIFO. Các tiến trình không liên quan có thể truy cập tệp này và giao tiếp.

Socket là loại tệp đặc biệt cuối cùng. Socket là một dạng IPC nâng cao cho phép giao tiếp giữa hai quy trình khác nhau, không chỉ trên cùng một máy mà còn trên hai máy khác nhau. Trên thực tế, socket tạo thành cơ sở của mạng

www.it-ebooks.info

Machine Translated by Google

và lập trình Internet. Chúng có nhiều loại, bao gồm socket miền Unix, là dạng socket đư ợc sử dụng để giao tiếp trong máy cục bộ. Trong khi các socket giao tiếp qua Internet có thể sử dụng cặp tên máy chủ và cổng để xác định mục tiêu giao tiếp, socket miền Unix sử dụng một tệp đặc biệt nằm trên hệ thống tệp, thư ờng đư ợc gọi đơ n giản là tệp socket.

Hệ thống tập tin và không gian

tên Linux, giống như tất cả các hệ thống Unix, cung cấp một không gian tên toàn cục và thống nhất cho các tập tin và thư mục. Một số hệ điều hành tách các đĩa và ổ đĩa khác nhau thành các không gian tên riêng biệt—ví dụ, một tệp trên đĩa mềm có thể truy cập đư ợc thông qua đư ờng dẫn A:\plank.jpg, trong khi ổ cứng nằm ở C:\. Trong Unix, cùng một tệp đó trên đĩa mềm có thể truy cập đư ợc thông qua đư ờng dẫn /media/floppy/plank.jpg, hoặc thậm chí thông qua /home/captain/stuff/plank.jpg, ngay bên cạnh các tệp từ phư ơ ng tiện khác. Nghĩa là, trên Unix, không gian tên đư ợc thống nhất.

Hệ thống tập tin là tập hợp các tập tin và thư mục theo hệ thống phân cấp chính thức và hợp lệ. Hệ thống tập tin có thể đư ợc thêm vào và xóa riêng lẻ khỏi không gian tên toàn cục của các tập tin và thư mục. Các hoạt động này đư ợc gọi là gắn kết và hủy gắn kết. Mỗi hệ thống tập tin đư ợc gắn kết vào một vị trí cụ thể trong không gian tên, đư ợc gọi là điểm gắn kết.
Thư mục gốc của hệ thống tập tin sau đó có thể truy cập đư ợc tại điểm gắn kết này. Ví dụ, một đĩa CD có thể đư ợc gắn kết tại /media/cdrom, làm cho gốc của hệ thống tập tin trên đĩa CD có thể truy cập đư ợc tại điểm gắn kết đó. Hệ thống tập tin đầu tiên đư ợc gắn kết nằm trong gốc của không gian tên, /, và đư ợc gọi là hệ thống tập tin gốc. Các hệ thống Linux luôn có một hệ thống tập tin gốc. Việc gắn kết các hệ thống tập tin khác tại các điểm gắn kết khác là tùy chọn.

Hệ thống tập tin thư ờng tồn tại vật lý (tức là đư ợc lư u trữ trên đĩa), mặc dù Linux cũng hỗ trợ các hệ thống tập tin ảo chỉ tồn tại trong bộ nhớ và các hệ thống tập tin mạng tồn tại trên các máy trên toàn mạng. Các hệ thống tập tin vật lý nằm trên các thiết bị lư u trữ khối, chẳng hạn như CD, đĩa mềm, thẻ nhớ flash nhỏ gọn hoặc ổ cứng. Một số thiết bị như vậy có thể phân vùng, nghĩa là chúng có thể đư ợc chia thành nhiều hệ thống tập tin, tất cả đều có thể đư ợc thao tác riêng lẻ. Linux hỗ trợ nhiều hệ thống tập tin khác nhau—chắc chắn là bất kỳ thứ gì mà ngư ời dùng trung bình có thể hy vọng bắt gặp—bao gồm các hệ thống tập tin dành riêng cho phư ơ ng tiện (ví dụ: ISO9660), hệ thống tập tin mạng (NFS), hệ thống tập tin gốc (ext3), hệ thống tập tin từ các hệ thống Unix khác (XFS) và thậm chí cả hệ thống tập tin từ các hệ thống không phải Unix (FAT).

Đơ n vị có thể định địa chỉ nhỏ nhất trên một thiết bị khối là sector. Sector là một đặc tính vật lý của thiết bị. Sector có nhiều dạng lũy thừa của hai, trong đó 512 byte là khá phổ biến. Một thiết bị khối không thể truyền hoặc truy cập một đơ n vị dữ liệu nhỏ hơ n một sector; tất cả I/O đều diễn ra theo một hoặc nhiều sector.

www.it-ebooks.info

Machine Translated by Google

Tương tự như vậy, đơn vị nhỏ nhất có thể định địa chỉ logic trên một hệ thống tập tin là khối. Khối là một sự trừu tượng của hệ thống tập tin, không phải của phương tiện vật lý mà hệ thống tập tin nằm trên đó. Một khối thường là bội số lũy thừa của hai của kích thước sector. Các khối thường lớn hơn sector, nhưng chúng phải nhỏ hơn kích thước trang* (đơn vị nhỏ nhất có thể định địa chỉ bởi đơn vị quản lý bộ nhớ, một thành phần phần cứng).
Kích thước khối phổ biến là 512 byte, 1 kilobyte và 4 kilobyte.

Theo truyền thống, các hệ thống Unix chỉ có một không gian tên dùng chung duy nhất, có thể xem được bởi tất cả người dùng và tất cả các quy trình trên hệ thống. Linux áp dụng một cách tiếp cận sáng tạo và hỗ trợ không gian tên cho mỗi quy trình, cho phép mỗi quy trình tùy chọn có một chế độ xem duy nhất về hệ thống phân cấp tệp và thư mục của hệ thống.† Theo mặc định, mỗi quy trình kế thừa không gian tên của quy trình cha, nhưng một quy trình có thể chọn tạo không gian tên riêng với tập hợp các điểm gắn kết riêng và một thư mục gốc duy nhất.

Các quy trình

Nếu tệp là sự trừu tượng cơ bản nhất trong hệ thống Unix, thì quy trình là thứ cơ bản thứ hai. Quy trình là mã đối tượng đang thực thi: các chương trình đang hoạt động, còn sống, đang chạy. Nhưng chúng không chỉ là mã đối tượng—quy trình bao gồm dữ liệu, tài nguyên, trạng thái và máy tính ảo.

Các tiến trình bắt đầu cuộc sống như mã đối tượng có thể thực thi, là mã có thể chạy bằng máy theo định dạng có thể thực thi mà hạt nhân hiểu được (định dạng phổ biến nhất trong Linux là ELF). Định dạng có thể thực thi chứa siêu dữ liệu và nhiều phần mã và dữ liệu. Các phần là các khối tuyến tính của mã đối tượng được tải vào các khối tuyến tính của bộ nhớ. Tất cả các byte trong một phần được xử lý giống nhau, được cấp cùng các quyền và thường được sử dụng cho các mục đích tương tự.

Các phần quan trọng và phổ biến nhất là phần văn bản, phần dữ liệu và phần bss. Phần văn bản chứa mã thực thi và dữ liệu chỉ đọc, chẳng hạn như các biến hằng số và thường được đánh dấu là chỉ đọc và thực thi. Phần dữ liệu chứa dữ liệu đã khởi tạo, chẳng hạn như các biến C có giá trị được xác định và thường được đánh dấu là có thể đọc và ghi. Phần bss chứa dữ liệu toàn cục chưa được khởi tạo. Vì chuẩn C chỉ định các giá trị mặc định cho các biến C về cơ bản là tất cả các số không, nên không cần lưu trữ các số không trong mã đối tượng trên đĩa. Thay vào đó, mã đối tượng có thể chỉ cần liệt kê các biến chưa được khởi tạo trong phần bss và hạt nhân có thể ánh xạ trang số không (một trang gồm tất cả các số không) trên phần này khi nó được tải vào bộ nhớ. Phần bss được hình thành chỉ như một tối ưu hóa cho mục đích này. Tên là một di tích lịch sử; nó là viết tắt của khối bắt đầu bằng ký hiệu hoặc phân đoạn lưu trữ khối. Các phần phổ biến khác trong tệp thực thi ELF là phần tuyệt đối (chứa các ký hiệu không thể di chuyển) và phần không xác định (một phần chung).

* Đây là một hạn chế nhân tạo, nhằm mục đích đơn giản hóa, có thể sẽ biến mất trong tương lai. † Phương pháp này lần đầu tiên được Plan9 của Bell Labs tiên phong thực hiện.

www.it-ebooks.info

Machine Translated by Google

Một tiến trình cũng liên quan đến nhiều tài nguyên hệ thống khác nhau, được phân xử và quản lý bởi hạt nhân. Các tiến trình thường yêu cầu và thao tác tài nguyên chỉ thông qua các lệnh gọi hệ thống. Tài nguyên bao gồm bộ đếm thời gian, tín hiệu đang chờ, tệp mở, kết nối mạng, phần cứng và cơ chế IPC. Tài nguyên của một tiến trình, cùng với dữ liệu và số liệu thống kê liên quan đến tiến trình, được lưu trữ bên trong hạt nhân trong mô tả tiến trình của tiến trình.

Một tiến trình là một sự trừu tượng hóa ảo hóa. Nhân Linux, hỗ trợ cả đa nhiệm ưu tiên và bộ nhớ ảo, cung cấp cho tiến trình cả bộ xử lý ảo hóa và chế độ xem ảo hóa của bộ nhớ. Theo quan điểm của tiến trình, chế độ xem hệ thống giống như thể chỉ có tiến trình đó kiểm soát. Nghĩa là, mặc dù một tiến trình nhất định có thể được lên lịch cùng với nhiều tiến trình khác, nhưng tiến trình đó chạy như thể nó có toàn quyền kiểm soát hệ thống. Nhân liền mạch và minh bạch chiếm quyền trước và lên lịch lại các tiến trình, chia sẻ bộ xử lý của hệ thống giữa tất cả các tiến trình đang chạy. Các tiến trình không bao giờ biết sự khác biệt. Tương tự như vậy, mỗi tiến trình được cung cấp một không gian địa chỉ tuyến tính duy nhất, như thể chỉ có nó kiểm soát toàn bộ bộ nhớ trong hệ thống. Thông qua bộ nhớ ảo và phân trang, nhân cho phép nhiều tiến trình cùng tồn tại trên hệ thống, mỗi tiến trình hoạt động trong một không gian địa chỉ khác nhau. Nhân quản lý quá trình ảo hóa này thông qua hỗ trợ phần cứng do các bộ xử lý hiện đại cung cấp, cho phép hệ điều hành đồng thời quản lý trạng thái của nhiều tiến trình độc lập.

Chủ đề

Mỗi tiến trình bao gồm một hoặc nhiều luồng thực thi (thường chỉ gọi là luồng). Luồng là đơn vị hoạt động trong một tiến trình, là phần trừu tượng chịu trách nhiệm thực thi mã và duy trì trạng thái chạy của tiến trình.

Hầu hết các tiến trình chỉ bao gồm một luồng duy nhất; chúng được gọi là luồng đơn. Các tiến trình chứa nhiều luồng được gọi là đa luồng. Theo truyền thống, các chương trình Unix là luồng đơn, do tính đơn giản lịch sử của Unix, thời gian tạo tiến trình nhanh và cơ chế IPC mạnh mẽ, tất cả đều làm giảm nhu cầu về luồng.

Một luồng bao gồm một ngăn xếp (lưu trữ các biến cục bộ của nó, giống như ngăn xếp quy trình trên các hệ thống không có luồng), trạng thái bộ xử lý và vị trí hiện tại trong mã đối tượng (thường được lưu trữ trong con trỏ lệnh của bộ xử lý). Phần lớn các phần còn lại của một quy trình được chia sẻ giữa tất cả các luồng.

Về mặt nội bộ, hạt nhân Linux triển khai một chế độ xem luồng duy nhất: chúng chỉ đơn giản là các tiến trình bình thường tình cờ chia sẻ một số tài nguyên (đáng chú ý nhất là không gian địa chỉ). Trong không gian người dùng, Linux triển khai luồng theo POSIX 1003.1c (được gọi là pthreads). Tên của triển khai luồng Linux hiện tại, là một phần của glibc, là Native POSIX Threading Library (NPTL).

Phân cấp quy trình

Mỗi quy trình được xác định bằng một số nguyên dương duy nhất được gọi là ID quy trình (pid). PID của quy trình đầu tiên là 1 và mỗi quy trình tiếp theo nhận được một pid mới, duy nhất.

www.it-ebooks.info

Machine Translated by Google

Trong Linux, các tiến trình tạo thành một hệ thống phân cấp chặt chẽ, đư ợc gọi là cây tiến trình. Cây tiến trình đư ợc bắt nguồn từ tiến trình đầu tiên, đư ợc gọi là tiến trình init, thư ờng là chư ơ ng trình init(8) . Các tiến trình mới đư ợc tạo thông qua lệnh gọi hệ thống fork() . Lệnh gọi hệ thống này tạo ra một bản sao của tiến trình gọi. Tiến trình gốc đư ợc gọi là tiến trình cha; tiến trình mới đư ợc gọi là tiến trình con. Mọi tiến trình ngoại trừ tiến trình đầu tiên đều có một tiến trình cha. Nếu một tiến trình cha kết thúc trư ớc tiến trình con của nó, thì hạt nhân sẽ chuyển đổi tiến trình con thành tiến trình init.

Khi một tiến trình kết thúc, nó không bị xóa ngay khỏi hệ thống. Thay vào đó, hạt nhân giữ các phần của tiến trình lư u trú trong bộ nhớ, để cho phép tiến trình cha của tiến trình hỏi về trạng thái của nó khi kết thúc. Điều này đư ợc gọi là chờ tiến trình đã kết thúc. Sau khi tiến trình cha đã chờ tiến trình con đã kết thúc của nó, tiến trình con sẽ bị hủy hoàn toàn. Một tiến trình đã kết thúc, như ng vẫn chư a đư ợc chờ đợi, đư ợc gọi là zombie. Tiến trình init thư ờng xuyên chờ tất cả các tiến trình con của nó, đảm bảo rằng các tiến trình đư ợc cha lại không mãi mãi là zombie.

Quyền hạn của ngư ời dùng

và nhóm trong Linux đư ợc cung cấp bởi ngư ời dùng và nhóm. Mỗi ngư ời dùng đư ợc liên kết với một số nguyên dư ơ ng duy nhất đư ợc gọi là ID ngư ời dùng (uid). Mỗi quy trình lần lư ợt đư ợc liên kết với chính xác một uid, xác định ngư ời dùng đang chạy quy trình và đư ợc gọi là uid thực của quy trình. Bên trong hạt nhân Linux, uid là khái niệm duy nhất về ngư ời dùng. Tuy nhiên, bản thân ngư ời dùng tham chiếu đến bản thân và những ngư ời dùng khác thông qua tên ngư ời dùng, không phải giá trị số. Tên ngư ời dùng và uid tư ơ ng ứng của chúng đư ợc lư u trữ trong /etc/passwd và các thói quen thư viện ánh xạ tên ngư ời dùng do ngư ời dùng cung cấp với các uid tư ơ ng ứng.

Trong quá trình đăng nhập, ngư ời dùng cung cấp tên ngư ời dùng và mật khẩu cho chư ơ ng trình login(1) . Nếu đư ợc cung cấp tên ngư ời dùng hợp lệ và mật khẩu đúng, chư ơ ng trình login(1) sẽ tạo ra shell đăng nhập của ngư ời dùng , cũng đư ợc chỉ định trong /etc/passwd và làm cho uid của shell bằng với uid của ngư ời dùng. Các tiến trình con kế thừa uid của các tiến trình cha mẹ.

Uid 0 đư ợc liên kết với một ngư ời dùng đặc biệt đư ợc gọi là root. Ngư ời dùng root có các đặc quyền đặc biệt và có thể làm hầu như mọi thứ trên hệ thống. Ví dụ, chỉ có ngư ời dùng root mới có thể thay đổi uid của một tiến trình. Do đó, chư ơ ng trình login(1) chạy dư ới dạng root.

Ngoài uid thực, mỗi tiến trình cũng có một uid hiệu dụng, một uid đã lư u và một uid hệ thống tệp. Trong khi uid thực luôn là uid của ngư ời dùng đã bắt đầu tiến trình, uid hiệu dụng có thể thay đổi theo nhiều quy tắc khác nhau để cho phép một tiến trình thực thi với các quyền của những ngư ời dùng khác nhau. Uid đã lư u lư u trữ uid hiệu dụng gốc; giá trị của uid này đư ợc sử dụng để quyết định giá trị uid hiệu dụng nào mà ngư ời dùng có thể chuyển sang. Uid hệ thống tệp, thư ờng bằng uid hiệu dụng, đư ợc sử dụng để xác minh quyền truy cập hệ thống tệp.

Mỗi ngư ời dùng có thể thuộc về một hoặc nhiều nhóm, bao gồm nhóm chính hoặc nhóm đăng nhập, đư ợc liệt kê trong /etc/passwd và có thể là một số nhóm bổ sung, đư ợc liệt kê trong /etc/group. Do đó, mỗi quy trình cũng đư ợc liên kết với một ID nhóm tư ơ ng ứng (gid) và có một gid thực, một gid hiệu quả, một gid đã lư u và một gid hệ thống tệp. Các quy trình thư ờng đư ợc liên kết với nhóm đăng nhập của ngư ời dùng, không phải bất kỳ nhóm bổ sung nào.

www.it-ebooks.info

Machine Translated by Google

Một số kiểm tra bảo mật cho phép các quy trình thực hiện một số hoạt động nhất định chỉ khi chúng đáp ứng các tiêu chí cụ thể. Theo truyền thống, Unix đã đưa ra quyết định này rất rõ ràng: các tiến trình có uid 0 đã có quyền truy cập, trong khi không có tiến trình nào khác có quyền truy cập. Gần đây, Linux đã thay thế hệ thống an ninh này với một hệ thống khả năng chung hơn . Thay vì một hệ thống đơn giản kiểm tra nhị phân, khả năng cho phép hạt nhân dựa trên quyền truy cập chi tiết hơn nhiều cài đặt.

Quyền

Cơ chế bảo mật và quyền tệp tiêu chuẩn trong Linux giống như trong Unix lịch sử.

Mỗi tệp được liên kết với một người dùng sở hữu, một nhóm sở hữu và một tập hợp các bit quyền. Các bit mô tả khả năng của người dùng sở hữu, nhóm sở hữu và mọi người khác đọc, viết và thực thi tệp; có ba bit cho mỗi ba lớp, tạo thành tổng cộng chín bit. Chủ sở hữu và quyền được lưu trữ trong inode của tập tin.

Đối với các tệp thông thường, các quyền khá rõ ràng: chúng chỉ định khả năng mở một tập tin để đọc, mở tập tin để ghi hoặc thực thi tập tin. Quyền đọc và ghi giống nhau đối với các tệp đặc biệt như đối với các tệp thông thường, mặc dù chính xác những gì được đọc hoặc ghi là tùy thuộc vào tệp đặc biệt đang đề cập. Quyền thực thi bị bỏ qua đối với các tệp đặc biệt tập tin. Đối với thư mục, quyền đọc cho phép liệt kê nội dung của thư mục, quyền ghi cho phép thêm các liên kết mới vào bên trong thư mục và quyền thực thi cho phép nhập và sử dụng thư mục trong một tên đường dẫn. Bảng 1-1 liệt kê từng quyền trong chín bit quyền, giá trị bát phân của chúng (một cách phổ biến để biểu diễn chín bit bit), giá trị văn bản của chúng (như ls có thể hiển thị) và ý nghĩa tương ứng của chúng.

Bảng 1-1. Các bit quyền và giá trị của chúng

Chút	Giá trị bát phân	Giá trị văn bản	Quyền tương ứng
8	400	r--------	Chủ sở hữu có thể đọc
7	200	-w-------	Chủ sở hữu có thể viết
6	100	--x------	Chủ sở hữu có thể thực hiện
5	040	---r-----	Nhóm có thể đọc
4	020	----w----	Nhóm có thể viết
3	010	-----x---	Nhóm có thể thực hiện
2	004	------đ--	Mọi người khác có thể đọc
1	002	-------w-	Mọi người khác có thể viết
0	001	--------x	Mọi người khác có thể thực hiện

Ngoài các quyền Unix lịch sử, Linux cũng hỗ trợ danh sách kiểm soát truy cập (ACL). ACL cho phép cấp phép và bảo mật chi tiết và chính xác hơn nhiều kiểm soát, với cái giá phải trả là tăng độ phức tạp và lưu trữ trên đĩa.

www.it-ebooks.info

Machine Translated by Google

Signals

Signals là cơ chế thông báo không đồng bộ một chiều. Một tín hiệu có thể được gửi từ nhân đến một tiến trình, từ một tiến trình đến một tiến trình khác hoặc từ một tiến trình đến chính nó. Signals thường cảnh báo một tiến trình về một số sự kiện, chẳng hạn như lỗi phân đoạn hoặc người dùng nhấn Ctrl-C.

Nhân Linux triển khai khoảng 30 tín hiệu (con số chính xác phụ thuộc vào kiến trúc). Mỗi tín hiệu được biểu diễn bằng một hằng số và một tên văn bản.
Ví dụ, SIGHUP, được dùng để báo hiệu thiết bị đầu cuối đã bị treo, có giá trị là 1 trên kiến trúc i386.

Ngoại trừ SIGKILL (luôn kết thúc tiến trình) và SIGSTOP (luôn dừng tiến trình), các tiến trình có thể kiểm soát những gì xảy ra khi chúng nhận được tín hiệu. Chúng có thể chấp nhận hành động mặc định, có thể là kết thúc tiến trình, kết thúc và coredump tiến trình, dừng tiến trình hoặc không làm gì cả, tùy thuộc vào tín hiệu. Ngoài ra, các tiến trình có thể chọn bỏ qua hoặc xử lý tín hiệu một cách rõ ràng. Các tín hiệu bị bỏ qua sẽ bị loại bỏ một cách âm thầm. Các tín hiệu được xử lý sẽ gây ra việc thực thi hàm xử lý tín hiệu do người dùng cung cấp . Chương trình nhảy đến hàm này ngay khi nhận được tín hiệu và (khi trình xử lý tín hiệu trả về) quyền điều khiển chương trình sẽ tiếp tục tại lệnh bị ngắt trước đó.

Giao tiếp giữa các tiến trình Cho

phép các tiến trình trao đổi thông tin và thông báo cho nhau về các sự kiện là một trong những công việc quan trọng nhất của hệ điều hành. Nhân Linux triển khai hầu hết các cơ chế Unix IPC lịch sử—bao gồm cả những cơ chế được định nghĩa và chuẩn hóa bởi cả System V và POSIX—cũng như triển khai một hoặc hai cơ chế của riêng nó.

Các cơ chế IPC được Linux hỗ trợ bao gồm các đường ống, đường ống được đặt tên, semaphore, hàng đợi tin nhắn, bộ nhớ chia sẻ và futex.

Tiêu đề

Lập trình hệ thống Linux xoay quanh một số ít tiêu đề. Cả hạt nhân và glibc đều cung cấp các tiêu đề được sử dụng trong lập trình cấp hệ thống. Các tiêu đề này bao gồm giá vé C chuẩn (ví dụ: <string.h>) và các dịch vụ Unix thông thường (ví dụ: <unistd.h>).

Xử lý lỗi Không cần

phải nói rằng việc kiểm tra và xử lý lỗi là vô cùng quan trọng. Trong lập trình hệ thống, lỗi được biểu thị thông qua giá trị trả về của hàm và được mô tả thông qua một biến đặc biệt, errno. glibc cung cấp hỗ trợ errno một cách minh bạch cho cả lệnh gọi thư viện và hệ thống. Phần lớn các giao diện được đề cập trong cuốn sách này sẽ sử dụng cơ chế này để truyền đạt lỗi.

www.it-ebooks.info

Machine Translated by Google

Các hàm thông báo cho ngư ời gọi về lỗi thông qua giá trị trả về đặc biệt, thư ờng là -1 (giá trị chính xác đư ợc sử dụng phụ thuộc vào chức năng). Giá trị lỗi cảnh báo ngư ời gọi sự xuất hiện của một lỗi, như ng không cung cấp thông tin chi tiết về lý do tại sao lỗi xảy ra. Biến errno đư ợc sử dụng để tìm nguyên nhân gây ra lỗi.

Biến này đư ợc định nghĩa trong <errno.h> như sau:

```
extern int errno;
```

Giá trị của nó chỉ có hiệu lực ngay sau khi hàm errno-setting chỉ ra lỗi (thư ờng bằng cách trả về -1), vì biến có thể đư ợc sửa đổi trong quá trình thực thi thành công một hàm.

Biến errno có thể đư ợc đọc hoặc ghi trực tiếp; nó là một lvalue có thể sửa đổi. giá trị của errno ánh xạ tới mô tả văn bản của một lỗi cụ thể. Một bộ tiền xử lý #define cũng ánh xạ tới giá trị errno số . Ví dụ, bộ tiền xử lý xác định EACCESS bằng 1 và biểu thị "quyền bị từ chối". Xem Bảng 1-2 để biết danh sách tiêu chuẩn xác định và mô tả lỗi tư ơng ứng.

Bảng 1-2. Lỗi và mô tả của chúng

Bộ tiền xử lý xác định	Sự miêu tả
E2BIG	Danh sách đối số quá dài
TRUY CẬP	Quyền bị từ chối
LẠI LẦN NỮA	Thử lại
EBADF	Số tập tin không hợp lệ
EBUSY	Thiết bị hoặc tài nguyên đang bận
TRẺ EM	Không có tiến trình con
EDOM	Đối số toán học nằm ngoài miền của hàm
Thoát	Tập tin đã tồn tại
MẶC ĐỊNH	Địa chỉ không đúng
EFBIG	Tệp quá lớn
TRỰC TIẾP	Cuộc gọi hệ thống đã bị gián đoạn
EINVAL	Lập luận không hợp lệ
EIO	Lỗi I/O
EISDR	Là một thư mục
TỆP EM	Quá nhiều tập tin mở
Liên kết EM	Quá nhiều liên kết
NỘP HỒ SƠ	Bảng tập tin tràn
ENODEV	Không có thiết bị như vậy
ENOENT	Không có tập tin hoặc thư mục như vậy
ENOEXEC	Lỗi định dạng Exec
ENOMEM	Hết bộ nhớ
ENOSPC	Không còn chỗ trống trên thiết bị

www.it-ebooks.info

Machine Translated by Google

Bảng 1-2. Lỗi và mô tả của chúng (tiếp theo)

Bộ tiền xử lý xác định	Sự miêu tả
ENOTDIR	Không phải là một thư mục
ENOTTY	Hoạt động điều khiển I/O không phù hợp
ENXIO	Không có thiết bị hoặc địa chỉ như vậy
EPERM	Hoạt động không được phép
TỔNG HỢP	Ống bị vỡ
KHOẢNG CÁCH	Kết quả quá lớn
EROFS	Hệ thống tập tin chỉ đọc
TRÍCH DẪN	Tìm kiếm không hợp lệ
CÂU CHUYỆN	Không có quá trình như vậy
ETXTBSY	Tệp văn bản đang bận
PHÁT TRIỂN	Liên kết không đúng

Thư viện C cung cấp một số hàm để dịch giá trị errno sang
biểu diễn văn bản tương ứng. Điều này chỉ cần thiết cho báo cáo lỗi và
giống như ; việc kiểm tra và xử lý lỗi có thể được thực hiện bằng cách sử dụng bộ tiền xử lý xác định và
errno trực tiếp.

Hàm đầu tiên như vậy là perror( ):

```
#include <stdio.h>

void perror (const char *str);
```

Hàm này in ra stderr (lỗi chuẩn) chuỗi biểu diễn của hiện tại
lỗi được mô tả bởi errno, được thêm tiền tố là chuỗi được trỏ tới bởi str, theo sau là a
dấu hai chấm. Để hữu ích, tên của hàm bị lỗi phải được đưa vào
chuỗi. Ví dụ:

```
nếu (đóng (fd) == -1)
        perror ("đóng");
```

Thư viện C cũng cung cấp strerror() và strerror_r( ) , có nguyên mẫu như sau:

```
#include <chuỗi.h>

char * strerror (int errnum);
```

Và:

```
#include <chuỗi.h>

int strerror_r (int errnum, char *buf, size_t len);
```

Hàm trước trả về một con trỏ tới một chuỗi mô tả lỗi do errnum đưa ra.
Chuỗi không thể được ứng dụng sửa đổi, nhưng có thể được sửa đổi bằng các lệnh gọi
perror() và strerror() tiếp theo . Theo cách này, nó không an toàn cho luồng.

www.it-ebooks.info

Machine Translated by Google

Hàm strerror_r() an toàn với luồng. Nó điền vào bộ đệm có độ dài len được trỏ tới bởi buf. Một lệnh gọi đến strerror_r() trả về 0 nếu thành công và -1 nếu thất bại. Thật buồn cười, nó đặt errno khi có lỗi.

Đối với một số hàm, toàn bộ phạm vi của kiểu trả về là giá trị trả về hợp lệ. Trong những trường hợp đó, errno phải được đặt thành 0 trước khi gọi và được kiểm tra sau đó (các hàm này hứa sẽ chỉ trả về errno khác 0 khi có lỗi thực tế). Ví dụ:

```
errno = 0;
arg = strtoul (buf, NULL, 0); nếu
(errno)
        perror ("strtoul");
```

Một lỗi thường gặp khi kiểm tra errno là quên rằng bất kỳ thư viện hoặc lệnh gọi hệ thống nào cũng có thể sửa đổi nó. Ví dụ, đoạn mã này có lỗi:

```
nếu (fsync (fd) == -1)
          { fprintf (stderr, "fsync không thành
          công!\n"); nếu
                   (errno == EIO) fprintf (stderr, "Lỗi I/O trên %d!\n", fd);
}
```

Nếu bạn cần giữ nguyên giá trị của errno trong các lần gọi hàm, hãy lưu nó:

```
if (fsync (fd) == -1)
          { int err = errno;
          fprintf (stderr, "fsync không thành công: %s\n", strerror
          (errno)); if (err
                   == EIO) { /* nếu lỗi liên quan đến I/O, hãy thoát
                   */ fprintf (stderr, "Lỗi I/O trên %d!\n", fd);
                   thoát (EXIT_FAILURE);
          }
}
```

Trong các chương trình luồng đơn, errno là một biến toàn cục, như đã trình bày trước đó trong phần này. Tuy nhiên, trong các chương trình luồng đa, errno được lưu trữ theo luồng và do đó an toàn cho luồng.

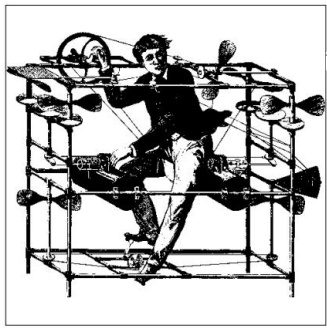
Bắt đầu với Lập trình Hệ thống

Chương này xem xét các nguyên tắc cơ bản của lập trình hệ thống Linux và cung cấp tổng quan của lập trình viên về hệ thống Linux. Chương tiếp theo thảo luận về I/O tệp cơ bản. Tất nhiên, điều này bao gồm đọc và ghi vào tệp; tuy nhiên, vì Linux triển khai nhiều giao diện dưới dạng tệp, nên I/O tệp rất quan trọng đối với nhiều thứ hơn là chỉ tệp.

Sau khi hoàn thành các bước chuẩn bị, chúng ta sẽ thấy ngày càng ngắn lại, đã đến lúc bắt đầu lập trình hệ thống thực tế. Bắt đầu thôi!

www.it-ebooks.info

Machine Translated by Google



CHƯƠNG 2

Tệp I/O

Chương này trình bày những điều cơ bản về đọc và ghi từ tệp. Các hoạt động như vậy tạo thành cốt lõi của hệ thống Unix. Chương tiếp theo trình bày về I/O chuẩn từ thư viện C chuẩn, và Chương 4 tiếp tục trình bày bằng cách xử lý các giao diện I/O tệp chuyên biệt và nâng cao hơn. Chương 7 kết thúc cuộc thảo luận bằng cách giải quyết chủ đề thao tác tệp và thư mục.

Trước khi có thể đọc hoặc ghi vào một tệp, tệp đó phải được mở. Nhân duy trì một danh sách các tệp đang mở theo từng quy trình, được gọi là bảng tệp. Bảng này được lập chỉ mục thông qua các số nguyên không âm được gọi là mô tả tệp (thường được viết tắt là fds). Mỗi mục trong danh sách chứa thông tin về một tệp đang mở, bao gồm một con trỏ đến bản sao trong bộ nhớ của inode sao lưu của tệp và siêu dữ liệu liên quan, chẳng hạn như vị trí tệp và chế độ truy cập. Cả không gian người dùng và không gian nhân đều sử dụng mô tả tệp làm cookie duy nhất cho từng quy trình. Việc mở tệp trả về một mô tả tệp, trong khi các hoạt động tiếp theo (đọc, ghi, v.v.) lấy mô tả tệp làm đối số chính của chúng.

Theo mặc định, một tiến trình con nhận được một bản sao của bảng tệp của tiến trình cha. Danh sách các tệp đang mở và chế độ truy cập của chúng, vị trí tệp hiện tại, v.v., đều giống nhau, nhưng một thay đổi trong một tiến trình—ví dụ, tiến trình con đóng tệp—không ảnh hưởng đến bảng tệp của tiến trình kia. Tuy nhiên, như bạn sẽ thấy trong Chương 5, tiến trình con và tiến trình cha có thể chia sẻ bảng tệp của tiến trình cha (như các luồng).

Các mô tả tệp được biểu diễn bằng kiểu C int . Việc không sử dụng một kiểu đặc biệt— ví dụ như fd_t —thường được coi là kỳ lạ, nhưng theo truyền thống, đó là cách của Unix. Mỗi quy trình Linux có số lượng tệp tối đa mà nó có thể mở. Các mô tả tệp bắt đầu từ 0 và tăng lên ít hơn một so với giá trị tối đa này. Theo mặc định, giá trị tối đa là 1.024, nhưng có thể được cấu hình cao tới 1.048.576. Vì các giá trị âm không phải là các mô tả tệp hợp lệ, nên –1 thường được sử dụng để chỉ ra lỗi từ một hàm mà nếu không thì sẽ trả về một mô tả tệp hợp lệ.

Trừ khi quy trình đóng chúng một cách rõ ràng, theo quy ước, mọi quy trình đều có ít nhất ba mô tả tệp mở: 0, 1 và 2. Mô tả tệp 0 là chuẩn vào (stdin), mô tả tệp 1 là chuẩn ra (stdout) và mô tả tệp 2 là chuẩn lỗi (stderr).

www.it-ebooks.info

Machine Translated by Google

Thay vì tham chiếu trực tiếp các số nguyên này, thư viện C cung cấp bộ tiền xử lý xác định STDIN_FILENO, STDOUT_FILENO và STDERR_FILENO.

Lưu ý rằng các mô tả tệp có thể tham chiếu đến nhiều thứ hơn là chỉ các tệp thông thường. Chúng được sử dụng để truy cập các tệp và đường ống thiết bị, thư mục và futex, FIFO và socket—theo triết lý mọi thứ đều là tệp, hầu như bất kỳ thứ gì bạn có thể đọc hoặc ghi đều có thể truy cập được thông qua mô tả tệp.

Mở tập tin

Phương pháp cơ bản nhất để truy cập tệp là thông qua lệnh gọi hệ thống read() và write() . Tuy nhiên, trước khi có thể truy cập một tệp, tệp đó phải được mở thông qua lệnh gọi hệ thống open() hoặc creat() . Sau khi sử dụng xong tệp, tệp đó phải được đóng lại bằng lệnh gọi hệ thống close( ).

Hệ thống gọi open( )

Một tệp được mở và mô tả tệp được lấy bằng lệnh gọi hệ thống open() :

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


int mở (const char *name, int flags); int
mở (const char *name, int flags, mode_t mode);
```

Lệnh gọi hệ thống open() ánh xạ tệp được cung cấp bởi pathname name tới một mô tả tệp, mô tả này sẽ trả về khi thành công. Vị trí tệp được đặt thành số không và tệp được mở để truy cập theo các cờ được cung cấp bởi flags.

Cờ cho open()

Đối số cờ phải là một trong các đối số O_RDONLY, O_WRONLY hoặc O_RDWR. Tương ứng, các đối số này yêu cầu tệp chỉ được mở để đọc, chỉ để ghi hoặc cả đọc và ghi.

Ví dụ, đoạn mã sau mở /home/kidd/madagascar để đọc:

```
số nguyên fd;


fd = mở ("/home/kidd/madagascar", O_RDONLY); nếu (fd == -1) /*
lỗi */
```

Một tệp chỉ được mở để ghi cũng không thể được đọc và ngược lại. Quá trình phát lệnh gọi hệ thống open() phải có đủ quyền để có được quyền truy cập được yêu cầu.

Đối số cờ có thể được OR bit với một hoặc nhiều giá trị sau, sửa đổi hành vi của yêu cầu mở:

www.it-ebooks.info

Machine Translated by Google

O_APPEND

Tệp sẽ đư ợc mở ở chế độ append. Nghĩa là, trư ớc mỗi lần ghi, vị trí tệp sẽ đư ợc cập nhật để trỏ đến cuối tệp. Điều này xảy ra ngay cả khi một tiến trình khác đã ghi vào tệp sau lần ghi cuối cùng của tiến trình phát hành, do đó thay đổi vị trí tệp. (Xem “Chế độ Append” sau trong chư ơ ng này).

O_ASYNC

Một tín hiệu (mặc định là SIGIO) sẽ đư ợc tạo ra khi tệp đư ợc chỉ định có thể đọc hoặc ghi đư ợc. Cờ này chỉ khả dụng cho các thiết bị đầu cuối và ổ cắm, không khả dụng cho các tệp thông thư ờng.

O_CREAT

Nếu tệp đư ợc biểu thị bằng tên không tồn tại, hạt nhân sẽ tạo tệp đó. Nếu tệp đã tồn tại, cờ này không có hiệu lực trừ khi O_EXCL cũng đư ợc cung cấp.

O_DIRECT

Tệp sẽ đư ợc mở để I/O trực tiếp (xem “I/O trực tiếp” ở phần sau của chư ơ ng này).

O_DIRECTORY

Nếu name không phải là thư mục, lệnh gọi open() sẽ không thành công. Cờ này đư ợc lệnh gọi thư viện opendir() sử dụng nội bộ .

O_EXCL

Khi đư ợc đư a ra với O_CREAT, cờ này sẽ khiến lệnh gọi open() thất bại nếu tệp đư ợc chỉ định theo tên đã tồn tại. Điều này đư ợc sử dụng để ngăn chặn tình trạng chạy đua khi tạo tệp.

O_TỆP LỚN

Tệp đã cho sẽ đư ợc mở bằng cách sử dụng offset 64 bit, cho phép mở các tệp lớn hơ n hai gigabyte. Điều này đư ợc ngụ ý trên kiến trúc 64 bit.

O_NOCTTY

Nếu tên đã cho tham chiếu đến một thiết bị đầu cuối (ví dụ: /dev/tty), nó sẽ không trở thành thiết bị đầu cuối điều khiển của quy trình, ngay cả khi quy trình hiện không có thiết bị đầu cuối điều khiển. Cờ này không đư ợc sử dụng thư ờng xuyên.

O_NOFOLLOW

Nếu name là liên kết tư ợng trư ng, lệnh gọi open() sẽ không thành công. Thông thư ờng, liên kết đư ợc giải quyết và tệp đích đư ợc mở. Nếu các thành phần khác trong đư ờng dẫn đã cho là liên kết, lệnh gọi vẫn sẽ thành công. Ví dụ, nếu name là /etc/ship/plank.txt, lệnh gọi sẽ không thành công nếu plank.txt là liên kết tư ợng trư ng. Tuy nhiên, lệnh gọi sẽ thành công nếu etc hoặc ship là liên kết tư ợng trư ng, miễn là plank.txt không phải.

O_NONBLOCK

Nếu có thể, tệp sẽ đư ợc mở ở chế độ không chặn. Cả lệnh gọi open() và bất kỳ thao tác nào khác đều không khiến quy trình chặn (ngủ) trên I/O. Hành vi này chỉ có thể đư ợc xác định cho FIFO.

www.it-ebooks.info

Machine Translated by Google

O_SYNC

Tệp sẽ đư ợc mở để I/O đồng bộ. Không có thao tác ghi nào hoàn tất cho đến khi dữ liệu đư ợc ghi vật lý vào đĩa; các thao tác đọc thông thư ờng đã đồng bộ, do đó cờ này không có tác dụng đối với các lần đọc. POSIX cũng định nghĩa thêm O_DSYNC và O_RSYNC; trên Linux, các cờ này đồng nghĩa với O_SYNC.
(Xem “Cờ O_SYNC” ở phần sau của chư ơ ng này.)

O_TRUNC

Nếu tệp tồn tại, đó là tệp thông thư ờng và các cờ cho phép ghi, tệp sẽ bị cắt bớt thành độ dài bằng không. Việc sử dụng O_TRUNC trên FIFO hoặc thiết bị đầu cuối bị bỏ qua. Việc sử dụng trên các loại tệp khác là không xác định. Việc chỉ định O_TRUNC với O_RDONLY cũng không xác định, vì bạn cần quyền ghi vào tệp để cắt bớt tệp.

Ví dụ, đoạn mã sau mở để ghi tệp /home/teach/pearl. Nếu tệp đã tồn tại, nó sẽ bị cắt bớt thành độ dài bằng không. Vì cờ O_CREAT không đư ợc chỉ định, nếu tệp không tồn tại, lệnh gọi sẽ không thành công:

```
số nguyên fd;

fd = mở ("/home/teach/pearl", O_WRONLY | O_TRUNC); nếu (fd == -1) /*
lỗi */
```

Chủ sở hữu của các tập tin mới

Việc xác định ngư ời dùng nào sở hữu tệp mới rất đơ n giản: uid của chủ sở hữu tệp là uid có hiệu lực của tiến trình tạo tệp.

Việc xác định nhóm sở hữu phức tạp hơ n. Hành vi mặc định là đặt nhóm của tệp thành gid hiệu quả của quy trình tạo tệp. Đây là hành vi của System V (mô hình hành vi cho phần lớn Linux) và là modus operandi chuẩn của Linux.

Tuy nhiên, để khó khăn hơ n, BSD đã định nghĩa hành vi riêng của nó: nhóm tệp đư ợc đặt thành gid của thư mục cha. Hành vi này khả dụng trên Linux thông qua tùy chọn mount-time*—đây cũng là hành vi sẽ xảy ra trên Linux theo mặc định nếu thư mục cha của tệp có bit set group ID (setgid) đư ợc đặt. Mặc dù hầu hết các hệ thống Linux sẽ sử dụng hành vi System V (trong đó các tệp mới nhận đư ợc gid của quy trình tạo), khả năng hành vi BSD (trong đó các tệp mới nhận đư ợc gid của thư mục cha) ngụ ý rằng mã thực sự quan tâm cần phải đặt thủ công nhóm thông qua lệnh gọi hệ thống chown() (xem Chư ơ ng 7).

May mắn thay, việc quan tâm đến nhóm sở hữu của một tệp là không phổ biến.

* Tùy chọn gắn kết bsdgroups hoặc sysvgroups.

www.it-ebooks.info

Machine Translated by Google

Quyền của các tập tin mới

Cả hai dạng lệnh gọi hệ thống open() đư ợc đư a ra trư ớc đó đều hợp lệ. Đối số mode bị bỏ qua trừ khi tệp đư ợc tạo; nó là bắt buộc nếu O_CREAT đư ợc đư a ra. Nếu bạn quên cung cấp đối số mode khi sử dụng O_CREAT, kết quả sẽ không xác định và thư ờng khá xấu— vì vậy đừng quên!

Khi một tệp đư ợc tạo, đối số mode cung cấp các quyền của tệp mới đư ợc tạo. Mode không đư ợc kiểm tra khi mở tệp cụ thể này, do đó bạn có thể thực hiện các thao tác trái ngư ợc nhau, chẳng hạn như mở tệp để ghi, như ng chỉ định quyền chỉ đọc cho tệp.

Đối số mode là bitset quyền Unix quen thuộc, chẳng hạn như octal 0644 (chủ sở hữu có thể đọc và ghi, mọi ngư ời khác chỉ có thể đọc). Về mặt kỹ thuật, POSIX cho phép các giá trị chính xác là cụ thể cho từng triển khai, cho phép các hệ thống Unix khác nhau bố trí các bit quyền theo bất kỳ cách nào họ muốn. Để bù đắp cho tính không thể chuyển đổi của các vị trí bit trong mode, POSIX đã giới thiệu bộ hằng số sau đây có thể đư ợc OR nhị phân với nhau và đư ợc cung cấp cho đối số mode :

S_IRWXU
: Chủ sở hữu có quyền đọc, ghi và thực thi.

S_IRUSR
: Chủ sở hữu đã đư ợc cấp quyền đọc.

S_IWUSR
: Chủ sở hữu có quyền ghi.

S_IXUSR
: Chủ sở hữu có quyền thực thi.

Nhóm
: S_IRWXG có quyền đọc, ghi và thực thi.

Nhóm
: S_IRGRP đã đư ợc cấp quyền đọc.

Nhóm
: S_IWGRP có quyền ghi.

Nhóm
: S_IXGRP có quyền thực thi.

S_IRWXO
: Mọi ngư ời khác đều có quyền đọc, ghi và thực thi.

S_IROTH
: Mọi ngư ời khác đều có quyền đọc.

www.it-ebooks.info

Machine Translated by Google

S_IWOTH

Mọi ngư ời khác đều có quyền ghi.

S_IXOTH

Mọi ngư ời khác đều có quyền thực thi.

Các bit quyền thực tế đánh vào đĩa đư ợc xác định bằng cách AND nhị phân đối số chế độ với phần bù của mặt nạ tạo tệp của ngư ời dùng (umask). Theo cách không chính thức, các bit trong umask bị tắt trong đối số chế độ đư ợc cung cấp cho open( ). Do đó, umask thông thư ờng của 022 sẽ khiến đối số chế độ của 0666 trở thành 0644 (0666 & ~022). Là một lập trình viên hệ thống, bạn thư ờng không xem xét umask khi thiết lập quyền—umask tồn tại để cho phép ngư ời dùng giới hạn các quyền mà chư ơ ng trình của anh ta thiết lập trên các tệp mới.

Ví dụ, đoạn mã sau đây mở tệp đư ợc cung cấp bởi file để ghi. Nếu tệp không tồn tại, giả sử umask là 022, tệp sẽ đư ợc tạo với quyền 0644 (mặc dù đối số mode chỉ định 0664). Nếu tệp tồn tại, tệp sẽ bị cắt bớt thành độ dài bằng không:

```
số nguyên fd;

fd = mở (tệp, O_WRONLY | O_CREAT | O_TRUNC,
               S_IWUSR | S_IRUSR | S_IWGRP | S_IRGRP | S_IROTH); nếu (fd == -1) /
* lỗi */
```

Hàm creat()

Sự kết hợp của O_WRONLY | O_CREAT | O_TRUNC rất phổ biến đến nỗi có một lệnh gọi hệ thống để cung cấp hành vi đó:

```
#include <sys/types.h> #include
<sys/stat.h> #include
<fcntl.h>

int creat (const char *name, mode_t chế độ);
```

Đúng vậy, tên của hàm này thiếu chữ e. Ken Thompson, ngư ời sáng tạo ra Unix, đã từng nói đùa rằng chữ cái bị thiếu là điều ông hối tiếc lớn nhất trong quá trình thiết kế Unix.

Sau đây là lệnh gọi creat() thông thư ờng :

```
số nguyên fd;

fd = tạo (tệp, 0644); nếu (fd ==
-1) /* lỗi */
```


www.it-ebooks.info

Machine Translated by Google

giống hệt với:

```
số nguyên fd;

fd = mở (tệp, O_WRONLY | O_CREAT | O_TRUNC, 0644); nếu (fd == -1) /* lỗi
*/
```

Trên hầu hết các kiến trúc Linux,* creat( ) là một lệnh gọi hệ thống, mặc dù nó có thể được triển khai trong không gian người dùng một cách đơn giản như sau:

```
int tạo (const char *tên, int chế độ) {

        trả về mở (tên, O_WRONLY | O_CREAT | O_TRUNC, chế độ);
}
```

Sự trùng lặp này là di tích lịch sử từ khi open() chỉ có hai đối số.
Ngày nay, lệnh gọi hệ thống creat() vẫn còn tồn tại để tương thích. Các kiến trúc mới có thể triển khai creat() như được hiển thị trong glibc.

Giá trị trả về và mã lỗi

Cả open() và creat() đều trả về một mô tả tệp khi thành công. Khi có lỗi, cả hai đều trả về -1 và đặt errno thành giá trị lỗi phù hợp (Chương 1 đã thảo luận về errno và liệt kê các giá trị lỗi tiềm ẩn). Xử lý lỗi khi mở tệp không phức tạp, vì nhìn chung sẽ có ít hoặc không có bước nào được thực hiện trước khi mở cần phải hoàn tác. Một phản hồi thông thường sẽ là nhắc người dùng nhập tên tệp khác hoặc chỉ cần kết thúc chương trình.

Đọc thông qua read()

Bây giờ bạn đã biết cách mở tệp, hãy cùng xem cách đọc tệp. Trong phần sau, chúng ta sẽ xem xét cách viết.

Cơ chế cơ bản nhất và phổ biến nhất được sử dụng để đọc là lệnh gọi hệ thống read() , được định nghĩa trong POSIX.1:

```
#include <unistd.h>

ssize_t đọc (int fd, void *buf, size_t len);
```

Mỗi lệnh gọi đọc tối đa len byte vào buf từ offset tệp hiện tại của tệp được fd tham chiếu . Khi thành công, số byte được ghi vào buf được trả về. Khi có lỗi, lệnh gọi trả về -1 và errno được đặt. Vị trí tệp được nâng cao theo số byte được đọc từ fd. Nếu đối tượng được fd biểu diễn không có khả năng tìm kiếm (ví dụ: tệp thiết bị ký tự), lệnh đọc luôn diễn ra từ vị trí "hiện tại".

* Hãy nhớ rằng các lệnh gọi hệ thống được định nghĩa trên cơ sở mỗi kiến trúc. Do đó, trong khi i386 có lệnh gọi hệ thống creat() , Alpha thì không. Tất nhiên, bạn có thể sử dụng creat() trên bất kỳ kiến trúc nào, nhưng nó có thể là một hàm thư viện thay vì có lệnh gọi hệ thống riêng.

www.it-ebooks.info

Machine Translated by Google

Cách sử dụng cơ bản rất đơn giản. Ví dụ này đọc từ mô tả tệp fd vào word. Số byte được đọc bằng kích thước của kiểu unsigned long , là bốn byte trên hệ thống Linux 32 bit và tám byte trên hệ thống 64 bit. Khi trả về, nr chứa số byte đã đọc hoặc -1 khi có lỗi:

```
từ dài không dấu; ssize_t
nr;

/* đọc một vài byte vào 'word' từ 'fd' */ nr = đọc (fd, &word,
sizeof (unsigned long)); nếu (nr == -1) /* lỗi */
```

Có hai vấn đề với cách triển khai ngây thơ này: lệnh gọi có thể trả về mà không đọc tất cả các byte len và có thể tạo ra một số lỗi nhất định mà mã này không kiểm tra và xử lý. Thật không may, mã như thế này rất phổ biến. Hãy cùng xem cách cải thiện nó.

Giá trị trả về

read() trả về giá trị dương khác không nhỏ hơn len là hợp lệ . Điều này có thể xảy ra vì một số lý do: có thể có ít hơn len byte khả dụng, lệnh gọi hệ thống có thể bị ngắt bởi tín hiệu, đường ống có thể bị hỏng (nếu fd là đường ống), v.v.

Khả năng trả về giá trị 0 là một cân nhắc khác khi sử dụng read( ).
Lệnh gọi hệ thống read() trả về 0 để chỉ ra kết thúc tệp (EOF); trong trường hợp này, tất nhiên, không có byte nào được đọc. EOF không được coi là lỗi (và do đó không đi kèm với giá trị trả về -1); nó chỉ đơn giản chỉ ra rằng vị trí tệp đã tiến xa hơn độ lệch hợp lệ cuối cùng trong tệp và do đó không có gì khác để đọc. Tuy nhiên, nếu một lệnh gọi được thực hiện cho len byte, nhưng không có byte nào khả dụng để đọc, lệnh gọi sẽ chặn (ngủ) cho đến khi các byte trở nên khả dụng (giả sử trình mô tả tệp không được mở ở chế độ không chặn; xem “Đọc không chặn”). Lưu ý rằng điều này khác với việc trả về EOF. Nghĩa là, có sự khác biệt giữa “không có dữ liệu khả dụng” và “kết thúc dữ liệu”. Trong trường hợp EOF, đã đạt đến cuối tệp. Trong trường hợp chặn, lệnh đọc đang chờ thêm dữ liệu—ví dụ, trong trường hợp đọc từ ổ cắm hoặc tệp thiết bị.

Một số lỗi có thể phục hồi được. Ví dụ, nếu lệnh gọi read() bị ngắt bởi một tín hiệu trước khi bất kỳ byte nào được đọc, nó trả về -1 (0 có thể bị nhầm lẫn với EOF) và errno được đặt thành EINTR. Trong trường hợp đó, bạn có thể gửi lại lệnh đọc.

Thật vậy, lệnh gọi read() có thể dẫn đến nhiều khả năng:

- Cuộc gọi trả về giá trị bằng len. Tất cả các byte đọc len được lưu trữ trong buf. Kết quả như mong đợi.
- Cuộc gọi trả về một giá trị nhỏ hơn len, nhưng lớn hơn không. Các byte đọc được lưu trữ trong buf. Điều này có thể xảy ra vì một tín hiệu ngắt giữa chừng quá trình đọc, một lỗi xảy ra giữa quá trình đọc, lớn hơn không, nhưng nhỏ hơn len byte'

www.it-ebooks.info

Machine Translated by Google

giá trị dữ liệu có sẵn hoặc EOF đạt đư ợc trư ớc khi len byte đư ợc đọc.
Việc phát hành lại lệnh đọc (với các giá trị buf và len đư ợc cập nhật tư ơ ng ứng) sẽ đọc các byte còn lại vào phần còn lại của bộ đệm hoặc chỉ ra nguyên nhân của sự cố.

- Cuộc gọi trả về 0. Điều này biểu thị EOF. Không có gì để đọc. • Cuộc gọi bị chặn vì hiện không có dữ liệu nào khả dụng. Điều này sẽ không xảy ra trong chế độ chặn.
- Cuộc gọi trả về -1 và errno đư ợc đặt thành EINTR. Điều này chỉ ra rằng tín hiệu đã đư ợc nhận trư ớc khi bất kỳ byte nào đư ợc đọc. Cuộc gọi có thể đư ợc phát hành lại.
- Cuộc gọi trả về -1 và errno đư ợc đặt thành EAGAIN. Điều này chỉ ra rằng lệnh đọc sẽ bị chặn vì hiện không có dữ liệu nào khả dụng và yêu cầu sẽ đư ợc phát hành lại sau. Điều này chỉ xảy ra ở chế độ không chặn. • Cuộc gọi trả về -1 và errno đư ợc đặt thành giá trị khác với EINTR hoặc EAGAIN. Điều này biểu thị một lỗi nghiêm trọng hơ n.

Đọc tất cả các byte Những khả

năng này ngụ ý rằng cách sử dụng read() đơ n giản, tầm thư ờng trư ớc đây không phù hợp nếu bạn muốn xử lý tất cả các lỗi và thực sự đọc tất cả các byte len (ít nhất là lên đến EOF). Để làm điều đó, bạn cần một vòng lặp và một số câu lệnh có điều kiện:

```
kích thư ớc_t ret;

trong khi (len != 0 && (ret = đọc (fd, buf, len)) != 0) { nếu
        (ret == -1) { nếu
                (errno == EINTR) tiếp
                        tục; lỗi
                ("đọc"); ngắt;

        }

        dày -=
        ret;
}
```

Đoạn mã này xử lý tất cả năm điều kiện. Vòng lặp đọc len byte từ vị trí tệp hiện tại của fd vào buf. Nó tiếp tục đọc cho đến khi đọc hết len byte hoặc cho đến khi đạt đến EOF. Nếu đọc nhiều hơ n không như ng ít hơ n len byte, len sẽ giảm đi lư ợng đã đọc, buf sẽ tăng lên lư ợng đã đọc và lệnh gọi đư ợc phát hành lại. Nếu lệnh gọi trả về -1 và errno bằng EINTR, lệnh gọi đư ợc phát hành lại mà không cập nhật các tham số. Nếu lệnh gọi trả về -1 và errno đư ợc đặt thành bất kỳ giá trị nào khác, perror() đư ợc gọi để in mô tả thành lỗi chuẩn và vòng lặp kết thúc.

Đọc một phần không chỉ hợp pháp mà còn phổ biến. Vô số lỗi xuất phát từ việc lập trình viên không kiểm tra và xử lý đúng các yêu cầu đọc ngắn. Đừng thêm vào danh sách!

www.it-ebooks.info

Machine Translated by Google

Đọc không chặn Đôi khi, các

lập trình viên không muốn lệnh gọi read() chặn khi không có dữ liệu khả dụng. Thay vào đó, họ thích lệnh gọi trả về ngay lập tức, cho biết không có dữ liệu khả dụng. Điều này được gọi là I/O không chặn; nó cho phép các ứng dụng thực hiện I/O, có khả năng trên nhiều tệp, mà không bao giờ chặn và do đó thiếu dữ liệu khả dụng trong tệp khác.

Do đó, một giá trị errno bổ sung đáng để kiểm tra: EAGAIN. Như đã thảo luận trước đó, nếu mô tả tệp đã cho được mở ở chế độ không chặn (nếu O_NONBLOCK được cung cấp cho open( ); xem "Cờ cho open()") và không có dữ liệu nào để đọc, lệnh gọi read() sẽ trả về -1 và đặt errno thành EAGAIN thay vì chặn. Khi thực hiện các lần đọc không chặn, bạn phải kiểm tra EAGAIN hoặc có nguy cơ nhầm lẫn một lỗi nghiêm trọng với việc chỉ thiếu dữ liệu. Ví dụ: bạn có thể sử dụng mã như sau:

```
char buf[BUFSIZ];
ssize_t số;

bắt đầu:
nr = đọc (fd, buf, BUFSIZ); nếu (nr == -1)
{ nếu (errno == EINTR)
        chuyển đến bắt đầu; /* oh
                shush */ nếu (errno == EAGAIN)

                /* gửi lại sau */
        khác
                /* lỗi */
}
```

Xử lý EAGAIN trong ví dụ này bằng lệnh goto start thực ra không có mấy ý nghĩa—bạn cũng có thể không sử dụng non-blocking I/O. Việc sử dụng lệnh này không tiết kiệm được thời gian mà còn gây ra nhiều chi phí hơn do phải lặp đi lặp lại.

Các giá trị lỗi khác

Các mã lỗi khác liên quan đến lỗi lập trình hoặc (đối với EIO) các vấn đề cấp thấp.
Các giá trị errno có thể có sau lỗi khi đọc () bao gồm:

EBADF

Mô tả tệp đã cho không hợp lệ hoặc không mở để đọc.

MẶC ĐỊNH

Con trỏ được buf cung cấp không nằm trong không gian địa chỉ của tiến trình gọi.

EINVAL

Bộ mô tả tệp được ánh xạ tới một đối tượng không cho phép đọc.

EIO

Đã xảy ra lỗi I/O cấp thấp.

www.it-ebooks.info

Machine Translated by Google

Giới hạn kích thư ớc khi đọc ()

Các kiểu size_t và ssize_t đư ợc POSIX yêu cầu. Kiểu size_t đư ợc sử dụng để lư u trữ các giá trị dùng để đo kích thư ớc theo byte. Kiểu ssize_t là phiên bản có dấu của size_t (các giá trị âm đư ợc sử dụng để biểu thị lỗi). Trên các hệ thống 32 bit, các kiểu C hỗ trợ thư ờng là unsigned int và int . Vì hai kiểu này thư ờng đư ợc sử dụng cùng nhau, nên phạm vi có thể nhỏ hơ n của ssize_t đặt ra giới hạn cho phạm vi của size_t.

Giá trị tối đa của size_t là SIZE_MAX; giá trị tối đa của ssize_t là SSIZE_MAX. Nếu len lớn hơ n SSIZE_MAX, kết quả của lệnh gọi read() không đư ợc xác định. Trên hầu hết các hệ thống Linux, SSIZE_MAX là LONG_MAX, tức là 0x7fffffff trên máy 32 bit. Con số này tư ơ ng đối lớn đối với một lần đọc duy nhất, như ng dù sao cũng là điều cần lư u ý. Nếu bạn sử dụng vòng lặp đọc trư ớc đó như một siêu đọc chung, bạn có thể muốn làm điều gì đó như thế này:

```
nếu (len > SSIZE_MAX)
        len = SSIZE_MAX;
```

Một lệnh gọi tới read() với len bằng 0 sẽ ngay lập tức trả về giá trị trả về là 0.

Viết với write()

Lệnh gọi hệ thống cơ bản và phổ biến nhất đư ợc sử dụng để ghi là write( ). write() là lệnh tư ơ ng ứng của read() và cũng đư ợc định nghĩa trong POSIX.1:

```
#include <unistd.h>

ssize_t ghi (int fd, const void *buf, size_t đếm);
```

Một lệnh gọi write( ) ghi tới số byte bắt đầu từ buf đến vị trí tệp hiện tại của tệp đư ợc tham chiếu bởi mô tả tệp fd. Các tệp đư ợc hỗ trợ bởi các đối tư ợng không hỗ trợ tìm kiếm (ví dụ: thiết bị ký tự) luôn ghi bắt đầu từ “đầu”.

Khi thành công, số byte đã ghi đư ợc trả về và vị trí tệp đư ợc cập nhật theo loại. Khi lỗi, -1 đư ợc trả về và errno đư ợc đặt phù hợp. Một lệnh gọi write() có thể trả về 0, như ng giá trị trả về này không có ý nghĩa đặc biệt nào; nó chỉ ngụ ý rằng không có byte nào đư ợc ghi.

Tư ơ ng tự như read( ), cách sử dụng cơ bản nhất rất đơ n giản:

```
const char *buf = "Tàu của tôi chắc chắn!";
ssize_t nr;

/* ghi chuỗi trong 'buf' vào 'fd' */ nr = write (fd,
buf, strlen (buf)); if (nr == -1) /* lỗi */
```


www.it-ebooks.info

Machine Translated by Google

Như ng một lần nữa, giống như read( ), cách sử dụng này không hoàn toàn đúng. Ngư ời gọi cũng cần kiểm tra khả năng xảy ra lệnh ghi một phần:

```
từ dài không dấu = 1720;
size_t đếm;
ssize_t số;

count = sizeof (từ); nr
= write (fd, &word, count); nếu
(nr == -1) /*
         lỗi, kiểm tra errno */ else
nếu (nr != count)
         /* có thể xảy ra lỗi, như ng 'errno' không đư ợc đặt */
```

Viết một phần

Một lệnh gọi hệ thống write() ít có khả năng trả về một lệnh ghi một phần hơ n là một lệnh gọi hệ thống read() trả về một lệnh đọc một phần. Ngoài ra, không có điều kiện EOF cho lệnh gọi hệ thống write() . Đối với các tệp thông thư ờng, write() đư ợc đảm bảo thực hiện toàn bộ lệnh ghi đư ợc yêu cầu, trừ khi có lỗi xảy ra.

Do đó, đối với các tệp thông thư ờng, bạn không cần thực hiện ghi trong vòng lặp. Tuy nhiên, đối với các loại tệp khác—ví dụ, socket—có thể cần một vòng lặp để đảm bảo rằng bạn thực sự ghi ra tất cả các byte đư ợc yêu cầu. Một lợi ích khác của việc sử dụng vòng lặp là lệnh gọi write() thứ hai có thể trả về lỗi tiết lộ nguyên nhân khiến lệnh gọi đầu tiên chỉ thực hiện ghi một phần (mặc dù, một lần nữa, tình huống này không phổ biến lắm).
Sau đây là một ví dụ:

```
ssize_t ret, số;

trong khi (len != 0 && (ret = ghi (fd, buf, len)) != 0) {
         nếu (ret == -1)
                  { nếu (errno ==
                           EINTR)
                  tiếp tục; lỗi
                  ("ghi"); ngắt;
         }

         dày -= ret;

}
```

Chế độ Thêm Khi fd

đư ợc mở ở chế độ thêm (thông qua O_APPEND), các thao tác ghi không xảy ra ở vị trí tệp hiện tại của mô tả tệp. Thay vào đó, chúng xảy ra ở phần cuối hiện tại của tệp.

Ví dụ, giả sử hai tiến trình đang ghi vào cùng một tệp. Nếu không có chế độ thêm vào, nếu tiến trình đầu tiên ghi vào cuối tệp, sau đó tiến trình thứ hai cũng làm như vậy, vị trí tệp của tiến trình đầu tiên sẽ không còn trỏ đến cuối tệp nữa.

www.it-ebooks.info

Machine Translated by Google

tệp; nó sẽ trỏ đến phần cuối của tệp, trừ dữ liệu mà quy trình thứ hai vừa ghi. Điều này có nghĩa là nhiều quy trình không bao giờ có thể thêm vào cùng một tệp mà không có sự đồng bộ hóa rõ ràng vì chúng sẽ gặp phải tình trạng chạy đua.

Chế độ Thêm tránh được vấn đề này. Nó đảm bảo rằng vị trí tệp luôn được đặt ở cuối tệp, do đó tất cả các lần ghi luôn được thêm vào, ngay cả khi có nhiều người ghi.
Bạn có thể nghĩ về nó như một bản cập nhật nguyên tử cho vị trí tệp trước mỗi yêu cầu ghi. Vị trí tệp sau đó được cập nhật để trỏ đến cuối dữ liệu mới được ghi. Điều này sẽ không quan trọng đối với lệnh gọi write( ) tiếp theo, vì nó tự động cập nhật vị trí tệp, nhưng có thể quan trọng nếu bạn gọi read() tiếp theo vì một số lý do kỳ lạ.

Chế độ Thêm rất có ý nghĩa đối với một số tác vụ nhất định, chẳng hạn như cập nhật tệp nhật ký, nhưng lại không có ý nghĩa đối với nhiều tác vụ khác.

Ghi không chặn Khi fd được

mở ở chế độ không chặn (thông qua O_NONBLOCK) và lệnh ghi được phát hành thường sẽ bị chặn, lệnh gọi hệ thống write() trả về -1 và errno được đặt thành EAGAIN.
Yêu cầu này sẽ được gửi lại sau. Thông thường, điều này không xảy ra với các tệp thông thường.

Mã lỗi khác

Các giá trị errno đáng chú ý khác bao gồm:

EBADF

Mô tả tập tin đã cho không hợp lệ hoặc không mở để ghi.

MẶC ĐỊNH

Con trỏ do buf cung cấp sẽ trỏ ra ngoài không gian địa chỉ của tiến trình.

EFBIG

Việc ghi sẽ làm cho tệp lớn hơn giới hạn tệp tối đa của mỗi quy trình hoặc giới hạn triển khai nội bộ.

EINVAL

Mô tả tệp đã cho được ánh xạ tới một đối tượng không phù hợp để ghi.

EIO

Đã xảy ra lỗi I/O cấp thấp.

ENOSPC

Hệ thống tập tin hỗ trợ mô tả tập tin đã cho không có đủ dung lượng.

EPIPE

Mô tả tệp đã cho được liên kết với một đường ống hoặc ổ cắm có đầu đọc đóng. Quy trình cũng sẽ nhận được tín hiệu SIGPIPE . Hành động mặc định cho tín hiệu SIGPIPE là chấm dứt quy trình nhận. Do đó, các quy trình chỉ nhận được giá trị errno này nếu chúng yêu cầu rõ ràng bỏ qua, chặn hoặc xử lý tín hiệu này.

www.it-ebooks.info

Machine Translated by Google

Giới hạn kích thư ớc khi ghi ()

Nếu count lớn hơ n SSIZE_MAX, kết quả của lệnh gọi write() sẽ không xác định.

Một lệnh gọi write() với số đếm bằng 0 sẽ ngay lập tức trả về giá trị là 0.

Hành vi của write()

Khi lệnh gọi write() trả về, kernel đã sao chép dữ liệu từ bộ đệm đư ợc cung cấp vào bộ đệm kernel, như ng không có gì đảm bảo rằng dữ liệu đã đư ợc ghi ra đích đến mong muốn. Thật vậy, lệnh gọi write trả về quá nhanh so với trư ờng hợp đó. Sự chênh lệch về hiệu suất giữa bộ xử lý và ổ cứng sẽ khiến hành vi như vậy trở nên rõ ràng đến đau đớn.

Thay vào đó, khi một ứng dụng không gian ngư ời dùng phát lệnh gọi hệ thống write() , hạt nhân Linux thực hiện một vài lần kiểm tra, sau đó chỉ cần sao chép dữ liệu vào một bộ đệm. Sau đó, trong nền, hạt nhân thu thập tất cả các bộ đệm "bẩn", sắp xếp chúng một cách tối ư u và ghi chúng ra đĩa (một quá trình đư ợc gọi là writeback). Điều này cho phép các lệnh gọi ghi diễn ra nhanh như chớp, trả về gần như ngay lập tức. Nó cũng cho phép hạt nhân hoãn các lệnh ghi sang các khoảng thời gian nhàn rỗi hơ n và gộp nhiều lệnh ghi lại với nhau.

Việc ghi bị trì hoãn không thay đổi ngữ nghĩa POSIX. Ví dụ, nếu lệnh đọc đư ợc phát hành cho một phần dữ liệu vừa đư ợc ghi nằm trong bộ đệm và chư a có trên đĩa, yêu cầu sẽ đư ợc đáp ứng từ bộ đệm và không gây ra lệnh đọc từ dữ liệu "cũ" trên đĩa.
Hành vi này thực sự cải thiện hiệu suất, vì lệnh đọc đư ợc đáp ứng từ bộ nhớ đệm trong bộ nhớ mà không cần phải chuyển đến đĩa. Các yêu cầu đọc và ghi xen kẽ như mong đợi, và kết quả như mong đợi—tức là, nếu hệ thống không bị sập trư ớc khi dữ liệu đư ợc đư a vào đĩa! Mặc dù ứng dụng có thể tin rằng lệnh ghi đã diễn ra thành công, như ng trong trư ờng hợp này, dữ liệu sẽ không bao giờ đư ợc đư a vào đĩa.

Một vấn đề khác với việc ghi bị trì hoãn là không thể thực thi thứ tự ghi. Mặc dù một ứng dụng có thể sắp xếp thứ tự các yêu cầu ghi của mình theo cách mà chúng chạm vào đĩa theo thứ tự cụ thể, như ng hạt nhân sẽ sắp xếp lại các yêu cầu ghi theo cách mà nó thấy phù hợp, chủ yếu là để tăng hiệu suất. Thông thư ờng, đây chỉ là vấn đề nếu hệ thống bị sập, vì cuối cùng tất cả các bộ đệm đều đư ợc ghi lại và mọi thứ đều ổn. Ngay cả khi đó, phần lớn các ứng dụng thực sự không quan tâm đến thứ tự ghi.

Một vấn đề cuối cùng với việc ghi bị trì hoãn liên quan đến việc báo cáo một số lỗi I/O nhất định. Bất kỳ lỗi I/O nào xảy ra trong quá trình ghi lại—ví dụ, lỗi ổ đĩa vật lý—không thể đư ợc báo cáo lại cho quy trình đã đư a ra yêu cầu ghi. Thật vậy, bộ đệm không liên quan gì đến quy trình. Nhiều quy trình có thể đã làm bẩn dữ liệu chứa trong một bộ đệm duy nhất và các quy trình có thể thoát sau khi ghi dữ liệu vào bộ đệm như ng trư ớc đó dữ liệu đư ợc ghi lại vào đĩa. Bên cạnh đó, làm thế nào bạn có thể thông báo cho một quy trình rằng một lần ghi đã thất bại sau sự kiện?

www.it-ebooks.info

Machine Translated by Google

Kernel cố gắng giảm thiểu rủi ro của việc ghi bị trì hoãn. Để đảm bảo dữ liệu được ghi ra kịp thời, kernel thiết lập tuổi bộ đệm tối đa và ghi ra tất cả các bộ đệm bẩn trước khi chúng đạt đến giá trị đã cho. Người dùng có thể cấu hình giá trị này qua /proc/sys/vm/dirty_expire_centiseconds. Giá trị được chỉ định bằng centi giây (một phần trăm giây).

Cũng có thể buộc ghi lại bộ đệm của một tệp nhất định hoặc thậm chí làm cho tất cả các lần ghi đồng bộ. Các chủ đề này được thảo luận trong phần tiếp theo, “I/O đồng bộ”.

Ở phần sau của chương này, “Nội dung bên trong hạt nhân” sẽ trình bày sâu hơn về hệ thống ghi lại bộ đệm của hạt nhân Linux.

I/O đồng bộ

Mặc dù đồng bộ hóa I/O là một chủ đề quan trọng, nhưng không nên lo ngại về các vấn đề liên quan đến ghi trễ. Ghi đệm cung cấp cải tiến hiệu suất rất lớn và do đó, bất kỳ hệ điều hành nào thậm chí xứng đáng được coi là "hiện đại" đều triển khai ghi trễ thông qua bộ đệm. Tuy nhiên, có những lúc các ứng dụng muốn kiểm soát thời điểm dữ liệu chạm vào đĩa. Đối với những mục đích sử dụng đó, hạt nhân Linux cung cấp một số tùy chọn cho phép trao đổi hiệu suất để lấy các hoạt động được đồng bộ hóa.

fsync() và fdatasync()

Phương pháp đơn giản nhất để đảm bảo dữ liệu đã đến đĩa là thông qua lệnh gọi hệ thống fsync() , được định nghĩa bởi POSIX.1b:

```
#include <unistd.h>

int fsync (int fd);
```

Một lệnh gọi đến fsync( ) đảm bảo rằng tất cả dữ liệu bẩn liên quan đến tệp được ánh xạ bởi mô tả tệp fd được ghi lại vào đĩa. Mô tả tệp fd phải được mở để ghi. Lệnh gọi ghi lại cả dữ liệu và siêu dữ liệu, chẳng hạn như dấu thời gian tạo và các thuộc tính khác có trong inode. Nó sẽ không trả về cho đến khi ổ cứng cho biết dữ liệu và siêu dữ liệu nằm trên đĩa.

Trong trường hợp bộ nhớ đệm ghi trên ổ cứng, fsync() không thể biết được dữ liệu có thực sự nằm trên đĩa hay không. Ổ cứng có thể báo cáo rằng dữ liệu đã được ghi, nhưng dữ liệu thực tế có thể nằm trong bộ nhớ đệm ghi của ổ đĩa. May mắn thay, dữ liệu trong bộ nhớ đệm của ổ cứng sẽ được cam kết với đĩa trong thời gian ngắn.

Linux cũng cung cấp lệnh gọi hệ thống fdatasync( ):

```
#include <unistd.h>

int fdatasync (int fd);
```


www.it-ebooks.info

Machine Translated by Google

Cuộc gọi hệ thống này thực hiện cùng một việc như fsync( ), ngoại trừ việc nó chỉ xóa dữ liệu. Cuộc gọi không đảm bảo rằng siêu dữ liệu được đồng bộ hóa với đĩa và do đó có khả năng nhanh hơn. Thường thì điều này là đủ.

Cả hai hàm đều được sử dụng theo cùng một cách, rất đơn giản:

```
int ret;

ret = fsync (fd); nếu
(ret == -1) /* lỗi
          */
```

Không hàm nào đảm bảo rằng bất kỳ mục nhập thư mục nào được cập nhật có chứa tệp đều được đồng bộ hóa với đĩa. Điều này ngụ ý rằng nếu liên kết của tệp vừa được cập nhật, dữ liệu của tệp có thể đến được đĩa nhưng không đến được mục nhập thư mục được liên kết, khiến tệp không thể truy cập được. Để đảm bảo rằng bất kỳ bản cập nhật nào cho mục nhập thư mục cũng được cam kết với đĩa, fsync() phải được gọi trên một mô tả tệp được mở đối với chính thư mục đó.

Giá trị trả về và mã lỗi

Khi thành công, cả hai lệnh gọi đều trả về 0. Khi thất bại, cả hai lệnh gọi đều trả về -1 và đặt errno thành một trong ba giá trị sau:

EBADF

Mô tả tệp đã cho không phải là mô tả tệp hợp lệ để ghi.

EINVAL

Mô tả tệp đã cho được ánh xạ tới một đối tượng không hỗ trợ đồng bộ hóa.

EIO

Đã xảy ra lỗi I/O cấp thấp trong quá trình đồng bộ hóa. Đây là lỗi I/O thực sự và thường là nơi phát hiện ra các lỗi như vậy.

Hiện tại, lệnh gọi đến fsync( ) có thể không thành công vì fsync( ) không được hệ thống tệp sao lưu triển khai, ngay cả khi fdatasync( ) được triển khai. Các ứng dụng hoang tưởng có thể muốn thử fdatasync( ) nếu fsync( ) trả về EINVAL. Ví dụ:

```
nếu (fsync (fd) == -1) {
        /*
         * Chúng tôi thích fsync() hơn, nhưng hãy thử fdatasync( ) *
         nếu fsync( ) không thành công, đề phòng. */

        nếu (errno == EINVAL)
                { nếu (fdatasync (fd) ==
                        -1) lỗi ("fdatasync");
        } khác
                lỗi ("fsync");
}
```


www.it-ebooks.info

Machine Translated by Google

Vì POSIX yêu cầu fsync( ) nhưng gắn nhãn fdatasync() là tùy chọn, nên lệnh gọi hệ thống fsync() luôn phải được triển khai cho các tệp thông thường trên bất kỳ hệ thống tệp Linux phổ biến nào. Tuy nhiên, các loại tệp lạ (có thể là những loại tệp không có siêu dữ liệu để đồng bộ hóa) hoặc các hệ thống tệp lạ chỉ có thể triển khai fdatasync( ).

đồng bộ hóa( )

Ít tối ưu hơn nhưng có phạm vi rộng hơn, lệnh gọi hệ thống sync() cũ được cung cấp để đồng bộ hóa tất cả các bộ đệm vào đĩa:

```
#include <unistd.h>

void đồng bộ (void);
```

Hàm không có tham số và không có giá trị trả về. Nó luôn thành công và khi trả về, tất cả các bộ đệm—cả dữ liệu và siêu dữ liệu—đều được đảm bảo nằm trên đĩa.* Các tiêu chuẩn

không yêu cầu sync() phải đợi cho đến khi tất cả các bộ đệm được xả vào đĩa trước khi trả về; chúng chỉ yêu cầu lệnh gọi khởi tạo quy trình cam kết tất cả các bộ đệm vào đĩa. Vì lý do này, người ta thường khuyên nên đồng bộ hóa nhiều lần để đảm bảo rằng tất cả dữ liệu đều an toàn trên đĩa. Tuy nhiên, Linux sẽ đợi cho đến khi tất cả các bộ đệm được cam kết. Do đó, chỉ cần một sync() là đủ.

Công dụng thực sự duy nhất của sync( ) là trong việc triển khai tiện ích sync(8) . Các ứng dụng nên sử dụng fsync( ) và fdatasync( ) để cam kết lưu trữ dữ liệu của chỉ các mô tả tệp cần thiết vào đĩa. Lưu ý rằng sync( ) có thể mất vài phút để hoàn tất trên hệ thống bận.

Cờ O_SYNC

Cờ O_SYNC có thể được truyền tới open( ), cho biết rằng tất cả I/O trên tệp phải được đồng bộ hóa:

```
số nguyên fd;

fd = mở (tệp, O_WRONLY | O_SYNC); nếu (fd
== -1) { perror
          ("mở"); trả về -1;

}
```

Yêu cầu đọc luôn được đồng bộ hóa. Nếu không, tính hợp lệ của dữ liệu đọc trong bộ đệm được cung cấp sẽ không được biết. Tuy nhiên, như đã thảo luận trước đó, các lệnh gọi write() thường không được đồng bộ hóa. Không có mối quan hệ nào giữa lệnh gọi trả về và dữ liệu được cam kết vào đĩa. Cờ O_SYNC buộc mối quan hệ này, đảm bảo rằng các lệnh gọi write() thực hiện I/O được đồng bộ hóa.

* Vâng, vẫn áp dụng cảnh báo như trước: ổ cứng có thể nói dối và thông báo cho nhân rằng bộ đệm nằm trên đĩa trong khi thực tế chúng vẫn nằm trong bộ nhớ đệm của đĩa.

www.it-ebooks.info

Machine Translated by Google

Một cách để xem xét O_SYNC là nó buộc một fsync() ngầm định sau mỗi thao tác write() , trư ớc khi cuộc gọi trả về. Đây thực sự là ngữ nghĩa đư ợc cung cấp, mặc dù hạt nhân Linux triển khai O_SYNC hiệu quả hơ n một chút.

O_SYNC dẫn đến thời gian ngư ời dùng và nhân tệ hơ n một chút (thời gian dành cho không gian ngư ời dùng và nhân, tư ơ ng ứng) cho các hoạt động ghi. Hơ n nữa, tùy thuộc vào kích thư ớc của tệp đang đư ợc ghi, O_SYNC có thể khiến tổng thời gian trôi qua tăng lên một hoặc hai cấp độ vì toàn bộ thời gian chờ I/O (thời gian dành cho việc chờ I/O hoàn tất) đều do quy trình thực hiện. Chi phí tăng lên rất lớn, do đó chỉ nên sử dụng I/O đồng bộ sau khi đã sử dụng hết mọi phư ơ ng án thay thế có thể.

Thông thư ờng, các ứng dụng cần đảm bảo rằng các hoạt động ghi đã đư ợc ghi vào đĩa sẽ sử dụng fsync() hoặc fdatasync( ). Các hoạt động này có xu hư ớng tốn ít chi phí hơ n O_SYNC vì chúng có thể đư ợc gọi ít thư ờng xuyên hơ n (tức là chỉ sau khi một số hoạt động quan trọng đã hoàn tất).

O_DSYNC và O_RSYNC POSIX định nghĩa

hai cờ open() liên quan đến synchronized-I/O khác : O_DSYNC và O_RSYNC.
Trên Linux, các cờ này đư ợc định nghĩa là đồng nghĩa với O_SYNC; chúng cung cấp cùng một hành vi.

Cờ O_DSYNC chỉ định rằng chỉ dữ liệu bình thư ờng đư ợc đồng bộ hóa sau mỗi thao tác ghi, không phải siêu dữ liệu. Hãy nghĩ về nó như gây ra fdatasync() ngầm định sau mỗi yêu cầu ghi. Vì O_SYNC cung cấp các đảm bảo mạnh hơ n, nên không có mất mát chức năng nào khi không hỗ trợ O_DSYNC một cách rõ ràng; chỉ có mất mát hiệu suất tiềm ẩn từ các yêu cầu mạnh hơ n do O_SYNC cung cấp.

Cờ O_RSYNC chỉ định việc đồng bộ hóa các yêu cầu đọc cũng như các yêu cầu ghi. Nó phải đư ợc sử dụng với một trong O_SYNC hoặc O_DSYNC. Như đã đề cập trư ớc đó, các lần đọc đã đư ợc đồng bộ hóa— sau cùng, chúng không trả về cho đến khi chúng có thứ gì đó để cung cấp cho ngư ời dùng. Cờ O_RSYNC quy định rằng bất kỳ tác dụng phụ nào của hoạt động đọc cũng phải đư ợc đồng bộ hóa. Điều này có nghĩa là các bản cập nhật siêu dữ liệu phát sinh từ một lần đọc phải đư ợc ghi vào đĩa trư ớc khi cuộc gọi trả về. Về mặt thực tế, yêu cầu này rất có thể chỉ có nghĩa là thời gian truy cập tệp phải đư ợc cập nhật trong bản sao trên đĩa của inode trư ớc khi cuộc gọi đến read() trả về. Linux định nghĩa O_RSYNC giống với O_SYNC, mặc dù điều này không có nhiều ý nghĩa (hai cái không liên quan nhiều như O_SYNC và O_DSYNC). Hiện tại, không có cách nào trong Linux để có đư ợc hành vi của O_RSYNC; cách gần nhất mà một nhà phát triển có thể làm là gọi fdatasync() sau mỗi lần gọi read() . Tuy nhiên, hành vi này hiếm khi cần thiết.

I/O trực tiếp

Nhân Linux, giống như bất kỳ nhân hệ điều hành hiện đại nào, triển khai một lớp phức tạp của bộ nhớ đệm, bộ đệm và quản lý I/O giữa các thiết bị và ứng dụng (xem “Nội dung bên trong nhân” ở cuối chư ơ ng này). Một ứng dụng hiệu suất cao có thể muốn bỏ qua lớp phức tạp này và thực hiện quản lý I/O của riêng nó.

www.it-ebooks.info

Machine Translated by Google

Tuy nhiên, việc tự xây dựng hệ thống I/O thư ờng không đáng công sức và trên thực tế, các công cụ có sẵn ở cấp hệ điều hành có thể đạt đư ợc hiệu suất tốt hơ n nhiều so với các công cụ có sẵn ở cấp ứng dụng. Tuy nhiên, các hệ thống cơ sở dữ liệu thư ờng thích thực hiện lư u trữ đệm riêng của chúng và muốn giảm thiểu sự hiện diện của hệ điều hành càng nhiều càng tốt.

Cung cấp cờ O_DIRECT cho open() sẽ hư ớng dẫn kernel giảm thiểu sự hiện diện của quản lý I/O. Khi cờ này đư ợc cung cấp, I/O sẽ khởi tạo trực tiếp từ bộ đệm không gian ngư ời dùng đến thiết bị, bỏ qua bộ đệm trang. Tất cả I/O sẽ đồng bộ; các hoạt động sẽ không trả về cho đến khi hoàn tất.

Khi thực hiện I/O trực tiếp, độ dài yêu cầu, căn chỉnh bộ đệm và độ lệch tệp đều phải là bội số nguyên của kích thư ớc sector của thiết bị cơ bản—nói chung, đây là 512 byte. Trư ớc hạt nhân Linux 2.6, yêu cầu này nghiêm ngặt hơ n: trong 2.4, mọi thứ phải đư ợc căn chỉnh theo kích thư ớc khối logic của hệ thống tệp (thư ờng là 4 KB). Để duy trì khả năng tư ơ ng thích, các ứng dụng phải căn chỉnh theo kích thư ớc khối logic lớn hơ n (và có khả năng kém thuận tiện hơ n).

Đóng tập tin

Sau khi chư ơ ng trình hoàn tất công việc với một mô tả tệp, nó có thể hủy ánh xạ mô tả tệp khỏi tệp đư ợc liên kết thông qua lệnh gọi hệ thống close() :

```
#include <unistd.h>

int đóng (int fd);
```

Một lệnh gọi close() sẽ hủy ánh xạ mô tả tệp mở fd và tách quy trình khỏi tệp. Mô tả tệp đã cho sau đó không còn hợp lệ nữa và hạt nhân có thể sử dụng lại nó làm giá trị trả về cho lệnh gọi open() hoặc creat() tiếp theo . Một lệnh gọi close() trả về 0 khi thành công. Nếu có lỗi, nó trả về -1 và đặt errno một cách thích hợp. Cách sử dụng rất đơ n giản:

```
nếu (đóng (fd) == -1)
        perror ("đóng");
```

Lư u ý rằng việc đóng một tệp không liên quan đến thời điểm tệp đư ợc đư a vào đĩa. Nếu một ứng dụng muốn đảm bảo rằng tệp đư ợc cam kết vào đĩa trư ớc khi đóng, ứng dụng đó cần sử dụng một trong các tùy chọn đồng bộ hóa đã thảo luận trư ớc đó trong “I/O đồng bộ”.

Tuy nhiên, việc đóng một tệp cũng có một số tác dụng phụ. Khi mô tả tệp mở cuối cùng tham chiếu đến một tệp bị đóng, cấu trúc dữ liệu biểu diễn tệp bên trong hạt nhân đư ợc giải phóng. Khi cấu trúc dữ liệu này đư ợc giải phóng, nó sẽ gỡ ghim bản sao trong bộ nhớ của inode đư ợc liên kết với tệp. Nếu không có gì khác ghim inode, nó cũng có thể đư ợc giải phóng khỏi bộ nhớ (nó có thể vẫn tồn tại vì hạt nhân lư u trữ đệm inode vì lý do hiệu suất, như ng không nhất thiết phải như vậy). Nếu một tệp đã đư ợc hủy liên kết khỏi đĩa, như ng vẫn đư ợc giữ mở trư ớc khi hủy liên kết, tệp đó sẽ không bị xóa về mặt vật lý cho đến khi tệp đó đư ợc đóng và inode của tệp đó bị xóa khỏi bộ nhớ. Do đó, việc gọi close() cũng có thể dẫn đến việc tệp không đư ợc liên kết cuối cùng bị xóa về mặt vật lý khỏi đĩa.

www.it-ebooks.info

Machine Translated by Google

Giá trị lỗi

Một lỗi thư ờng gặp là không kiểm tra giá trị trả về của close( ). Điều này có thể dẫn đến việc bỏ sót điều kiện lỗi quan trọng vì lỗi liên quan đến các hoạt động bị hoãn có thể không biểu hiện cho đến sau này và close() có thể báo cáo chúng.

Có một số giá trị errno có thể xảy ra khi lỗi. Ngoài EBADF (mô tả tệp đã cho không hợp lệ), giá trị lỗi quan trọng nhất là EIO, biểu thị lỗi I/O cấp thấp có thể không liên quan đến việc đóng thực tế. Bất kể lỗi nào đư ợc báo cáo, mô tả tệp, nếu hợp lệ, luôn đư ợc đóng và các cấu trúc dữ liệu liên quan đư ợc giải phóng.

Mặc dù POSIX cho phép, close() sẽ không bao giờ trả về EINTR. Các nhà phát triển hạt nhân Linux hiểu rõ hơ n—việc triển khai như vậy là không thông minh.

Tìm kiếm với lseek()

Thông thư ờng, I/O xảy ra tuyến tính thông qua một tệp và các bản cập nhật ngầm định vào vị trí tệp do đọc và ghi gây ra là tất cả các tìm kiếm cần thiết. Tuy nhiên, một số ứng dụng cần phải nhảy xung quanh trong tệp. Lệnh gọi hệ thống lseek() đư ợc cung cấp để đặt vị trí tệp của một mô tả tệp thành một giá trị nhất định. Ngoài việc cập nhật vị trí tệp, nó không thực hiện bất kỳ hành động nào khác và không khởi tạo bất kỳ I/O nào:

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek (int fd, off_t pos, int origin);
```

Hành vi của lseek() phụ thuộc vào đối số gốc , có thể là một trong những đối số sau:

SEEK_CUR
: Vị trí tệp hiện tại của fd đư ợc đặt thành giá trị hiện tại cộng với pos, có thể là số âm, số không hoặc số dư ơ ng. Pos bằng số không trả về giá trị vị trí tệp hiện tại.

SEEK_END
: Vị trí tệp hiện tại của fd đư ợc đặt thành độ dài hiện tại của tệp cộng với pos, có thể là số âm, số không hoặc số dư ơ ng. Pos bằng số không đặt độ lệch đến cuối tệp.

SEEK_SET
: Vị trí tệp hiện tại của fd đư ợc đặt thành pos. Pos bằng 0 sẽ đặt offset vào đầu tệp.

Cuộc gọi trả về vị trí tệp mới khi thành công. Khi có lỗi, nó trả về -1 và errno đư ợc đặt thành phù hợp.

www.it-ebooks.info

Machine Translated by Google

Ví dụ, để đặt vị trí tệp fd thành 1825:

```
tắt_t ret;

ret = lseek (fd, (off_t) 1825, SEEK_SET); nếu (ret
== (off_t) -1) /* lỗi */
```

Ngoài ra, để đặt vị trí tệp fd vào cuối tệp:

```
tắt_t ret;

ret = lseek(fd, 0, SEEK_END); nếu (ret
== (off_t) -1) /* lỗi */
```

Vì lseek() trả về vị trí tệp đã cập nhật nên nó có thể đư ợc sử dụng để tìm vị trí tệp hiện tại thông qua SEEK_CUR tới số không:

```
int pos;

pos = lseek (fd, 0, SEEK_CUR); nếu (pos
== (off_t) -1) /* lỗi */

khác
          /* 'pos' là vị trí hiện tại của fd */
```

Cho đến nay, cách sử dụng phổ biến nhất của lseek() là tìm kiếm đến đầu, tìm kiếm đến cuối hoặc xác định vị trí tệp hiện tại của một mô tả tệp.

Tìm kiếm quá cuối tệp Có thể hư ớng dẫn

lseek() để đư a con trỏ tệp tiến xa hơ n đến cuối tệp. Ví dụ, đoạn mã này tìm kiếm đến 1.688 byte sau cuối tệp đư ợc ánh xạ bởi fd:

```
int ret;

ret = lseek (fd, (off_t) 1688, SEEK_END); nếu (ret
== (off_t) -1) /* lỗi */
```

Riêng việc tìm kiếm sau phần cuối của tệp không có tác dụng gì—yêu cầu đọc đến vị trí tệp mới tạo sẽ trả về EOF. Tuy nhiên, nếu yêu cầu ghi sau đó đư ợc thực hiện đến vị trí này, khoảng cách mới sẽ đư ợc tạo giữa độ dài cũ của tệp và độ dài mới, và nó sẽ đư ợc đệm bằng số không.

Phần đệm bằng không này đư ợc gọi là lỗ hổng. Trên các hệ thống tệp kiểu Unix, lỗ hổng không chiếm bất kỳ không gian đĩa vật lý nào. Điều này ngụ ý rằng tổng kích thư ớc của tất cả các tệp trên một hệ thống tệp có thể cộng lại nhiều hơ n kích thư ớc vật lý của đĩa. Các tệp có lỗ hổng đư ợc gọi là tệp thư a thớt. Các tệp thư a thớt có thể tiết kiệm đáng kể không gian và tăng cư ờng hiệu suất vì việc thao tác các lỗ hổng không khởi tạo bất kỳ I/O vật lý nào.

Yêu cầu đọc một phần tệp trong lỗ sẽ trả về số lư ợng số không nhị phân thích hợp.

www.it-ebooks.info

Machine Translated by Google

Giá trị lỗi

Khi xảy ra lỗi, lseek() trả về -1 và errno được đặt thành một trong bốn giá trị sau:

EBADF
: Mô tả tệp được cung cấp không đề cập đến mô tả tệp đang mở.

EINVAL
: Giá trị được đưa ra cho origin không phải là một trong các giá trị SEEK_SET, SEEK_CUR hoặc SEEK_END, nếu không vị trí tệp kết quả sẽ là số âm. Thật không may khi EINVAL biểu thị cả hai lỗi này. Lỗi trước gần như chắc chắn là lỗi lập trình thời gian biên dịch, trong khi lỗi sau có thể biểu thị logic thời gian chạy nguy hiểm hơn
lỗi.

TRÀN
: Không thể biểu diễn độ lệch tệp kết quả trong off_t. Điều này chỉ có thể xảy ra trên kiến trúc 32 bit. Hiện tại, vị trí tệp được cập nhật; lỗi này chỉ ra rằng không thể trả về vị trí đó.

TRÍCH DẪN
: Bộ mô tả tệp được cung cấp có liên quan đến một đối tượng không thể tìm kiếm, chẳng hạn như đường ống, FIFO hoặc ổ cắm.

Hạn chế

Các vị trí tệp tối đa bị giới hạn bởi kích thước của kiểu off_t . Hầu hết các kiến trúc máy định nghĩa đây là kiểu C long , trên Linux luôn là word size (thường là kích thước của các thanh ghi mục đích chung của máy). Tuy nhiên, về mặt nội bộ, hạt nhân lưu trữ các offset trong kiểu C long long . Điều này không gây ra vấn đề gì trên các máy 64 bit, nhưng điều đó có nghĩa là các máy 32 bit có thể tạo ra lỗi EOVERFLOW khi thực hiện tìm kiếm tương đối.

Đọc và ghi theo vị trí

Thay vì sử dụng lseek( ), Linux cung cấp hai biến thể của lệnh gọi hệ thống read() và write() , mỗi biến thể lấy vị trí tệp để đọc hoặc ghi làm tham số.
Sau khi hoàn tất, họ không cập nhật vị trí tệp.

Biểu mẫu đọc được gọi là pread( ):

```
#define _XOPEN_SOURCE 500

#include <unistd.h>

ssize_t pread (int fd, void *buf, size_t count, off_t pos);
```


www.it-ebooks.info

Machine Translated by Google

Cuộc gọi này đọc tới số byte vào buf từ mô tả tệp fd tại vị trí tệp pos.

Biểu mẫu ghi đư ợc gọi là pwrite( ):

```
#define _XOPEN_SOURCE 500

#include <unistd.h>

ssize_t pwrite (int fd, const void *buf, size_t count, off_t pos);
```

Cuộc gọi này ghi tới số byte từ buf vào bộ mô tả tệp fd tại vị trí tệp pos.

Những cuộc gọi này có hành vi gần giống với những ngư ời anh em không phải p của chúng, ngoại trừ việc chúng hoàn toàn bỏ qua vị trí tệp hiện tại; thay vì sử dụng vị trí hiện tại, chúng sử dụng giá trị do pos cung cấp. Ngoài ra, khi thực hiện xong, chúng không cập nhật vị trí tệp.
Nói cách khác, bất kỳ lệnh gọi read() và write() nào đư ợc trộn lẫn đều có khả năng làm hỏng công việc đư ợc thực hiện bởi các lệnh gọi vị trí.

Cả hai lệnh gọi vị trí đều chỉ có thể đư ợc sử dụng trên các mô tả tệp có thể tìm kiếm. Chúng cung cấp ngữ nghĩa tư ơ ng tự như lệnh gọi trư ớc lệnh read() hoặc write() bằng lệnh gọi đến lseek( ), với ba điểm khác biệt. Đầu tiên, các lệnh gọi này dễ sử dụng hơ n, đặc biệt là khi thực hiện thao tác phức tạp như di chuyển ngư ợc lại hoặc ngẫu nhiên qua tệp. Thứ hai, chúng không cập nhật con trỏ tệp khi hoàn tất. Cuối cùng, và quan trọng nhất, chúng tránh mọi cuộc đua tiềm ẩn có thể xảy ra khi sử dụng lseek( ). Vì các luồng chia sẻ các mô tả tệp, nên một luồng khác trong cùng một chư ơ ng trình có thể cập nhật vị trí tệp sau lệnh gọi lseek( ) của luồng đầu tiên, như ng trư ớc khi thực hiện thao tác đọc hoặc ghi của nó. Có thể tránh các tình trạng đua như vậy bằng cách sử dụng các lệnh gọi hệ thống pread() và pwrite() .

Giá trị lỗi

Khi thành công, cả hai lệnh gọi đều trả về số byte đã đọc hoặc đã ghi. Giá trị trả về 0 từ pread() chỉ ra EOF; từ pwrite( ), giá trị trả về 0 chỉ ra rằng lệnh gọi không ghi bất cứ thứ gì. Khi có lỗi, cả hai lệnh gọi đều trả về -1 và đặt errno một cách thích hợp.

Đối với pread( ), bất kỳ giá trị errno read() hoặc lseek() hợp lệ nào cũng có thể thực hiện đư ợc. Đối với pwrite( ), bất kỳ giá trị write() hoặc lseek() hợp lệ nào cũng có thể thực hiện đư ợc.

Cắt bớt tập tin

Linux cung cấp hai lệnh gọi hệ thống để cắt bớt độ dài của tệp, cả hai đều đư ợc xác định và yêu cầu (ở các mức độ khác nhau) bởi nhiều tiêu chuẩn POSIX. Chúng là:

```
#include <unistd.h>
#include <sys/types.h>

int ftruncate(int fd, off_t len);
```


www.it-ebooks.info

Machine Translated by Google

Và:

```
#include <unistd.h>
#include <sys/types.h>

int cắt ngắn (const char *path, off_t len);
```

Cả hai lệnh gọi hệ thống đều cắt bớt tệp đã cho theo độ dài được chỉ định bởi len. Lệnh gọi hệ thống ftruncate() hoạt động trên mô tả tệp được chỉ định bởi fd, phải được mở để ghi. Lệnh gọi hệ thống truncate() hoạt động trên tên tệp được chỉ định bởi path, phải có thể ghi. Cả hai đều trả về 0 nếu thành công. Nếu có lỗi, chúng trả về -1 và đặt errno theo đúng giá trị.

Cách sử dụng phổ biến nhất của các lệnh gọi hệ thống này là cắt bớt một tệp thành kích thước nhỏ hơn độ dài hiện tại của tệp đó. Khi trả về thành công, độ dài của tệp là len. Dữ liệu trước đó tồn tại giữa len và độ dài cũ sẽ bị loại bỏ và không còn có thể truy cập được thông qua yêu cầu đọc.

Các hàm này cũng có thể được sử dụng để "cắt bớt" một tệp thành một tệp có kích thước lớn hơn, tương tự như sự kết hợp tìm kiếm và ghi được mô tả trước đó trong "Tìm kiếm sau phần cuối của tệp". Các byte mở rộng được điền bằng số không.

Không có thao tác nào cập nhật vị trí tệp hiện tại.

Ví dụ, hãy xem xét tệp pirate.txt có độ dài 74 byte với nội dung sau:

```
Edward Teach là một tên cướp biển khét tiếng người Anh.
Ông có biệt danh là Râu Đen.
```

Từ cùng thư mục đó, chạy chương trình sau:

```
#include <unistd.h>
#include <stdio.h>

int chính ( )
{
          int ret;

          ret = cắt ngắn ("./pirate.txt", 45); nếu (ret
          == -1) { perror
                    ("cắt ngắn"); trả về -1;

          }

          trả về 0;
}
```

kết quả là một tập tin có độ dài 45 byte với nội dung:

```
Edward Teach là một tên cướp biển khét tiếng người Anh.
```


www.it-ebooks.info

Machine Translated by Google

I/O ghép kênh

Các ứng dụng thường cần chặn nhiều hơn một mô tả tệp, xử lý I/O giữa đầu vào bàn phím (stdin), giao tiếp giữa các tiến trình và một số tệp.
Các ứng dụng giao diện người dùng đồ họa (GUI) hiện đại điều khiển sự kiện có thể xử lý hàng trăm sự kiện đang chờ xử lý thông qua các vòng lặp chính của chúng.*

Nếu không có sự trợ giúp của các luồng—về cơ bản là phục vụ từng mô tả tệp riêng biệt—một quy trình đơn lẻ không thể chặn hợp lý trên nhiều hơn một mô tả tệp cùng một lúc. Làm việc với nhiều mô tả tệp là tốt, miễn là chúng luôn sẵn sàng để đọc hoặc ghi vào. Nhưng ngay khi gặp phải một mô tả tệp chưa sẵn sàng—ví dụ, nếu lệnh gọi hệ thống read() được đưa ra và vẫn chưa có dữ liệu nào—quy trình sẽ chặn, không còn có thể phục vụ các mô tả tệp khác. Nó có thể chặn chỉ trong vài giây, khiến ứng dụng kém hiệu quả và gây khó chịu cho người dùng.

Tuy nhiên, nếu không có dữ liệu nào khả dụng trên trình mô tả tệp, tệp có thể bị chặn mãi mãi. Bởi vì I/O của các mô tả tệp thường có mối liên hệ với nhau—hãy nghĩ đến các đường ống—nên có khả năng một mô tả tệp không sẵn sàng cho đến khi mô tả tệp khác được phục vụ. Đặc biệt với các ứng dụng mạng, có thể có nhiều ổ cắm mở cùng lúc, đây có khả năng là một vấn đề khá lớn.

Hãy tưởng tượng việc chặn một mô tả tệp liên quan đến giao tiếp giữa các tiến trình trong khi stdin có dữ liệu đang chờ xử lý. Ứng dụng sẽ không biết rằng đầu vào bàn phím đang chờ xử lý cho đến khi mô tả tệp IPC bị chặn cuối cùng trả về dữ liệu—nhưng nếu thao tác bị chặn không bao giờ trả về thì sao?

Ở phần trước của chương này, chúng ta đã xem xét I/O không chặn như một giải pháp cho vấn đề này. Với I/O không chặn, các ứng dụng có thể đưa ra các yêu cầu I/O trả về một điều kiện lỗi đặc biệt thay vì chặn. Tuy nhiên, giải pháp này không hiệu quả vì hai lý do.
Đầu tiên, quy trình cần liên tục phát hành các hoạt động I/O theo một thứ tự tùy ý, chờ một trong các mô tả tệp mở của nó sẵn sàng cho I/O. Đây là thiết kế chương trình kém. Thứ hai, sẽ hiệu quả hơn nhiều nếu chương trình có thể ngủ, giải phóng bộ xử lý cho các tác vụ khác, chỉ được đánh thức khi một hoặc nhiều mô tả tệp đã sẵn sàng để thực hiện I/O.

Nhập I/O ghép kênh.

I/O ghép kênh cho phép ứng dụng đồng thời chặn nhiều mô tả tệp và nhận thông báo khi bất kỳ mô tả nào trong số chúng sẵn sàng để đọc hoặc ghi mà không bị chặn. Do đó, I/O ghép kênh trở thành điểm xoay cho ứng dụng, được thiết kế tương tự như sau:

1. I/O đa kênh: Cho tôi biết khi nào bất kỳ mô tả tệp nào trong số này sẵn sàng cho I/O.
2. Ngủ cho đến khi một hoặc nhiều mô tả tệp đã sẵn sàng.

* Mainloops sẽ quen thuộc với bất kỳ ai đã viết ứng dụng GUI—ví dụ, các ứng dụng GNOME sử dụng mainloop do GLib cung cấp, thư viện cơ sở của họ. Mainloop cho phép theo dõi và phản hồi nhiều sự kiện từ một điểm chặn duy nhất.

www.it-ebooks.info

Machine Translated by Google

3. Thức dậy: Cái gì đã sẵn sàng?
4. Xử lý tất cả các mô tả tệp sẵn sàng cho I/O mà không chặn.
5. Quay lại bước 1 và bắt đầu lại.

Linux cung cấp ba giải pháp I/O đa kênh: giao diện select, poll và epoll . Chúng tôi sẽ đề cập đến hai giải pháp đầu tiên ở đây, còn giải pháp cuối cùng, là giải pháp nâng cao dành riêng cho Linux, sẽ được đề cập trong Chương 4.

lựa chọn( )

Lệnh gọi hệ thống select() cung cấp cơ chế để triển khai I/O đa luồng đồng bộ:

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select (int n,
            fd_set *readfds,
            fd_set *writefds,
            fd_set *exceptfds,
            struct timeval *timeout);

FD_CLR(int fd, fd_set *đặt);
FD_ISSET(int fd, fd_set *đặt);
FD_SET(int fd, fd_set *đặt);
FD_ZERO(fd_set *bộ);
```

Lệnh gọi select() sẽ chặn cho đến khi các mô tả tệp đã cho sẵn sàng thực hiện I/O hoặc cho đến khi hết thời gian chờ tùy chọn.

Các mô tả tệp được theo dõi được chia thành ba tập hợp, mỗi tập hợp chờ một sự kiện khác nhau. Các mô tả tệp được liệt kê trong tập hợp readfds được theo dõi để xem dữ liệu có sẵn để đọc hay không (tức là, nếu một hoạt động đọc sẽ hoàn tất mà không bị chặn). Các mô tả tệp được liệt kê trong tập hợp writefds được theo dõi để xem hoạt động ghi sẽ hoàn tất mà không bị chặn hay không. Cuối cùng, các mô tả tệp trong tập hợp exceptfds được theo dõi để xem có ngoại lệ nào xảy ra hay không hoặc có dữ liệu ngoài băng tần nào khả dụng không (các trạng thái này chỉ áp dụng cho các ổ cắm). Một tập hợp nhất định có thể là NULL, trong trường hợp đó select() không theo dõi điều đó sự kiện.

Khi trả về thành công, mỗi tập hợp được sửa đổi sao cho chỉ chứa các mô tả tệp đã sẵn sàng cho I/O thuộc loại được phân định bởi tập hợp đó. Ví dụ, giả sử hai mô tả tệp, với các giá trị 7 và 9, được đặt trong tập hợp readfds . Khi lệnh gọi trả về, nếu 7 vẫn còn trong tập hợp, mô tả tệp đó đã sẵn sàng để đọc mà không cần chặn. Nếu 9 không còn trong tập hợp nữa, có lẽ không thể đọc được mà không cần chặn. (Tôi nói

www.it-ebooks.info

Machine Translated by Google

có lẽ ở đây vì có thể dữ liệu đã có sẵn sau khi cuộc gọi hoàn tất. Trong trường hợp đó, một cuộc gọi tiếp theo đến select() sẽ trả về mô tả tệp là sẵn sàng để đọc.*)

Tham số đầu tiên, n, bằng giá trị của mô tả tệp có giá trị cao nhất trong bất kỳ tập hợp nào, cộng với một. Do đó, người gọi select () có trách nhiệm kiểm tra mô tả tệp nào được đưa ra là có giá trị cao nhất và truyền giá trị đó cộng với một cho tham số đầu tiên.

Tham số timeout là con trỏ tới cấu trúc timeval được định nghĩa như sau:

```
#include <hệ thống/thời gian.h>

struct timeval
        { giây tv              /* giây */ /*
        dài; giây tv dài;      micro giây */
};
```

Nếu tham số này không phải là NULL, lệnh gọi select() sẽ trả về sau tv_sec giây và tv_usec microseconds, ngay cả khi không có mô tả tệp nào sẵn sàng cho I/O. Khi trả về, trạng thái của cấu trúc này trên nhiều hệ thống Unix khác nhau là không xác định và do đó, nó phải được khởi tạo lại (cùng với các tập mô tả tệp) trước mỗi lần gọi. Thật vậy, các phiên bản Linux hiện tại tự động sửa đổi tham số này, đặt các giá trị thành thời gian còn lại. Do đó, nếu thời gian chờ được đặt là 5 giây và 3 giây trôi qua trước khi mô tả tệp sẵn sàng, tv.tv_sec sẽ chứa 2 khi lệnh gọi

trở lại.

Nếu cả hai giá trị trong thời gian chờ được đặt thành 0, cuộc gọi sẽ trả về ngay lập tức, báo cáo mọi sự kiện đang chờ xử lý tại thời điểm cuộc gọi, nhưng không chờ bất kỳ sự kiện nào sau đó.

Các tập hợp các mô tả tệp không được thao tác trực tiếp mà được quản lý thông qua các macro trợ giúp. Điều này cho phép các hệ thống Unix triển khai các tập hợp theo bất kỳ cách nào chúng muốn. Tuy nhiên, hầu hết các hệ thống triển khai các tập hợp dưới dạng các mảng bit đơn giản. FD_ZERO xóa tất cả các mô tả tệp khỏi tập hợp đã chỉ định. Nó phải được gọi trước mỗi lần gọi select( ):

```
fd_đặt lệnh ghifds;

FD_ZERO(&writefds);
```

FD_SET thêm một mô tả tệp vào một tập hợp nhất định và FD_CLR xóa một mô tả tệp khỏi một tập hợp nhất định:

```
FD_SET(fd, &writefds);          /* thêm 'fd' vào tập hợp */ /
FD_CLR(fd, &writefds);          * ôi, xóa 'fd' khỏi tập hợp */
```

* Điều này là do select( ) và poll( ) được kích hoạt theo mức chứ không phải kích hoạt theo cạnh. epoll( ), mà chúng ta sẽ thảo luận trong Chương 4, có thể hoạt động ở cả hai chế độ. Hoạt động kích hoạt theo cạnh đơn giản hơn, nhưng cho phép bỏ qua các sự kiện I/O nếu không cẩn thận.

www.it-ebooks.info

Machine Translated by Google

Mã đư ợc thiết kế tốt không bao giờ phải sử dụng FD_CLR và hiếm khi, nếu có, sử dụng.

FD_ISSET kiểm tra xem một mô tả tệp có phải là một phần của một tập hợp nhất định hay không. Nó trả về một số nguyên khác không nếu mô tả tệp nằm trong tập hợp và trả về 0 nếu không. FD_ISSET đư ợc sử dụng sau khi gọi select() trả về để kiểm tra xem một mô tả tệp nhất định đã sẵn sàng để hành động hay chư a:

```
if (FD_ISSET(fd, &readfds)) /
        * 'fd' có thể đọc đư ợc mà không bị chặn!
```

*/ Vì các tập mô tả tệp đư ợc tạo tĩnh, chúng áp đặt giới hạn về số lư ợng tối đa các mô tả tệp và mô tả tệp có giá trị lớn nhất có thể đư ợc đặt bên trong chúng, cả hai đều đư ợc chỉ định bởi FD_SETSIZE. Trên Linux, giá trị này là 1.024. Chúng ta sẽ xem xét hậu quả của giới hạn này sau trong chư ơ ng này.

Giá trị trả về và mã lỗi

Khi thành công, select() trả về số lư ợng mô tả tệp sẵn sàng cho I/O, trong số cả ba tập hợp. Nếu thời gian chờ đư ợc cung cấp, giá trị trả về có thể là 0. Khi có lỗi, lệnh gọi trả về -1 và errno đư ợc đặt thành một trong các giá trị sau:

EBADF

Một mô tả tập tin không hợp lệ đã đư ợc cung cấp trong một trong các tập hợp.

EINTR

Đã bắt đư ợc tín hiệu trong khi chờ và cuộc gọi có thể đư ợc phát lại.

EINVAL

Tham số n là số âm hoặc thời gian chờ đã cho không hợp lệ.

ENOMEM

Không đủ bộ nhớ để hoàn tất yêu cầu.

Ví dụ select() Hãy

xem xét một chư ơ ng trình ví dụ, đơ n giản như ng đầy đủ chức năng, để minh họa việc sử dụng select( ). Ví dụ này chặn việc chờ đầu vào trên stdin trong tối đa 5 giây.
Bởi vì nó chỉ theo dõi một mô tả tệp duy nhất, nên nó không thực sự ghép kênh I/O, như ng cách sử dụng lệnh gọi hệ thống đư ợc làm rõ:

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

#define TIMEOUT 5           /* chọn thời gian chờ tính bằng giây */ /
#define BUF_LEN 1024        * đọc bộ đệm tính bằng byte */

int main (void)
{
        cấu trúc timeval
        tv; fd_set
        readfds; int ret;
```


www.it-ebooks.info

Machine Translated by Google

```
        /* Chờ stdin để nhập dữ liệu. */
        FD_ZERO(&readfds);
        FD_SET(STDIN_FILENO, &readfds);

        /* Chờ tối đa năm giây. */
        tv.tv_sec = TIMEOUT;
        tv.tv_usec = 0;

        /* Được rồi, bây giờ chặn! */
        ret = select (STDIN_FILENO + 1,
                      &readfds,
                      NULL,
                      NULL,
                      &tv);
        if (ret == -1)
                { perror ("select");
                return
        1; } else if (!ret)
                { printf ("%d giây đã trôi qua.\n", TIMEOUT);
                return 0;
        }

        /*
         * Mô tả tập tin của chúng ta đã sẵn sàng để đọc chưa?
         * (Phải vậy thôi, vì đó là fd duy nhất mà chúng tôi cung
         *  cấp và cuộc gọi trả về giá trị khác không,
         *  nhưng chúng tôi sẽ tự giải thích.) */

        nếu (FD_ISSET(STDIN_FILENO, &readfds))
                { char buf[BUF_LEN+1];
                int len;

                /* đảm bảo không chặn */ len =
                read (STDIN_FILENO, buf, BUF_LEN); if (len
                == -1) { perror
                         ("read"); return
                         1;
                }

                nếu (len)
                         { buf[len] =
                         '\0'; printf ("đọc: %s\n", buf);
                }

                trả về 0;
        }

        fprintf (stderr, "Điều này không nên xảy ra!\n"); trả
        về 1;
}
```


www.it-ebooks.info

Machine Translated by Google

Ngủ di động với select( )

Bởi vì select( ) trư ớc đây đã đư ợc triển khai dễ dàng hơ n trên nhiều hệ thống Unix so với cơ chế ngủ độ phân giải dư ới một giây, nên nó thư ờng đư ợc sử dụng như một cách di động để ngủ bằng cách cung cấp thời gian chờ không NULL như ng NULL cho cả ba tập hợp:

```
cấu trúc thời gian tv;

tv.tv_sec = 0;
tv.tv_usec = 500;

/* ngủ trong 500 micro giây */
select (0, NULL, NULL, NULL, &tv);
```

Tất nhiên, Linux cung cấp giao diện cho chế độ ngủ độ phân giải cao. Chúng tôi sẽ đề cập đến những điều này trong Chư ơ ng 10.

pselect( )

Lệnh gọi hệ thống select() , lần đầu tiên đư ợc giới thiệu TRONG 4.2BSD, rất phổ biến, như ng POSIX đã định nghĩa giải pháp riêng của mình, pselect( ), trong POSIX 1003.1g-2000 và sau đó là trong POSIX 1003.1-2001:

```
#define _XOPEN_SOURCE 600
#include <sys/select.h>

int pselect (int n,
             fd_set *readfds,
             fd_set *writefds,
             fd_set *exceptfds,
             const struct timespec *timeout,
             const sigset_t *sigmask);

FD_CLR(int fd, fd_set *đặt);
FD_ISSET(int fd, fd_set *đặt);
FD_SET(int fd, fd_set *đặt);
FD_ZERO(fd_set *bộ);
```

Có ba điểm khác biệt giữa pselect() và select( ):

1. pselect() sử dụng cấu trúc timespec , không phải cấu trúc timeval , cho tham số timeout của nó. Cấu trúc timespec sử dụng giây và nano giây, không phải giây và micro giây, cung cấp độ phân giải timeout vư ợt trội về mặt lý thuyết. Tuy nhiên, trong thực tế, không có lệnh gọi nào cung cấp độ phân giải micro giây một cách đáng tin cậy.
2. Lệnh gọi pselect() không sửa đổi tham số timeout . Do đó, tham số này không cần phải đư ợc khởi tạo lại trong các lần gọi tiếp theo.
3. Lệnh gọi hệ thống select() không có tham số sigmask . Đối với tín hiệu, khi tham số này đư ợc đặt thành NULL, pselect() hoạt động giống như select( ).

Cấu trúc timespec đư ợc định nghĩa như sau:

```
#include <hệ thống/thời gian.h>

cấu trúc timespec {
```


www.it-ebooks.info

Machine Translated by Google

```
        tv_sec dài;             /* giây */
        tv_nsec dài;            /* nano giây */
};
```

Động lực chính thúc đẩy việc bổ sung pselect() vào hộp công cụ của Unix là việc bổ sung tham số sigmask , nhằm giải quyết tình trạng chạy đua giữa đang chờ mô tả tập tin và tín hiệu (tín hiệu đư ợc trình bày chi tiết trong Chư ơ ng 9). Giả sử rằng trình xử lý tín hiệu đặt cờ toàn cục (như hầu hết các trình xử lý khác) và quy trình kiểm tra cờ này trư ớc khi gọi select( ). Bây giờ, giả sử rằng tín hiệu đến sau kiểm tra, như ng trư ớc khi gọi. Ứng dụng có thể chặn vô thời hạn và không bao giờ phản hồi đến cờ thiết lập. Lệnh gọi pselect() giải quyết vấn đề này bằng cách cho phép ứng dụng gọi pselect( ), cung cấp một tập hợp các tín hiệu để chặn. Các tín hiệu bị chặn không đư ợc xử lý cho đến khi chúng đư ợc bỏ chặn. Khi pselect() trả về, hạt nhân sẽ khôi phục tín hiệu cũ mặt nạ. Nghiêm túc mà nói, hãy xem Chư ơ ng 9.

Cho đến phiên bản kernel 2.6.16, việc triển khai pselect() của Linux không phải là một hệ thống call, như ng một wrapper đơ n giản xung quanh select( ) do glibc cung cấp. Wrapper này đã giảm thiểu tối đa—như ng không loại bỏ hoàn toàn—rủi ro xảy ra tình trạng chạy đua này. Với với sự ra đời của một hệ thống gọi thực sự, cuộc đua đã kết thúc.

Bất chấp những cải tiến (tư ơ ng đối nhỏ) trong pselect( ) , hầu hết các ứng dụng vẫn tiếp tục sử dụng select( ) , có thể là do thói quen hoặc vì tính di động cao hơ n.

thăm dò ( )

Cuộc gọi hệ thống poll() là giải pháp I/O ghép kênh của System V. Nó giải quyết một số thiếu sót trong select( ), mặc dù select() vẫn thư ờng đư ợc sử dụng (một lần nữa, rất có thể là ngoài thói quen, hoặc theo tên của tính di động):

```
#include <sys/poll.h>

int poll (struct pollfd *fds, unsigned int nfds, int timeout);
```

Không giống như select( ), với ba bộ mô tả tệp dựa trên bitmask không hiệu quả, poll() sử dụng một mảng duy nhất các cấu trúc pollfd nfds , đư ợc trỏ tới bởi fds. Cấu trúc đư ợc định nghĩa như sau:

```
#include <sys/poll.h>

cấu trúc pollfd {
        int fd;             /* mô tả tập tin */
        sự kiện ngắn;       /* các sự kiện đư ợc yêu cầu để xem */
        sự kiện ngắn;       /* trả về các sự kiện đã chứng kiến */
};
```

Mỗi cấu trúc pollfd chỉ định một mô tả tệp duy nhất để theo dõi. Nhiều cấu trúc có thể đư ợc thông qua, hư ớng dẫn poll() để theo dõi nhiều mô tả tệp. Trư ờng sự kiện của mỗi cấu trúc là một bitmask của các sự kiện cần theo dõi trên mô tả tệp đó. Ngư ời dùng

www.it-ebooks.info

Machine Translated by Google

đặt trư ờng này. Trư ờng revents là một bitmask của các sự kiện đã đư ợc chứng kiến trên mô tả tệp.

Kernel đặt trư ờng này khi trả về. Tất cả các sự kiện đư ợc yêu cầu trong trư ờng sự kiện có thể đư ợc trả về trong trư ờng revents . Các sự kiện hợp lệ như sau:

PHẤN HOA
: Có dữ liệu để đọc.

BÌNH CHỌN
: Có dữ liệu bình thư ờng để đọc.

BĂNG ĐEO BÓNG ĐỎ
: Có dữ liệu ư u tiên để đọc.

BÌNH CHỌN
: Có dữ liệu khẩn cấp cần đọc.

Ô NHIỄM
: Viết sẽ không bị chặn.

BÌNH LUẬN
: Việc ghi dữ liệu bình thư ờng sẽ không bị chặn.

POLLWBAND
: Việc ghi dữ liệu ư u tiên sẽ không bị chặn.

POLLMSG
: Đã có tin nhắn SIGPOLL .

Ngoài ra, các sự kiện sau đây có thể đư ợc trả về trong trư ờng revents :

Lỗi
: POLLER trên mô tả tập tin đã cho.

POLLHUP
: Sự kiện bị treo trên mô tả tệp đã cho.

POLLNVAL
: Mô tả tệp đã cho không hợp lệ.

Những sự kiện này không có ý nghĩa trong trư ờng sự kiện , vì chúng luôn đư ợc trả về nếu có thể áp dụng. Với poll( ), không giống như select( ), bạn không cần yêu cầu rõ ràng về việc báo cáo các ngoại lệ.

POLLIN | POLLPRI tư ơ ng đư ơ ng với sự kiện đọc của select( ), và POLLOUT | POLLWRBAND tư ơ ng đư ơ ng với sự kiện ghi của select( ). POLLIN tư ơ ng đư ơ ng với POLLRDNORM | POLLRDBAND, và POLLOUT tư ơ ng đư ơ ng với POLLWRNORM.

Ví dụ, để theo dõi một mô tả tệp cho cả khả năng đọc và khả năng ghi, chúng ta sẽ đặt các sự kiện thành POLLIN | POLLOUT. Khi trả về, chúng ta sẽ kiểm tra các sự kiện cho các cờ này trong cấu trúc tư ơ ng ứng với mô tả tệp đang xét. Nếu POLLIN đư ợc đặt, mô tả tệp sẽ có thể đọc đư ợc mà không bị chặn. Nếu POLLOUT đư ợc đặt, mô tả tệp sẽ có thể ghi đư ợc mà không bị chặn. Các cờ không loại trừ lẫn nhau: cả hai đều có thể đư ợc đặt, biểu thị rằng cả lệnh đọc và lệnh ghi sẽ trả về thay vì chặn trên mô tả tệp đó.

www.it-ebooks.info

Machine Translated by Google

Tham số timeout chỉ định khoảng thời gian chờ, tính bằng mili giây, trư ớc khi trả về bất kể bất kỳ I/O nào đã sẵn sàng. Giá trị âm biểu thị thời gian chờ vô hạn. Giá trị 0 chỉ thị lệnh gọi trả về ngay lập tức, liệt kê bất kỳ mô tả tệp nào có I/O đã sẵn sàng đang chờ xử lý, như ng không chờ bất kỳ sự kiện nào khác. Theo cách này, poll() đúng với tên của nó, thăm dò một lần và trả về ngay lập tức.

Giá trị trả về và mã lỗi

Khi thành công, poll() trả về số lư ợng các mô tả tệp có cấu trúc có các trư ờng revents khác không . Nó trả về 0 nếu thời gian chờ xảy ra trư ớc khi bất kỳ sự kiện nào xảy ra.
Khi thất bại, -1 đư ợc trả về và errno đư ợc đặt thành một trong những giá trị sau:

EBADF

Một mô tả tập tin không hợp lệ đã đư ợc đư a ra trong một hoặc nhiều cấu trúc.

EFAULT

Con trỏ tới fds trỏ ra ngoài không gian địa chỉ của tiến trình.

TRỰC TIẾP

Có tín hiệu xảy ra trư ớc bất kỳ sự kiện nào đư ợc yêu cầu. Cuộc gọi có thể đư ợc phát lại.

EINVAL

Tham số nfds vư ợt quá giá trị RLIMIT_NOFILE .

ENOMEM

Không đủ bộ nhớ để hoàn tất yêu cầu.

ví dụ poll()

Hãy xem một chư ơ ng trình ví dụ sử dụng poll() để đồng thời kiểm tra xem lệnh đọc từ stdin và lệnh ghi vào stdout có bị chặn không:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/poll.h>

#define THỜI GIAN CHỜ 5          /* thời gian chờ thăm dò, tính bằng giây */

int main (void) {

            cấu trúc pollfd fds[2]; int
            ret;

            /* theo dõi stdin để biết đầu vào */
            fds[0].fd = STDIN_FILENO;
            fds[0].events = POLLIN;

            /* theo dõi stdout để biết khả năng ghi (gần như luôn đúng) */ fds[1].fd =
            STDOUT_FILENO; fds[1].events =
            POLLOUT;

            /* Đã xong, chặn! */ ret =
            poll (fds, 2, TIMEOUT * 1000);
```


www.it-ebooks.info

Machine Translated by Google

```
            nếu (ret == -1)
                    { perror ("thăm
                    dò"); trả về 1;
            }

            nếu (!ret)
                    { printf ("%d giây đã trôi qua.\n", TIMEOUT); trả
                    về 0;
            }

            nếu (fds[0].revents & POLLIN)
                    printf ("stdin có thể đọc đư ợc\n");

            nếu (fds[1].revents & POLLOUT)
                    printf ("stdout có thể ghi đư ợc\n");

            trả về 0;
    }
```

Chạy lệnh này, chúng ta sẽ nhận đư ợc kết quả sau như mong đợi:

```
$ ./poll
stdout có thể ghi đư ợc
```

Chạy lại lần nữa, như ng lần này chuyển hư ớng tệp vào chuẩn, chúng ta thấy cả hai sự kiện:

```
$ ./poll < ode_to_my_parrot.txt stdin có thể
đọc đư ợc stdout có thể
ghi đư ợc
```

Nếu chúng ta sử dụng poll() trong một ứng dụng thực tế, chúng ta sẽ không cần phải xây dựng lại các cấu trúc pollfd trên mỗi lần gọi. Cấu trúc tư ơ ng tự có thể đư ợc truyền lặp lại; hạt nhân sẽ xử lý việc đư a trư ờng revents về 0 khi cần.

ppoll ( )

Linux cung cấp một ppoll() tư ơ ng tự như poll( ), tư ơ ng tự như pselect( ). Tuy nhiên, không giống như pselect( ), ppoll() là một giao diện dành riêng cho Linux:

```
#define _GNU_SOURCE
#include <sys/poll.h>

int ppoll (struct pollfd *fds,
            nfds_t nfds,
            const struct timespec *timeout,
            const sigset_t *sigmask);
```

Giống như pselect( ), tham số timeout chỉ định giá trị thời gian chờ tính bằng giây và nano giây, còn tham số sigmask cung cấp một tập hợp các tín hiệu để chờ.

www.it-ebooks.info

Machine Translated by Google

poll( ) so với select( )

Mặc dù thực hiện cùng một công việc cơ bản, lệnh gọi hệ thống poll() lại vượt trội hơn select() vì một số lý do sau:

- poll() không yêu cầu người dùng tính toán và truyền vào dưới dạng tham số giá trị của bộ mô tả tệp có số cao nhất cộng với một.
- poll( ) hiệu quả hơn đối với các mô tả tệp có giá trị lớn. Hãy tưởng tượng việc xem một mô tả tệp đơn lẻ có giá trị 900 thông qua select( )—kernel sẽ phải kiểm tra từng bit của mỗi tập hợp được truyền vào, lên đến bit thứ 900.
- các tập mô tả tệp của select( ) có kích thước tĩnh, đưa ra một sự đánh đổi: chúng nhỏ, giới hạn mô tả tệp tối đa mà select( ) có thể theo dõi hoặc chúng không hiệu quả. Các thao tác trên các mặt nạ bit lớn không hiệu quả, đặc biệt là nếu không biết chúng có được điền thưa thớt hay không.* Với poll( ), người ta có thể tạo một mảng có kích thước chính xác. Chỉ theo dõi một mục? Chỉ cần truyền vào một
  kết cấu.
- Với select( ), các tập mô tả tệp được xây dựng lại khi trả về, do đó mỗi lệnh gọi tuần tự phải khởi tạo lại chúng. Lệnh gọi hệ thống poll() tách đầu vào (trường sự kiện) khỏi đầu ra (trường sự kiện), cho phép mảng được sử dụng lại mà không cần thay đổi.
- Tham số timeout cho select() không được xác định khi trả về. Mã di động cần phải khởi tạo lại nó. Tuy nhiên, đây không phải là vấn đề với pselect( ).

Tuy nhiên, lệnh gọi hệ thống select() có một số ưu điểm sau:

- select( ) di động hơn, vì một số hệ thống Unix không hỗ trợ poll( ). • select( )

  cung cấp độ phân giải thời gian chờ tốt hơn: xuống đến micro giây. Cả ppoll( ) và pselect( ) về mặt lý thuyết đều cung cấp độ phân giải nano giây, nhưng trên thực tế, không có lệnh gọi nào trong số này cung cấp độ phân giải thậm chí là micro giây một cách đáng tin cậy.

Vượt trội hơn cả poll() và select() là giao diện epoll , một giải pháp I/O đa luồng dành riêng cho Linux mà chúng ta sẽ xem xét trong Chương 4.

Nội dung bên trong của Kernel

Phần này xem xét cách hạt nhân Linux triển khai I/O, tập trung vào ba hệ thống con chính của hạt nhân: hệ thống tệp ảo (VFS), bộ đệm trang và ghi lại trang. Cùng nhau, các hệ thống con này giúp I/O liền mạch, hiệu quả và tối ưu.

* Nếu bitmask thường thưa thớt, mỗi từ tạo nên mặt nạ có thể được kiểm tra với số không; chỉ khi thao tác đó trả về false thì mới cần kiểm tra từng bit. Tuy nhiên, công việc này sẽ bị lãng phí nếu bitmask có mật độ dày đặc.

www.it-ebooks.info

Machine Translated by Google

Trong Chư ơ ng 4, chúng ta sẽ xem xét hệ thống con thứ tư , bộ lập lịch I/O.

Hệ thống tập tin ảo Hệ thống tập

tin ảo, đôi khi còn đư ợc gọi là công tắc tập tin ảo, là một cơ chế trừu tư ợng cho phép hạt nhân Linux gọi các hàm hệ thống tập tin và thao tác dữ liệu hệ thống tập tin mà không cần biết—hoặc thậm chí không cần quan tâm đến—loại hệ thống tập tin cụ thể đang đư ợc sử dụng.

VFS thực hiện sự trừu tư ợng này bằng cách cung cấp một mô hình tệp chung, là cơ sở cho tất cả các hệ thống tệp trong Linux. Thông qua các con trỏ hàm và nhiều phư ơ ng pháp hư ớng đối tư ợng khác nhau,* mô hình tệp chung cung cấp một khuôn khổ mà các hệ thống tệp trong hạt nhân Linux phải tuân thủ. Điều này cho phép VFS đư a ra các yêu cầu chung cho hệ thống tệp. Khuôn khổ cung cấp các móc để hỗ trợ việc đọc, tạo liên kết, đồng bộ hóa, v.v. Sau đó, mỗi hệ thống tệp sẽ đăng ký các hàm để xử lý các hoạt động mà nó có khả năng thực hiện.

Cách tiếp cận này buộc phải có một số điểm chung nhất định giữa các hệ thống tệp. Ví dụ, VFS nói về inode, superblock và mục nhập thư mục. Một hệ thống tệp không có nguồn gốc từ Unix, có thể không có các khái niệm giống Unix như inode, chỉ đơ n giản là phải đối phó. Thật vậy, chúng đối phó: Linux hỗ trợ các hệ thống tệp như FAT và NTFS mà không có vấn đề gì.

Lợi ích của VFS là rất lớn. Một lệnh gọi hệ thống duy nhất có thể đọc từ bất kỳ hệ thống tập tin nào trên bất kỳ phư ơ ng tiện nào; một tiện ích duy nhất có thể sao chép từ bất kỳ hệ thống tập tin nào sang bất kỳ hệ thống tập tin nào khác. Tất cả các hệ thống tập tin đều hỗ trợ cùng một khái niệm, cùng một giao diện và cùng một lệnh gọi. Mọi thứ đều hoạt động và hoạt động tốt.

Khi một ứng dụng phát lệnh gọi hệ thống read() , nó sẽ trải qua một hành trình thú vị. Thư viện C cung cấp các định nghĩa về lệnh gọi hệ thống đư ợc chuyển đổi thành các câu lệnh bẫy thích hợp tại thời điểm biên dịch. Khi một quy trình không gian ngư ời dùng bị bẫy vào nhân, đư ợc truyền qua trình xử lý lệnh gọi hệ thống và đư ợc chuyển đến lệnh gọi hệ thống read() , nhân sẽ tìm ra đối tư ợng nào sao lư u mô tả tệp đã cho. Sau đó, nhân sẽ gọi hàm read đư ợc liên kết với đối tư ợng sao lư u. Đối với hệ thống tệp, hàm này là một phần của mã hệ thống tệp. Sau đó, hàm thực hiện công việc của nó—ví dụ, đọc dữ liệu vật lý từ hệ thống tệp—và trả về dữ liệu cho lệnh gọi read() không gian ngư ời dùng , sau đó trả về trình xử lý lệnh gọi hệ thống, sao chép dữ liệu trở lại không gian ngư ời dùng, tại đó lệnh gọi hệ thống read() trả về và quy trình tiếp tục thực thi.

* Có, trong C.

www.it-ebooks.info

Machine Translated by Google

Đối với các lập trình viên hệ thống, các nhánh của VFS rất quan trọng. Các lập trình viên không cần phải lo lắng về loại hệ thống tập tin hoặc phư ơ ng tiện mà tập tin nằm trên đó. Các lệnh gọi hệ thống chung—read( ) , write( ) , v.v.—có thể thao tác các tập tin trên bất kỳ hệ thống tập tin nào đư ợc hỗ trợ và trên bất kỳ phư ơ ng tiện nào đư ợc hỗ trợ.

Bộ đệm trang Bộ đệm

trang là một kho lư u trữ trong bộ nhớ dữ liệu đư ợc truy cập gần đây từ một hệ thống tệp trên đĩa. Truy cập đĩa chậm một cách đau đớn, đặc biệt là so với tốc độ bộ xử lý ngày nay. Lư u trữ dữ liệu đư ợc yêu cầu trong bộ nhớ cho phép hạt nhân thực hiện các yêu cầu tiếp theo cho cùng một dữ liệu từ bộ nhớ, tránh truy cập đĩa nhiều lần.

Bộ nhớ đệm trang khai thác khái niệm về vị trí thời gian, một loại vị trí tham chiếu, nói rằng một tài nguyên đư ợc truy cập tại một thời điểm có khả năng cao sẽ đư ợc truy cập lại trong tư ơ ng lai gần. Do đó, bộ nhớ đư ợc sử dụng để lư u trữ dữ liệu trong lần truy cập đầu tiên sẽ có lợi, vì nó ngăn chặn các lần truy cập đĩa tốn kém trong tư ơ ng lai.

Bộ đệm trang là nơ i đầu tiên mà nhân tìm kiếm dữ liệu hệ thống tệp. Nhân gọi hệ thống con bộ nhớ để đọc dữ liệu từ đĩa chỉ khi không tìm thấy dữ liệu trong bộ đệm. Do đó, lần đầu tiên bất kỳ mục dữ liệu nào đư ợc đọc, dữ liệu đó sẽ đư ợc chuyển từ đĩa vào bộ đệm trang và đư ợc trả về ứng dụng từ bộ đệm. Nếu dữ liệu đó sau đó đư ợc đọc lại, dữ liệu đó chỉ đư ợc trả về từ bộ đệm. Tất cả các hoạt động đều đư ợc thực hiện một cách minh bạch thông qua bộ đệm trang, đảm bảo rằng dữ liệu của nó có liên quan và luôn hợp lệ.

Bộ đệm trang Linux có kích thư ớc động. Khi các hoạt động I/O đư a ngày càng nhiều dữ liệu vào bộ nhớ, bộ đệm trang sẽ ngày càng lớn hơ n, tiêu thụ bất kỳ bộ nhớ trống nào. Nếu bộ đệm trang cuối cùng tiêu thụ hết bộ nhớ trống và một phân bổ đư ợc cam kết yêu cầu thêm bộ nhớ, bộ đệm trang sẽ đư ợc cắt tỉa, giải phóng các trang ít đư ợc sử dụng nhất của nó, để tạo chỗ cho việc sử dụng bộ nhớ "thực". Việc cắt tỉa này diễn ra liền mạch và tự động. Bộ đệm có kích thư ớc động cho phép Linux sử dụng toàn bộ bộ nhớ trong hệ thống và lư u trữ càng nhiều dữ liệu càng tốt.

Tuy nhiên, thông thư ờng, việc hoán đổi một phần dữ liệu ít sử dụng sang đĩa sẽ hợp lý hơ n là cắt bớt một phần bộ đệm trang thư ờng dùng có thể đư ợc đọc lại vào bộ nhớ trong yêu cầu đọc tiếp theo (hoán đổi cho phép nhân lư u trữ dữ liệu trên đĩa, để có diện tích bộ nhớ lớn hơ n RAM của máy).

Nhân Linux triển khai các phư ơ ng pháp tìm kiếm để cân bằng việc hoán đổi dữ liệu so với việc cắt tỉa bộ đệm trang (và các dự trữ trong bộ nhớ khác). Các phư ơ ng pháp tìm kiếm này có thể quyết định hoán đổi dữ liệu ra đĩa thay vì cắt tỉa bộ đệm trang, đặc biệt là nếu dữ liệu đư ợc hoán đổi không đư ợc sử dụng.

Sự cân bằng giữa hoán đổi và bộ nhớ đệm đư ợc điều chỉnh thông qua /proc/sys/vm/swappiness. Tệp ảo này có giá trị từ 0 đến 100, với giá trị mặc định là 60. Giá trị cao hơ n ngụ ý sở thích mạnh hơ n đối với việc giữ bộ nhớ đệm trang trong bộ nhớ và hoán đổi dễ dàng hơ n. Giá trị thấp hơ n ngụ ý sở thích mạnh hơ n đối với việc cắt tỉa bộ nhớ đệm trang và không hoán đổi.

www.it-ebooks.info

Machine Translated by Google

Một dạng khác của địa phương tham chiếu là địa phương tuần tự, nói rằng dữ liệu thường được tham chiếu tuần tự. Để tận dụng nguyên tắc này, hạt nhân cũng triển khai đọc trước bộ đệm trang. Đọc trước là hành động đọc dữ liệu bổ sung từ đĩa và vào bộ đệm trang sau mỗi yêu cầu đọc—trên thực tế, đọc trước một chút. Khi hạt nhân đọc một khối dữ liệu từ đĩa, nó cũng đọc khối tiếp theo hoặc hai khối tiếp theo. Đọc các khối dữ liệu tuần tự lớn cùng một lúc là hiệu quả, vì đĩa thường không cần tìm kiếm. Ngoài ra, hạt nhân có thể thực hiện yêu cầu đọc trước trong khi quy trình đang thao tác khối dữ liệu đọc đầu tiên. Nếu, như thường xảy ra, quy trình tiếp tục gửi yêu cầu đọc mới cho khối tiếp theo, hạt nhân có thể chuyển giao dữ liệu từ lần đọc trước ban đầu mà không cần phải đưa ra yêu cầu I/O đĩa.

Giống như bộ đệm trang, hạt nhân quản lý readahead theo cách động. Nếu nó nhận thấy một quy trình liên tục sử dụng dữ liệu được đọc qua readahead, hạt nhân sẽ mở rộng cửa sổ readahead, do đó đọc trước ngày càng nhiều dữ liệu. Cửa sổ readahead có thể nhỏ tới 16 KB và lớn tới 128 KB. Ngược lại, nếu hạt nhân nhận thấy readahead không tạo ra bất kỳ kết quả hữu ích nào—tức là ứng dụng đang tìm kiếm xung quanh tệp và không đọc tuần tự—nó có thể vô hiệu hóa hoàn toàn readahead.

Sự hiện diện của bộ đệm trang được cho là minh bạch. Các lập trình viên hệ thống thường không thể tối ưu hóa mã của họ để tận dụng tốt hơn thực tế là bộ đệm trang tồn tại—ngoại trừ, có lẽ, không triển khai bộ đệm như vậy trong không gian người dùng của chính họ. Thông thường, mã hiệu quả là tất cả những gì cần thiết để sử dụng tốt nhất bộ đệm trang. Mặt khác, có thể sử dụng readahead. I/O tệp tuần tự luôn được ưu tiên hơn truy cập ngẫu nhiên, mặc dù không phải lúc nào cũng khả thi.

Viết lại trang

Như đã thảo luận trước đó trong “Hành vi của write(),” hạt nhân trì hoãn việc ghi thông qua bộ đệm. Khi một tiến trình đưa ra yêu cầu ghi, dữ liệu sẽ được sao chép vào bộ đệm và bộ đệm được đánh dấu là bẩn, biểu thị rằng bản sao trong bộ nhớ mới hơn bản sao trên đĩa. Yêu cầu ghi sau đó chỉ cần trả về. Nếu một yêu cầu ghi khác được thực hiện cho cùng một đoạn của tệp, bộ đệm sẽ được cập nhật với dữ liệu mới. Yêu cầu ghi ở nơi khác trong cùng một tệp sẽ tạo ra bộ đệm mới.

Cuối cùng, bộ đệm bẩn cần được cam kết vào đĩa, đồng bộ hóa các tệp trên đĩa với dữ liệu trong bộ nhớ. Điều này được gọi là ghi lại. Nó xảy ra trong hai tình huống:

- Khi bộ nhớ trống giảm xuống dưới ngưỡng có thể cấu hình, các bộ đệm bẩn sẽ được ghi lại vào đĩa để các bộ đệm sạch hiện tại có thể được xóa, giải phóng ký ức.
- Khi bộ đệm bẩn trở nên cũ hơn ngưỡng có thể cấu hình, bộ đệm sẽ được ghi quay lại đĩa. Điều này ngăn dữ liệu bị bẩn vô thời hạn.

www.it-ebooks.info

Machine Translated by Google

Writeback được thực hiện bởi một nhóm luồng kernel có tên là luồng pdflush (có lẽ là để xóa trang bẩn, nhưng ai mà biết được). Khi một trong hai điều kiện trước đó được đáp ứng, luồng pdflush sẽ thức dậy và bắt đầu cam kết bộ đệm bẩn vào đĩa cho đến khi không có điều kiện nào là đúng.

Có thể có nhiều luồng pdflush khởi tạo lệnh ghi lại cùng một lúc.
Điều này được thực hiện để tận dụng lợi ích của tính song song và để triển khai tránh tắc nghẽn. Tránh tắc nghẽn cố gắng ngăn các lệnh ghi được sao lưu trong khi chờ được ghi vào bất kỳ thiết bị khối nào. Nếu bộ đệm bẩn từ các thiết bị khối khác nhau tồn tại, các luồng pdflush khác nhau sẽ hoạt động để sử dụng đầy đủ từng thiết bị khối. Điều này khắc phục một thiếu sót trong các hạt nhân trước đó: tiền thân của các luồng pdflush (bdflush, một luồng đơn) có thể dành toàn bộ thời gian để chờ trên một thiết bị khối duy nhất, trong khi các thiết bị khối khác ở trạng thái nhàn rỗi. Trên một máy hiện đại, hạt nhân Linux hiện có thể giữ cho một số lượng lớn đĩa bị bão hòa.

Bộ đệm được biểu diễn trong nhân bởi cấu trúc dữ liệu buffer_head . Cấu trúc dữ liệu này theo dõi nhiều siêu dữ liệu liên quan đến bộ đệm, chẳng hạn như bộ đệm sạch hay bẩn. Nó cũng chứa một con trỏ đến dữ liệu thực tế. Dữ liệu này nằm trong bộ đệm trang. Theo cách này, hệ thống đệm và bộ đệm trang được hợp nhất.

Trong các phiên bản đầu của nhân Linux—trước 2.4—hệ thống đệm con tách biệt với bộ đệm trang, và do đó có cả bộ đệm trang và bộ đệm. Điều này ngụ ý rằng dữ liệu có thể tồn tại trong bộ đệm đệm (dưới dạng bộ đệm bẩn) và bộ đệm trang (dưới dạng dữ liệu được lưu trong bộ đệm) cùng một lúc. Đương nhiên, việc đồng bộ hóa hai bộ đệm riêng biệt này cần một số nỗ lực. Bộ đệm trang thống nhất được giới thiệu trong nhân Linux 2.4 là một cải tiến được hoan nghênh.

Deferred writes và hệ thống đệm trong Linux cho phép ghi nhanh, với cái giá phải trả là nguy cơ mất dữ liệu khi mất điện. Để tránh nguy cơ này, các ứng dụng quan trọng và hoang tưởng có thể sử dụng I/O đồng bộ (đã thảo luận trước đó trong chương này).

Phần kết luận

Chương này thảo luận về những điều cơ bản của lập trình hệ thống Linux: tệp I/O. Trên một hệ thống như Linux, cố gắng biểu diễn càng nhiều càng tốt dưới dạng tệp, điều rất quan trọng là phải biết cách mở, đọc, ghi và đóng tệp. Tất cả các hoạt động này đều là Unix cổ điển và được biểu diễn trong nhiều tiêu chuẩn.

Chương tiếp theo sẽ giải quyết I/O đệm và giao diện I/O chuẩn của thư viện C chuẩn. Thư viện C chuẩn không chỉ là tiện ích; việc đệm I/O trong không gian người dùng mang lại những cải tiến hiệu suất quan trọng.

www.it-ebooks.info