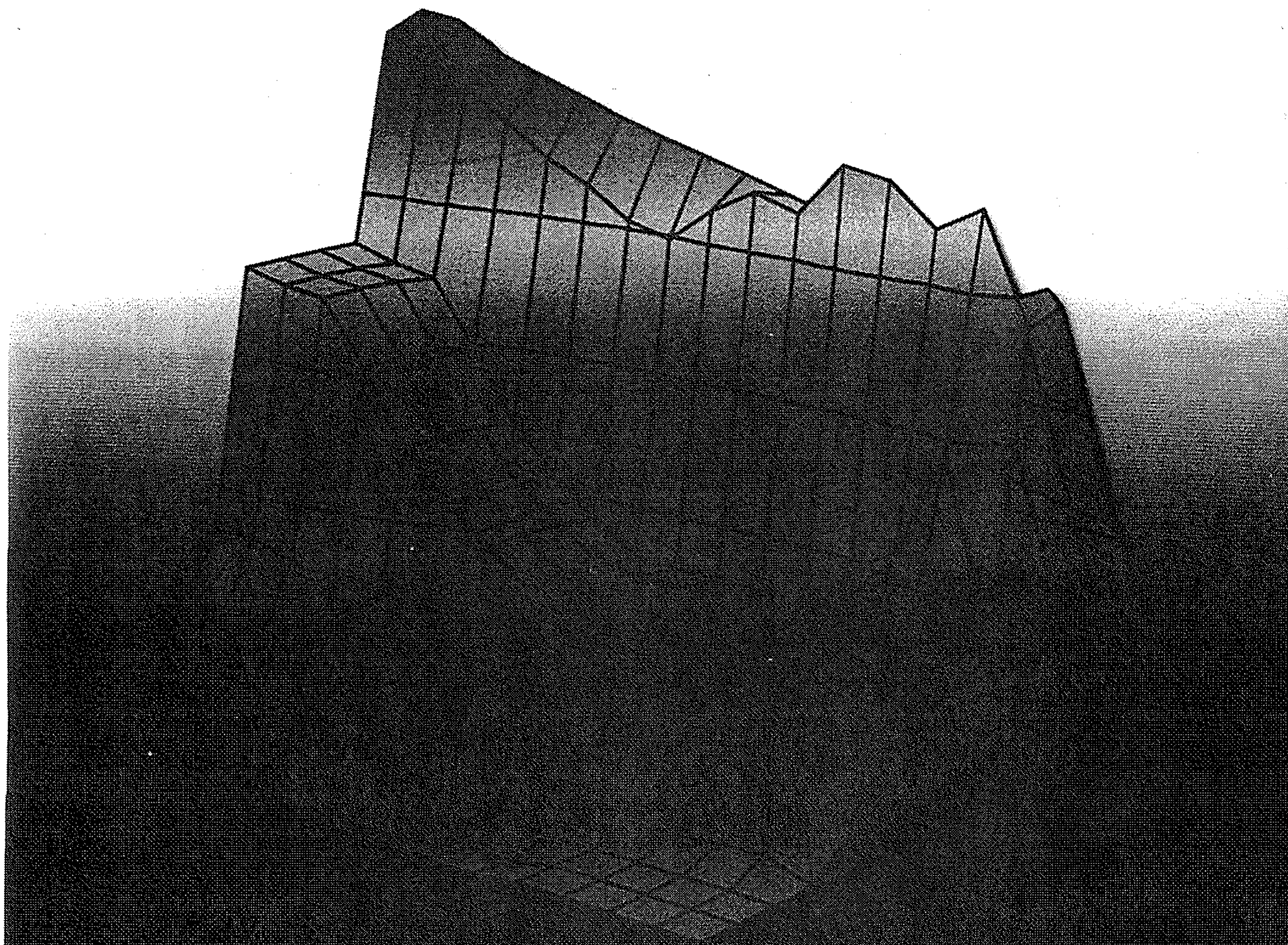
COMPUTER SYSTEMS

A PROGRAMMER'S PERSPECTIVE



Randal E. Bryant and David R. O'Hallaron

Library of Congress Cataloging-in-Publication Data

Bryant, Randal E.,
Computer Systems / Randal E. Bryant and David R. O'Hallaron
p. cm.
Includes bibliographical references and index.
ISBN 0-13-034074-X
1. Computer Systems. I. O'Hallaron, David R.
TA177.4.S85 2003
658.15-dc21 2001058726

Vice President and Editorial Director, ECS: *Marcia Horton*
Acquisitions Editor: *Eric Frank*
Editorial Assistant: *Delores J. Mars*
Vice President and Director of Production and Manufacturing, ESM: *David W. Riccardi*
Executive Managing Editor: *Vince O'Brien*
Managing Editor: *David A. George*
Production Editor: *Rose Kernan*
Director of Creative Services: *Paul Belfanti*
Creative Director: *Carole Anson*
Art Director/Cover Designer: *Jon Boylan*
Art Editor: *Xiaohong Zhu*
Manufacturing Manager: *Trudy Pisciotti*
Manufacturing Buyer: *Lisa McDowell*
Marketing Manager: *Holly Stark*

Prentice Hall

Pearson Education, Inc.
Upper Saddle River, New Jersey 07458

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-13-034074-X

Pearson Education Ltd., *London*
Pearson Education Australia Pty. Limited, *Sydney*
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd., *Hong Kong*
Pearson Education Canada, Inc., *Toronto*
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education—Japan, *Tokyo*
Pearson Education Malaysia, Pte. Ltd.
Pearson Education, Inc., *Upper Saddle River, New Jersey*

8.2 Processes

Exceptions provide the basic building blocks that allow the operating system to provide the notion of a *process*, one of the most profound and successful ideas in computer science.

When we run a program on a modern system, we are presented with the illusion that our program is the only one currently running in the system. Our program appears to have exclusive use of both the processor and the memory. The processor appears to execute the instructions in our program, one after the other, without interruption. Finally, the code and data of our program appear to be the only objects in the system's memory. These illusions are provided to us by the notion of a process.

The classic definition of a process is *an instance of a program in execution*. Each program in the system runs in the *context* of some process. The context consists of the state that the program needs to run correctly. This state includes the program's code and data stored in memory, its stack, the contents of its general-purpose registers, its program counter, environment variables, and the set of open file descriptors.

Each time a user runs a program by typing the name of an executable object file to the shell, the shell creates a new process and then runs the executable object file in the context of this new process. Application programs can also create new processes and run either their own code or other applications in the context of the new process.

A detailed discussion of how operating systems implement processes is beyond our scope. Instead, we will focus on the key abstractions that a process provides to the application:

- An independent *logical control flow* that provides the illusion that our program has exclusive use of the processor.
- A private address space that provides the illusion that our program has exclusive use of the memory system.

Let's look more closely at these abstractions.

8.2.1 Logical Control Flow

A process provides each program with the illusion that it has exclusive use of the processor, even though many other programs are typically running on the system. If we were to use a debugger to single step the execution of our program, we would observe a series of program counter (PC) values that corresponded exclusively to instructions contained in our program's executable object file or in shared objects linked into our program dynamically at run time. This sequence of PC values is known as a *logical control flow*.

Consider a system that runs three processes, as shown in Figure 8.10. The single physical control flow of the processor is partitioned into three *logical flows*, one for each process. Each vertical line represents a portion of the logical flow for a process. In the example, process A runs for a while, followed by B, which runs

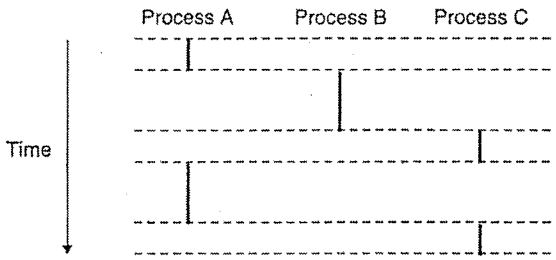


Figure 8.10
Logical control flows. Processes provide each program with the illusion that it has exclusive use of the processor. Each vertical bar represents a portion of the logical control flow for a process.

to completion. C then runs for awhile, followed by A, which runs to completion. Finally, C is able to run to completion.

The key point in Figure 8.10 is that processes take turns using the processor. Each process executes a portion of its flow and then is *preempted* (temporarily suspended) while other processes take their turns. To a program running in the context of one of these processes, it appears to have exclusive use of the processor. The only evidence to the contrary is that if we were to precisely measure the elapsed time of each instruction (see Chapter 9), we would notice that the CPU appears to periodically stall between the execution of some of the instructions in our program. However, each time the processor stalls, it subsequently resumes execution of our program without any change to the contents of the program's memory locations or registers.

In general, each logical flow is independent of any other flow in the sense that the logical flows associated with different processes do not affect the states of any other processes. The only exception to this rule occurs when processes use interprocess communication (IPC) mechanisms such as pipes, sockets, shared memory, and semaphores to explicitly interact with each other.

Any process whose logical flow overlaps in time with another flow is called a *concurrent process*, and the two processes are said to run *concurrently*. For example, in Figure 8.10, processes A and B run concurrently, as do A and C. On the other hand, B and C do not run concurrently because the last instruction of B executes before the first instruction of C.

The notion of processes taking turns with other processes is known as *multitasking*. Each time period that a process executes a portion of its flow is called a *time slice*. Thus, multitasking is also referred to as *time slicing*.

8.2.2 Private Address Space

A process also provides each program with the illusion that it has exclusive use of the system's address space. On a machine with n-bit addresses, the *address space* is the set of 2^n possible addresses, $0, 1, \ldots, 2^n - 1$. A process provides each program with its own *private address space*. This space is private in the sense that a byte of memory associated with a particular address in the space cannot in general be read or written by any other process.

Although the contents of the memory associated with each private address space is different in general, each such space has the same general organization.

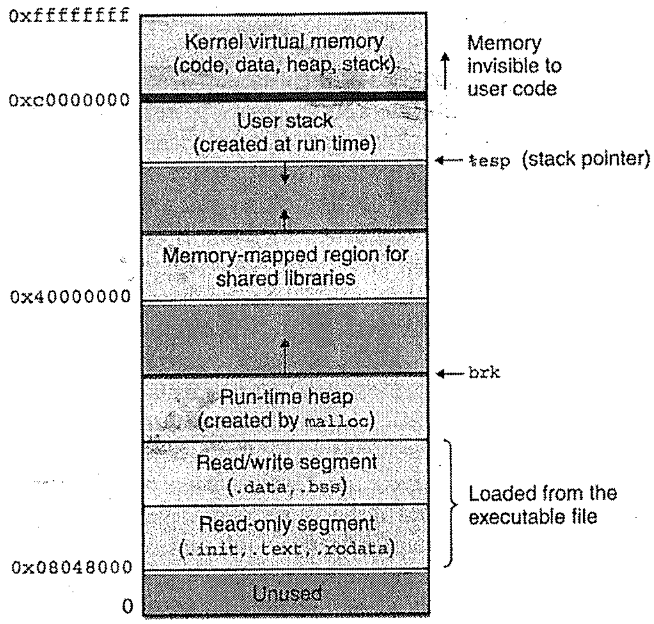


Figure 8.11
Process address space.

For example, Figure 8.11 shows the organization of the address space for a Linux process. The bottom three-fourths of the address space is reserved for the user program, with the usual text, data, heap, and stack segments. The top quarter of the address space is reserved for the kernel. This portion of the address space contains the code, data, and stack that the kernel uses when it executes instructions on behalf of the process (e.g., when the application program executes a system call).

8.2.3 User and Kernel Modes

In order for the operating system kernel to provide an airtight process abstraction, the processor must provide a mechanism that restricts the instructions that an application can execute, as well as the portions of the address space that it can access.

Processors typically provide this capability with a *mode bit* in some control register that characterizes the privileges that the process currently enjoys. When the mode bit is set, the process is running in *kernel mode* (sometimes called *supervisor mode*). A process running in kernel mode can execute any instruction in the instruction set and access any memory location in the system.

When the mode bit is not set, the process is running in *user mode*. A process in user mode is not allowed to execute *privileged instructions* that do things such as halt the processor, change the mode bit, or initiate an I/O operation. Nor is it allowed to directly reference code or data in the kernel area of the address space. Any such attempt results in a fatal protection fault. User programs must instead access kernel code and data indirectly via the system call interface.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.