INSTRUCTOR'S MANUAL
TO ACCOMPANY

OPERATING-SYSTEM CONCEPTS with Java

SEVENTH EDITION

ABRAHAM SILBERSCHATZ
Yale University

PETER BAER GALVIN
Corporate Technologies

GREG GAGNE
Westminster College

Preface

This volume is an instructor's manual for the Seventh Edition of *Operating System Concepts with Java*, by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. It consists of answers to the exercises in the parent text.

Although we have tried to produce an instructor's manual that will aid all of the users of our book as much as possible, there can always be improvements (improved answers, additional questions, sample test questions, programming projects, alternative orders of presentation of the material, additional references, and so on). We invite you to help us in improving this manual. If you have better solutions to the exercises or other items that would be of use with *Operating-System Concepts*, we invite you to send them to us for consideration in later editions of this manual. All contributions will, of course, be properly credited to their contributor.

Internet electronic mail should be addressed to osj-book@cs.yale.edu. Physical mail may be sent to Avi Silberschatz, Department of Computer Science, Yale University, 51 Prospect Street, New Haven, CT 06520, USA.

A. S.
P. B. G.
G. G.

Contents

Chapter 1	**Introduction**	1
Chapter 2	**Operating-System Structures**	9
Chapter 3	**Processes**	17
Chapter 4	**Threads**	21
Chapter 5	**CPU Scheduling**	25
Chapter 6	**Process Synchronization**	33
Chapter 7	**Deadlocks**	47
Chapter 8	**Memory Management**	57
Chapter 9	**Virtual Memory**	63
Chapter 10	**File-Systems Interface**	73
Chapter 11	**File-Systems Implementation**	77
Chapter 12	**Mass Storage Structure**	83
Chapter 13	**I/O Systems**	99
Chapter 14	**Protection**	105
Chapter 15	**Security**	111
Chapter 16	**Network Structures**	117
Chapter 17	**Distributed Communication**	123
Chapter 18	**Distributed File Systems**	127
Chapter 19	**Multimedia Systems**	133
Chapter 20	**Embedded Systems**	137
Chapter 21	**The Linux System**	143
Chapter 22	**Windows XP**	151
Chapter 23	**Influential Operating Systems**	155

Introduction

Chapter 1 introduces the general topic of operating systems and a handful of important concepts (multiprogramming, time sharing, distributed system, and so on). The purpose is to show *why* operating systems are what they are by showing *how* they developed. In operating systems, as in much of computer science, we are led to the present by the paths we took in the past, and we can better understand both the present and the future by understanding the past.

Additional work that might be considered is learning about the particular systems that the students will have access to at your institution. This is still just a general overview, as specific interfaces are considered in Chapter 3.

Exercises

1.1 In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems.

a. What are two such problems?

b. Can we ensure the same degree of security in a time-shared machine as in a dedicated machine? Explain your answer.

Answer:

a. Stealing or copying one's programs or data; using system resources (CPU, memory, disk space, peripherals) without proper accounting.

b. Probably not, since any protection scheme devised by humans can inevitably be broken by a human, and the more complex the scheme, the more difficult it is to feel confident of its correct implementation.

1.2 The issue of resource utilization shows up in different forms in different types of operating systems. List what resources must be managed carefully in the following settings:

a. Mainframe or minicomputer systems

b. Workstations connected to servers

c. Handheld computers

Answer:

a. Mainframes: memory and CPU resources, storage, network bandwidth

b. Workstations: memory and CPU resources

c. Handheld computers: power consumption, memory resources

1.3 Under what circumstances would a user be better off using a time-sharing system rather than a PC or single-user workstation?
Answer: When there are few other users, the task is large, and the hardware is fast, time-sharing makes sense. The full power of the system can be brought to bear on the user's problem. The problem can be solved faster than on a personal computer. Another case occurs when lots of other users need resources at the same time.
A personal computer is best when the job is small enough to be executed reasonably on it and when performance is sufficient to execute the program to the user's satisfaction.

1.4 Which of the functionalities listed below need to be supported by the operating system for the following two settings: (a) handheld devices and (b) real-time systems?

a. Batch programming

b. Virtual memory

c. Time sharing

Answer: For real-time systems, the operating system needs to support virtual memory and time sharing in a fair manner. For handheld systems, the operating system needs to provide virtual memory, but does not need to provide time-sharing. Batch programming is not necessary in both settings.

1.5 Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?
Answer: Symmetric multiprocessing treats all processors as equals, and I/O can be processed on any CPU. Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes tasks among the slaves, and I/O is usually done by the master only. Multiprocessors can save money by not duplicating power supplies, housings, and peripherals. They can execute programs more quickly and can have increased reliability. They are also more complex in both hardware and software than uniprocessor systems.

1.6 How do clustered systems differ from multiprocessor systems? What is required for two machines belonging to a cluster to cooperate to provide a highly available service?

Answer: Clustered systems are typically constructed by combining multiple computers into a single system to perform a computational task distributed across the cluster. Multiprocessor systems on the other hand could be a single physical entity comprising of multiple CPUs. A clustered system is less tightly coupled than a multiprocessor system. Clustered systems communicate using messages, while processors in a multiprocessor system could communicate using shared memory.
In order for two machines to provide a highly available service, the state on the two machines should be replicated and should be consistently updated. When one of the machines fails, the other could then takeover the functionality of the failed machine.

1.7 Distinguish between the client–server and peer-to-peer models of distributed systems.
Answer: The client-server model firmly distinguishes the roles of the client and server. Under this model, the client requests services that are provided by the server. The peer-to-peer model doesn't have such strict roles. In fact, all nodes in the system are considered peers and thus may act as *either* clients or servers—or both. A node may request a service from another peer, or the node may in fact provide such a service to other peers in the system.
For example, let's consider a system of nodes that share cooking recipes. Under the client-server model, all recipes are stored with the server. If a client wishes to access a recipe, it must request the recipe from the specified server. Using the peer-to-peer model, a peer node could ask other peer nodes for the specified recipe. The node (or perhaps nodes) with the requested recipe could provide it to the requesting node. Notice how each peer may act as both a client (it may request recipes) and as a server (it may provide recipes).

1.8 Consider a computing cluster consisting of two nodes running a database. Describe two ways in which the cluster software can manage access to the data on the disk. Discuss the benefits and disadvantages of each.
Answer: Consider the following two alternatives: **asymmetric clustering** and **parallel clustering**. With asymmetric clustering, one host runs the database application with the other host simply monitoring it. If the server fails, the monitoring host becomes the active server. This is appropriate for providing redundancy. However, it does not utilize the potential processing power of both hosts. With parallel clustering, the database application can run in parallel on both hosts. The difficulty in implementing parallel clusters is providing some form of distributed locking mechanism for files on the shared disk.

1.9 How are network computers different from traditional personal computers? Describe some usage scenarios in which it is advantageous to use network computers.
Answer: A network computer relies on a centralized computer for most of its services. It can therefore have a minimal operating system to manage its resources. A personal computer on the other hand has to be capable of providing all of the required functionality in a stand-alone manner without relying on a centralized manner. Scenarios where

administrative costs are high and where sharing leads to more efficient use of resources are precisely those settings where network computers are preferred.

1.10 What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?
Answer: An interrupt is a hardware-generated change of flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction. A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.

1.11 Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.

a. How does the CPU interface with the device to coordinate the transfer?

b. How does the CPU know when the memory operations are complete?

c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of the user programs? If so, describe what forms of interference are caused.

Answer: The CPU can initiate a DMA operation by writing values into special registers that can be independently accessed by the device. The device initiates the corresponding operation once it receives a command from the CPU. When the device is finished with its operation, it interrupts the CPU to indicate the completion of the operation.
Both the device and the CPU can be accessing memory simultaneously. The memory controller provides access to the memory bus in a fair manner to these two entities. A CPU might therefore be unable to issue memory operations at peak speeds since it has to compete with the device in order to obtain access to the memory bus.

1.12 Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems? Give arguments both that it is and that it is not possible.
Answer: An operating system for a machine of this type would need to remain in control (or monitor mode) at all times. This could be accomplished by two methods:

a. Software interpretation of all user programs (like some BASIC, Java, and LISP systems, for example). The software interpreter would provide, in software, what the hardware does not provide.

b. Require that all programs be written in high-level languages so that all object code is compiler-produced. The compiler would

generate (either in-line or by function calls) the protection checks that the hardware is missing.

1.13 Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

Answer: Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device. The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated. This is especially a problem on multiprocessor systems where more than one process may be accessing a datum. A component may be eliminated by an equal-sized cache, but only if: (a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.

1.14 Discuss, with examples, how the problem of maintaining coherence of cached data manifests itself in the following processing environments:

a. Single-processor systems

b. Multiprocessor systems

c. Distributed systems

Answer: In single-processor systems, the memory needs to be updated when a processor issues updates to cached values. These updates can be performed immediately or in a lazy manner. In a multiprocessor system, different processors might be caching the same memory location in its local caches. When updates are made, the other cached locations need to be invalidated or updated. In distributed systems, consistency of cached memory values is not an issue. However, consistency problems might arise when a client caches file data.

1.15 Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs.

Answer: The processor could keep track of what locations are associated with each process and limit access to locations that are outside of a program's extent. Information regarding the extent of a program's memory could be maintained by using base and limits registers and by performing a check for every memory access.

1.16 What network configuration would best suit the following environments?

a. A dormitory floor

b. A university campus

c. A state

d. A nation

Answer:

a. **A dormitory floor**—A LAN.

b. **A university campus**—A LAN, possibly a WAN for very large campuses.

c. **A state**—A WAN.

d. **A nation**—A WAN.

1.17 Define the essential properties of the following types of operating systems:

a. Batch

b. Interactive

c. Time sharing

d. Real time

e. Network

f. SMP

g. Distributed

h. Clustered

i. Handheld

Answer:

a. **Batch**. Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering, off-line operation, spooling, and multiprogramming. Batch is good for executing large jobs that need little interaction; they can be submitted and picked up later.

b. **Interactive**. This system is composed of many short transactions where the results of the next transaction may be unpredictable. Response time needs to be short (seconds) since the user submits and waits for the result.

c. **Time sharing**. This system uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal, and output is normally printed immediately to the screen.

d. **Real time**. Often used in a dedicated application, this system reads information from sensors and must respond within a fixed amount of time to ensure correct performance.

e. **Network**. Provides operating system features across a network such as file sharing.

f. **SMP**. Used in systems where there are multiple CPUs each running the same copy of the operating system. Communication takes place across the system bus.

g. **Distributed**.This system distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines, such as a high-speed bus or local area network.

h. **Clustered**. A clustered system combines multiple computers into a single system to perform computational tasks distributed across the cluster.

i. **Handheld**. A small computer system that performs simple tasks such as calendars, email, and web browsing. Handheld systems differ from traditional desktop systems with smaller memory and display screens and slower processors.

1.18 What are the tradeoffs inherent in handheld computers?

Answer: Handheld computers are much smaller than traditional desktop PCs. This results in smaller memory, smaller screens, and slower processing capabilities than a standard desktop PC. Because of these limitations, most handhelds currently can perform only basic tasks such as calendars, email, and simple word processing. However, due to their small size, they are quite portable and, when they are equipped with wireless access, can provide remote access to electronic mail and the world wide web.

Operating System Structures

Chapter 3 is concerned with the operating-system interfaces that users (or at least programmers) actually see: system calls. The treatment is somewhat vague since more detail requires picking a specific system to discuss. This chapter is best supplemented with exactly this detail for the specific system the students have at hand. Ideally they should study the system calls and write some programs making system calls. This chapter also ties together several important concepts including layered design, virtual machines, Java and the Java virtual machine, system design and implementation, system generation, and the policy/mechanism difference.

Exercises

2.1 The services and functions provided by an operating system can be divided into two main categories. Briefly describe the two categories and discuss how they differ.

Answer: One class of services provided by an operating system is to enforce protection between different processes running concurrently in the system. Processes are allowed to access only those memory locations that are associated with their address spaces. Also, processes are not allowed to corrupt files associated with other users. A process is also not allowed to access devices directly without operating system intervention. The second class of services provided by an operating system is to provide new functionality that is not supported directly by the underlying hardware. Virtual memory and file systems are two such examples of new services provided by an operating system.

2.2 List five services provided by an operating system that are designed to make it more convenient for users to use the computer system. In what cases it would be impossible for user-level programs to provide these services? Explain.

Answer:

a. **Program execution**. The operating system loads the contents (or sections) of a file into memory and begins its execution. A user-level program could not be trusted to properly allocate CPU time.

b. **I/O operations**. Disks, tapes, serial lines, and other devices must be communicated with at a very low level. The user need only specify the device and the operation to perform on it, while the system converts that request into device- or controller-specific commands. User-level programs cannot be trusted to access only devices they should have access to and to access them only when they are otherwise unused.

c. **File-system manipulation**. There are many details in file creation, deletion, allocation, and naming that users should not have to perform. Blocks of disk space are used by files and must be tracked. Deleting a file requires removing the name file information and freeing the allocated blocks. Protections must also be checked to assure proper file access. User programs could neither ensure adherence to protection methods nor be trusted to allocate only free blocks and deallocate blocks on file deletion.

d. **Communications**. Message passing between systems requires that messages be turned into packets of information, sent to the network controller, transmitted across a communications medium, and reassembled by the destination system. Packet ordering and data correction must take place. Again, user programs might not coordinate access to the network device, or they might receive packets destined for other processes.

e. **Error detection**. Error detection occurs at both the hardware and software levels. At the hardware level, all data transfers must be inspected to ensure that data have not been corrupted in transit. All data on media must be checked to be sure they have not changed since they were written to the media. At the software level, media must be checked for data consistency; for instance, do the number of allocated and unallocated blocks of storage match the total number of blocks on the device. There, errors are frequently process-independent (for instance, the corruption of data on a disk), so there must be a global program (the operating system) that handles all types of errors. Also, by having errors processed by the operating system, processes need not contain code to catch and correct all the errors possible on a system.

2.3 Describe three general methods for passing parameters to the operating system.
Answer:

a. Pass parameters in registers

b. Registers pass starting addresses of blocks of parameters

c. Parameters can be placed, or *pushed*, onto the *stack* by the program, and *popped* off the stack by the operating system

2.4 Describe how you could obtain a statistical profile of the amount of time spent by a program executing different sections of its code. Discuss the importance of obtaining such a statistical profile.

Answer: One could issue periodic timer interrupts and monitor what instructions or what sections of code are currently executing when the interrupts are delivered. A statistical profile of which pieces of code were active should be consistent with the time spent by the program in different sections of its code. Once such a statistical profile has been obtained, the programmer could optimize those sections of code that are consuming more of the CPU resources.

2.5 What are the five major activities of an operating system in regard to file management?
Answer:

- The creation and deletion of files
- The creation and deletion of directories
- The support of primitives for manipulating files and directories
- The mapping of files onto secondary storage
- The backup of files on stable (nonvolatile) storage media

2.6 What are the advantages and disadvantages of using the same system-call interface for manipulating both files and devices?
Answer: Each device can be accessed as though it was a file in the file system. Since most of the kernel deals with devices through this file interface, it is relatively easy to add a new device driver by implementing the hardware-specific code to support this abstract file interface. Therefore, this benefits the development of both user program code, which can be written to access devices and files in the same manner, and device-driver code, which can be written to support a well-defined API. The disadvantage with using the same interface is that it might be difficult to capture the functionality of certain devices within the context of the file access API, thereby resulting in either a loss of functionality or a loss of performance. Some of this could be overcome by the use of the ioctl operation that provides a general-purpose interface for processes to invoke operations on devices.

2.7 What is the purpose of the command interpreter? Why is it usually separate from the kernel? Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?
Answer: The command interpreter reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls. It is usually not part of the kernel since the command interpreter is subject to changes. An user should be able to develop a new command interpreter using the system-call interface provided by the operating system. The command interpreter allows an user to create and manage processes and also determine ways by which they communicate (such as through pipes and files). As all of this functionality could be accessed by an user-level program using the system calls, it should be possible for the user to develop a new command-line interpreter.

2.8 What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?
Answer: The two models of interprocess communication are (1) shared memory and (2) message passing, The fundamental difference between the two models has to do with performance. A shared memory segment is set up through a system call. However, once a shared memory segment is established between two — or more — processes, the processes can communicate via the shared memory segment without any intervention from the kernel. This leads to an efficient mechanism for interprocess communication. Message passing on the other hand typically involves a system call when the `send()` and `receive()` operations are invoked. As a result, because the kernel is directly involved during interprocess communication, it typically is less efficient than shared memory. However, message passing can be used as a mechanism for synchronizing the actions between communicating processes. That is, the `send()` and `receive()` statements can be used to coordinate the actions of two communicating processes. Shared memory on the other hand provides no such process synchronization mechanisms.

2.9 Why is the separation of mechanism and policy desirable?
Answer: Mechanism and policy must be separate to ensure that systems are easy to modify. No two system installations are the same, so each installation may want to tune the operating system to suit its needs. With mechanism and policy separate, the policy may be changed at will while the mechanism stays unchanged. This arrangement provides a more flexible system.

2.10 Why does Java provide the ability to call from a Java program native methods that are written in, say, C or C++? Provide an example of a situation in which a native method is useful.
Answer: Java programs are intended to be platform I/O independent. Therefore, the language does not provide access to most specific system resources such as reading from I/O devices or ports. To perform a system I/O specific operation, you must write it in a language that supports such features (such as C or C++). Keep in mind that a Java program that calls a native method written in another language will no longer be architecture-neutral.

2.11 It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.
Answer: The virtual memory subsystem and the storage subsystem are typically tightly coupled and requires careful design in a layered system due to the following interactions. Many systems allow files to be mapped into the virtual memory space of an executing process. On the other hand, the virtual memory subsystem typically uses the storage system to provide the backing store for pages that do not currently reside in memory. Also, updates to the file system are sometimes buffered in physical memory before it is flushed to disk, thereby requiring care-

ful coordination of the usage of memory between the virtual memory subsystem and the file system.

2.12 What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

Answer: Benefits typically include the following: (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, and (c) a simpler kernel design and functionality typically results in a more reliable operating system. User programs and system services interact in a microkernel architecture by using interprocess communication mechanisms such as messaging. These messages are conveyed by the operating system. The primary disadvantages of the microkernel architecture are the overheads associated with interprocess communication and the frequent use of the operating system's messaging functions in order to enable the user process and the system service to interact with each other.

2.13 In what ways is the modular kernel approach similar to the layered approach? In what ways does it differ from the layered approach?

Answer: The modular kernel approach requires subsystems to interact with each other through carefully constructed interfaces that are typically narrow (in terms of the functionality that is exposed to external modules). The layered kernel approach is similar in that respect. However, the layered kernel imposes a strict ordering of subsystems such that subsystems at the lower layers are not allowed to invoke operations corresponding to the upper-layer subsystems. There are no such restrictions in the modular-kernel approach, wherein modules are free to invoke each other without any constraints.

2.14 What is the main advantage for an operating-system designer of using a virtual-machine architecture? What is the main advantage for a user?

Answer: The system is easy to debug, and security problems are easy to solve. Virtual machines also provide a good platform for operating system research since many different operating systems can run on one physical system.

2.15 Why is a just-in-time compiler useful for executing Java programs?

Answer: Java is an interpreted language. This means that the JVM interprets the bytecode instructions one at a time. Typically, most interpreted environments are slower than running native binaries, for the interpretation process requires converting each instruction into native machine code. A just-in-time (JIT) compiler compiles the bytecode for a method into native machine code the first time the method is encountered. This means that the Java program is essentially running as a native application (of course, the conversion process of the JIT takes time as well, but not as much as bytecode interpretation). Furthermore, the JIT caches compiled code so that it can be reused the next time the method is encountered. A Java program that is run by a JIT rather than a traditional interpreter typically runs much faster.

2.16 What is the relationship between a guest operating system and a host operating system in a system like VMware? What factors need to be considered in choosing the host operating system?

Answer: A guest operating system provides its services by mapping them onto the functionality provided by the host operating system. A key issue that needs to be considered in choosing the host operating system is whether it is sufficiently general in terms of its system-call interface in order to be able to support the functionality associated with the guest operating system.

2.17 The experimental Synthesis operating system has an assembler incorporated within the kernel. To optimize system-call performance, the kernel assembles routines within kernel space to minimize the path that the system call must take through the kernel. This approach is the antithesis of the layered approach, in which the path through the kernel is extended to make building the operating system easier. Discuss the pros and cons of the Synthesis approach to kernel design and to system-performance optimization.

Answer: Synthesis is impressive due to the performance it achieves through on-the-fly compilation. Unfortunately, it is difficult to debug problems within the kernel due to the fluidity of the code. Also, such compilation is system specific, making Synthesis difficult to port (a new compiler must be written for each architecture).

2.18 In Section 2.3, we described a program that copies the contents of one file to a destination file. This program works by first prompting the user for the name of the source and destination files. Write this as a C program using the POSIX API. Be sure to include all necessary error checking, including ensuring that the source file exists. Also be sure to close the input and output files when the program completes.

Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. For example, Linux systems provide the `strace` utility for tracing system calls. If the executable file is called `filecopy`, the `strace` command is invoked as follows:

```
strace ./filecopy
```

Next, answer the following questions (this will require carefully examining the output of system calls):

a. What system call or calls correspond to prompting the user for the name of the source and destination files?

b. What system call or calls correspond to opening the source and destination files?

c. What system call or calls correspond to reading the source file and writing the destination file?

d. What system call or calls correspond to closing the source and destination files?

Answer: The answers may vary, but expect mostly the `close()` system call.

2.19 Repeat Exercise 2.18, but this time write a Java program to copy the file. To trace the actual system calls that are invoked from a Java program—for example, `FileCopy.java`—using `strace`, enter the following:

```
strace java FileCopy
```

This will trace all system calls that are used to invoke the JVM as well as the Java program `Filecopy.java`.

Are there any differences in the system calls for performing I/O between the C program for Exercise 2.18 and the Java program for this exercise? Explain.

Answer: The answers may vary widely for this problem. Many students may comment that the strace output for the Java program produces alot more system calls—mostly due to the JVM. However, they should notice that the actual system calls for the file I/O is actually quite similar between the two programs.

Processes

In this chapter we introduce the concepts of a process and concurrent execution; These concepts are at the very heart of modern operating systems. A process is a program in execution and is the unit of work in a modern time-sharing system. Such a system consists of a collection of processes: Operating-system processes executing system code and user processes executing user code. All these processes can potentially execute concurrently, with the CPU (or CPUs) multiplexed among them. By switching the CPU between processes, the operating system can make the computer more productive. We also introduce the notion of a thread (lightweight process) and interprocess communication (IPC). Threads are discussed in more detail in Chapter 4.

Exercises

3.1 Describe the differences among short-term, medium-term, and long-term scheduling.
Answer:

a. **Short-term** (CPU scheduler)—selects from jobs in memory those jobs that are ready to execute and allocates the CPU to them.

b. **Medium-term**—used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.

c. **Long-term** (job scheduler)—determines which jobs are brought into memory for processing.

The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

3.2 Describe the actions taken by a kernel to context-switch between processes.

Answer: In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

3.3 Consider the RPC mechanism. Describe the undesirable circumstances that could arise from not enforcing either the "at most once" or "exactly once" semantics. Describe possible uses for a mechanism that had neither of these guarantees.

Answer: If an RPC mechanism cannot support either the "at most once" or "at least once" semantics, then the RPC server cannot guarantee that a remote procedure will not be invoked multiple occurrences. Consider if a remote procedure were withdrawing money from a bank account on a system that did not support these semantics. It is possible that a single invocation of the remote procedure might lead to multiple withdrawals on the server.

For a system to support either of these semantics generally requires the server maintain some form of client state such as the timestamp described in the text.

If a system were unable to support either of these sematics, then such a system could only safely provide remote procedures that do not alter data or provide time-sensitive results. Using our bank account as an example, we certainly require "at most once" or "at least once" semantics for performing a withdrawal (or deposit!). However, an inquiry into an account balance or other accunt information such as name, address, etc. does not require these semantics.

3.4 Using the program shown in Figure 3.24, explain what will be output at `Line A`.

Answer: When control returns to the parent, its value remains at 5 as the child updates its copy of the value.

3.5 What are the benefits and detriments of each of the following? Consider both the systems and the programmers' levels.

a. Symmetric and asymmetric communication

b. Automatic and explicit buffering

c. Send by copy and send by reference

d. Fixed-sized and variable-sized messages

Answer:

a. **Symmetric and asymmetric communication**—A benefit of symmetric communication is that it allows a rendezvous between the sender and receiver. A disadvantage of a blocking send is that a rendezvous may not be required and the message could be delivered asynchronously. As a result, message-passing systems often provide both forms of synchronization.

b. **Automatic and explicit buffering**—Automatic buffering provides a queue with indefinite length, thus ensuring the sender will never have to block while waiting to copy a message. There are no specifications on how automatic buffering will be provided; one scheme may reserve sufficiently large memory where much of the memory is wasted. Explicit buffering specifies how large the buffer is. In this situation, the sender may be blocked while waiting for available space in the queue. However, it is less likely that memory will be wasted with explicit buffering.

c. **Send by copy and send by reference**—Send by copy does not allow the receiver to alter the state of the parameter; send by reference does allow it. A benefit of send by reference is that it allows the programmer to write a distributed version of a centralized application. Java's RMI provides both; however, passing a parameter by reference requires declaring the parameter as a remote object as well.

d. **Fixed-sized and variable-sized messages**—The implications of this are mostly related to buffering issues; with fixed-size messages, a buffer with a specific size can hold a known number of messages. The number of variable-sized messages that can be held by such a buffer is unknown. Consider how Windows 2000 handles this situation: with fixed-sized messages (anything < 256 bytes), the messages are copied from the address space of the sender to the address space of the receiving process. Larger messages (i.e. variable-sized messages) use shared memory to pass the message.

3.6 The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ... Formally, it can be expressed as:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

Write a C program using the `fork()` system call that that generates the Fibonacci sequence in the child process. The number of the sequence will be provided in the command line. For example, if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child process. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the `wait()` call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a non-negative number is passed on the command line.
Answer: Please refer to the supporting Web site for source code solution.

3.7 Repeat the preceding exercise, this time using the `CreateProcess()` in the Win32 API. In this instance, you will need to specify a separate program to be invoked from `CreateProcess()`. It is this separate program that will run as a child process outputting the Fibonacci sequence. Per-

form necessary error checking to ensure that a non-negative number is passed on the command line.
Answer: Please refer to the supporting Web site for source code solution.

3.8 Modify the Date server shown in Figure 3.19 so that it delivers random one-line fortunes rather than the current date. Allow the fortunes to contain multiple lines. The date client shown in Figure 3.20 can be used to read the multi-line fortunes returned by the fortune server.
Answer: Please refer to the supporting Web site for source code solution.

3.9 An **echo server** is a server that echoes back whatever it receives from a client. For example, if a client sends the server the string *Hello there!* the server will respond with the exact data it received from the client—that is, *Hello there!*

Write an echo server using the Java networking API described in Section 3.6.1. This server will wait for a client connection using the `accept()` method. When a client connection is received, the server will loop, performing the following steps:

- Read data from the socket into a buffer.
- Write the contents of the buffer back to the client.

The server will break out of the loop only when it has determined that the client has closed the connection.

The date server shown in Figure 3.19 uses the `java.io.BufferedReader` class. `BufferedReader` extends the `java.io.Reader` class, which is used for reading character streams. However, the echo server cannot guarantee that it will read characters from clients; it may receive binary data as well. The class `java.io.InputStream` deals with data at the byte level rather than the character level. Thus, this echo server must use an object that extends `java.io.InputStream`. The `read()` method in the `java.io.InputStream` class returns -1 when the client has closed its end of the socket connection.
Answer: Please refer to the supporting Web site for source code solution.

3.10 Write an RMI application in which the server delivers random one-line fortunes. The interface for the remote object appears as

```
import java.rmi.*;

public interface RemoteFortune extends Remote
{
   public abstract String getFortune()
      throws RemoteException;
}
```

A client invoking the `getFortune()` method will receive a random one-line fortune from the remote object.
Answer: Please refer to the supporting Web site for source code solution.

Threads

The process model introduced in Chapter 3 assumed that a process was an executing program with a single thread of control. Many modern operating systems now provide features for a process to contain multiple threads of control. This chapter introduces many concepts associated with multithreaded computer systems and covers how to use Java to create and manipulate threads. We have found it especially useful to discuss how a Java thread maps to the thread model of the host operating system.

Exercises

4.1 Provide two programming examples in which multithreading does *not* provide better performance than a single-threaded solution

Answer: (1) Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

4.2 Describe the actions taken by a thread library to context switch between user-level threads.

Answer: Context switching between user threads is quite similar to switching between kernel threads, although it is dependent on the threads library and how it maps user threads to kernel threads. In general, context switching between user threads involves taking a user thread of its LWP and replacing it with another thread. This act typically involves saving and restoring the state of the registers.

4.3 Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

Answer: When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand, will not be capable of

performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multithreaded solution would perform better even on a single-processor system.

4.4 Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values

b. Heap memory

c. Global variables

d. Stack memory

Answer: The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

4.5 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system?
Answer: A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

4.6 The Java API provides several different thread-pool architectures:

a. `newFixedThreadPool(int)`

b. `newCachedThreadPool()`

c. `newSingleThreadExecutor()`

Discuss the merits of each.
Answer: No Answer

4.7 As described in Section 4.5.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the `clone()` system call. However, many operating systems—such as Windows XP and Solaris—treat processes and threads differently. Typically, such systems use a notation wherein the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.
Answer: On one hand, in systems where processes and threads are considered as similar entities, some of the operating system code could be simplified. A scheduler, for instance, can consider the different processes and threads on an equal footing without requiring special code to examine the threads associated with a process during every scheduling step. On the other hand, this uniformity could make it harder to impose

process-wide resource constraints in a direct manner. Instead, some extra complexity is required to identify which threads correspond to which process and perform the relevant accounting tasks.

4.8 The program shown in Figure 4.11 uses the Pthreads API. What would be output from the program at `LINE C` and `LINE P`?
Answer: Output at `LINE C` is 5. Output at `LINE P` is 0.

4.9 Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processors in the system. Discuss the performance implications of the following scenarios.

a. The number of kernel threads allocated to the program is less than the number of processors.

b. The number of kernel threads allocated to the program is equal to the number of processors.

c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.

Answer: When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors. When the number of kernel threads is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when a kernel-thread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle. When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.

4.10 Write a multithreaded Java, Pthreads, or Win32 program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.
Answer: Please refer to the supporting Web site for source code solution.

4.11 The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ... Formally, it can be expressed as:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

Write a multithreaded program that generates the Fibonacci series using either the Java, Pthreads, or Win32 thread library. This program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program will

then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that is shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish using the techniques described in Section 4.3.
Answer: Please refer to the supporting Web site for source code solution.

4.12 Modify the socket-based date server (Figure 3.19) in Chapter 3 so that the server services each client request in a separate thread.
Answer: Please refer to the supporting Web site for source code solution.

4.13 Exercise 3.9 in Chapter 3 specifies designing an echo server using the Java threading API. However, this server is single-threaded, meaning the server cannot respond to concurrent echo clients until the current client exits. Modify the solution to Exercise 3.9 so that the echo server services each client in a separate request
Answer: Please refer to the supporting Web site for source code solution.

4.14 In Exercise 4.13, you are asked to service each client in a separate thread. Modify your solution so that each client is serviced using a thread pool, as discussed in Section 4.5.4. Experiment with the three different models o f thread-pool architecture presented in that section.
Answer: Please refer to the supporting Web site for source code solution.

CPU Scheduling

CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. In this chapter, we introduce the basic scheduling concepts and discuss in great length CPU scheduling. FCFS, SJF, Round-Robin, Priority, and the other scheduling algorithms should be familiar to the students. This is their first exposure to the idea of resource allocation and scheduling, so it is important that they understand how it is done. Gantt charts, simulations, and play acting are valuable ways to get the ideas across. Show how the ideas are used in other situations (like waiting in line at a post office, a waiter time sharing between customers, even classes being an interleaved round-robin scheduling of professors).

A simple project is to write several different CPU schedulers and compare their performance by simulation. The source of CPU and I/O bursts may be generated by random number generators or by a trace tape. The instructor can make up the trace tape in advance to provide the same data for all students. The file that I used was a set of jobs, each job being a variable number of alternating CPU and I/O bursts. The first line of a job was the word JOB and the job number. An alternating sequence of CPU n and I/O n lines followed, each specifying a burst time. The job was terminated by an END line with the job number again. Compare the time to process a set of jobs using FCFS, Shortest-Burst-Time, and round-robin scheduling. Round-robin is more difficult, since it requires putting unfinished requests back in the ready queue.

Exercises

5.1 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

Answer: I/O-bound programs have the property of performing only a small amount of computation before performing I/O. Such programs typically do not use up their entire CPU quantum. CPU-bound programs, on the other hand, use their entire quantum without performing any blocking I/O operations. Consequently, one could make better use of the

computer's resouces by giving higher priority to I/O-bound programs and allow them to execute ahead of the CPU-bound programs.

5.2 Discuss how the following pairs of scheduling criteria conflict in certain settings.

a. CPU utilization and response time

b. Average turnaround time and maximum waiting time

c. I/O device utilization and CPU utilization

Answer:

a. CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could, however, result in increasing the response time for processes.

b. Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy could, however, starve long-running tasks and thereby increase their waiting time.

c. I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

5.3 Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

a. $\alpha = 0$ and $\tau_0 = 100$ milliseconds

b. $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds

Answer: When $\alpha = 0$ and $\tau_0 = 100$ milliseconds, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds, the most recent behavior of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memoryless, and simply predicts the length of the previous burst for the next quantum of CPU execution.

5.4 Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

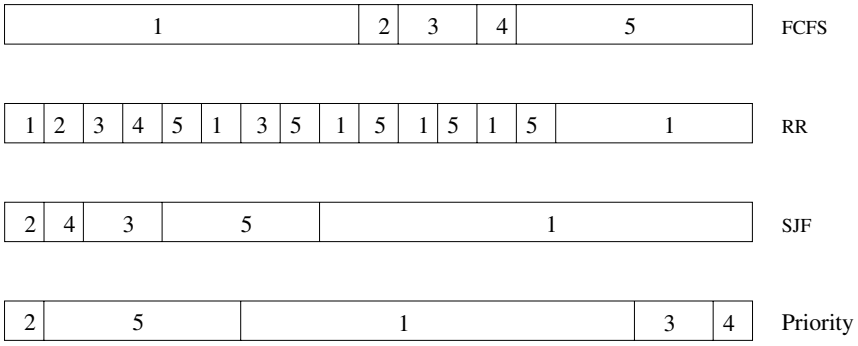
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

Answer:

a. The four Gantt charts are



b. Turnaround time

	FCFS	RR	SJF	Priority
P_1	10	19	19	16
P_2	11	2	1	1
P_3	13	7	4	18
P_4	14	4	2	19
P_5	19	14	9	6

c. Waiting time (turnaround time minus burst time)

	FCFS	RR	SJF	Priority
P_1	0	9	9	6
P_2	10	1	0	0
P_3	11	5	2	16
P_4	13	3	1	18
P_5	14	9	4	1

d. Shortest Job First

5.5 Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served

b. Shortest job first

c. Round robin

d. Priority

Answer: Shortest job first and priority-based scheduling algorithms could result in starvation.

5.6 Consider a variant of the RR scheduling algorithm where the entries in the ready queue are pointers to the PCBs.

a. What would be the effect of putting two pointers to the same process in the ready queue?

b. What would be the major advantages and disadvantages of this scheme?

c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

Answer:

a. In effect, that process will have increased its priority since by getting time more often it is receiving preferential treatment.

b. The advantage is that more important jobs could be given more time, in other words, higher priority in treatment. The consequence, of course, is that shorter jobs will suffer.

c. Allot a longer amount of time to processes deserving higher priority. In other words, have two or more quantums possible in the Round-Robin scheme.

5.7 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

Answer:

a. The time quantum is 1 millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of 1/1.1 * 100 = 91%.

b. The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore 10*1.1 + 10.1 (as each I/O-bound task executes for 1 millisecond and then incur the context switch task, whereas the

CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = 94%.

5.8 Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

Answer: The program could maximize the CPU time allocated to it by not fully utilizing its time quantums. It could use a large fraction of its assigned quantum, but relinquish the CPU before the end of the quantum, thereby increasing the priority associated with the process.

5.9 Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α; when it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

a. What is the algorithm that results from $\beta > \alpha > 0$?

b. What is the algorithm that results from $\alpha < \beta < 0$?

Answer:

a. FCFS

b. LIFO

5.10 Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes:

a. FCFS

b. RR

c. Multilevel feedback queues

Answer:

a. FCFS—discriminates against short jobs since any short jobs arriving after long jobs will have a longer waiting time.

b. RR—treats all jobs equally (giving them equal bursts of CPU time) so short jobs will be able to leave the system faster since they will finish first.

c. Multilevel feedback queues work similar to the RR algorithm—they discriminate favorably toward short jobs.

5.11 Using the Windows XP scheduling algorithm, what is the numeric priority of a thread for the following scenarios?

a. A thread in the REALTIME_PRIORITY_CLASS with a relative priority of HIGHEST.

b. A thread in the NORMAL_PRIORITY_CLASS with a relative priority of NORMAL.

c. A thread in the HIGH_PRIORITY_CLASS with a relative priority of ABOVE_NORMAL.

Answer:

a. 26

b. 8

c. 14

5.12 Consider the scheduling algorithm in the Solaris operating system for time-sharing threads:

a. What is the time quantum (in milliseconds) for a thread with priority 10? With priority 55?

b. Assume a thread with priority 35 has used its entire time quantum without blocking. What new priority will the scheduler assign this thread?

c. Assume a thread with priority 35 blocks for I/O before its time quantum has expired. What new priority will the scheduler assign this thread?

Answer:

a. 160 and 40

b. 35

c. 54

5.13 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (recent CPU usage / 2) + base

where base = 60 and *recent CPU usage* refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P_1 is 40, process P_2 is 18, and process P_3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Answer: The priorities assigned to the processes are 80, 69, and 65 respectively. The scheduler lowers the relative priority of CPU-bound processes.

5.14 Modify your solution to Exercise 4.13 so that each separate thread increases its priority by +2.

Answer:
This program should be modified so that each thread performs the following:

```
Thread.currentThread().setPriority(Thread.NORM_PRIORITY+2)
```

5.15 As discussed in Section 5.7, the specification for the JVM allows implementations to ignore calls to `setPriority()`. One argument in favor of ignoring `setPriority()` is that increasing—or decreasing—the priority of a thread has little effect once the thread begins running, because the operating-system scheduler takes over and modifies the priority of the kernel thread to which the Java thread is mapped based on how CPU- or I/O-intensive the thread is. Discuss the pros and cons of this argument.

Answer: An argument in favor of ignoring calls is that there are different values passed to the `setPriority()` method that may have no difference in how priorities are expressed on the native operating system. (i.e. Solaris doesn't distinguish between the values 1..4 or the values 5 .. 10. An argument in favor of supporting `setPriority()` is that it is part of the API and that programmers may want some of their threads to run in preference to others. Because it is part of the API, implementations of the JVM should implement it, even though it is possible that the programmer may specify priority values that are not distinctive on the native operating system.

CHAPTER 6

Process Synchronization

Chapter 6 is concerned with the topic of process synchronization among concurrently executing processes. Concurrency is generally very hard for students to deal with correctly, and so we have tried to introduce it and its problems with the classic process coordination problems: mutual exclusion, bounded-buffer, readers/writers, and so on. An understanding of these problems and their solutions is part of current operating-system theory and development.

We first use semaphores and monitors to introduce synchronization techniques. Next, Java synchronization is introduced to further demonstrate a language-based synchronization technique. We conclude with a discussion of how contemporary operating systems provide features for process synchronization and thread safety.

Exercises

6.1 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of process P_i (`i` == 0 or 1) is shown in Figure 6.25; the other process is P_j (`j` == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

Answer: This algorithm satisfies the three conditions of mutual exclusion. (1) Mutual exclusion is ensured through the use of the `flag` and `turn` variables. If both processes set their `flag` to `true`, only one will succeed, namely, the process whose turn it is. The waiting process can only enter its critical section when the other process updates the value of turn. (2) Progress is provided, again through the `flag` and `turn` variables. This algorithm does not provide strict alternation. Rather, if a process wishes to access their critical section, it can set their `flag` variable to `true` and enter their critical section. It sets `turn` to the value of the

other process only upon exiting its critical section. If this process wishes to enter its critical section again—before the other process—it repeats the process of entering its critical section and setting `turn` to the other process upon exiting. (3) Bounded waiting is preserved through the use of the TTturn variable. Assume two processes wish to enter their respective critical sections. They both set their value of `flag` to true; however, only the thread whose turn it is can proceed; the other thread waits. If bounded waiting were not preserved, it would therefore be possible that the waiting process would have to wait indefinitely while the first process repeatedly entered—and exited—its critical section. However, Dekker's algorithm has a process set the value of `turn` to the other process, thereby ensuring that the other process will enter its critical section next.

6.2 The first known correct software solution to the critical-section problem for n processes with a lower bound on waiting of $n - 1$ turns was presented by Eisenberg and McGuire. The processes share the following variables:

```
enum pstate {idle, want_in, in_cs};
pstate flag[n];
int turn;
```

All the elements of `flag` are initially `idle`; the initial value of `turn` is immaterial (between 0 and `n-1`). The structure of process P_i is shown in Figure 6.26. Prove that the algorithm satisfies all three requirements for the critical-section problem.

Answer: This algorithm satisfies the three conditions. Before we show that the three conditions are satisfied, we give a brief explanation of what the algorithm does to ensure mutual exclusion. When a process i requires access to critical section, it first sets its `flag` variable to want_in to indicate its desire. It then performs the following steps: (1) It ensures that all processes whose index lies between `turn` and i are idle. (2) If so, it updates its `flag` to in_cs and checks whether there is already some other process that has updated its `flag` to in_cs. (3) If not and if it is this process's turn to enter the critical section or if the process indicated by the `turn` variable is idle, it enters the critical section. Given the above description, we can reason about how the algorithm satisfies the requirements in the following manner:

a. Mutual exclusion is ensured: Notice that a process enters the critical section only if the following requirements is satisfied: no other process has its `flag` variable set to in_cs. Since the process sets its own `flag` variable set to in_cs before checking the status of other processes, we are guaranteed that no two processes will enter the critical section simultaneously.

b. Progress requirement is satisfied: Consider the situation where multiple processes simultaneously set their `flag` variables to in_cs and then check whether there is any other process has the `flag` variable set to in_cs. When this happens, all processes realize that there are competing processes, enter the next iteration of the outer

while(1) loop and reset their `flag` variables to want_in. Now the only process that will set its `turn` variable to in_cs is the process whose index is closest to `turn`. It is however possible that new processes whose index values are even closer to `turn` might decide to enter the critical section at this point and therefore might be able to simultaneously set its `flag` to in_cs. These processes would then realize there are competing processes and might restart the process of entering the critical section. However, at each iteration, the index values of processes that set their `flag` variables to in_cs become closer to `turn` and eventually we reach the following condition: only one process (say `k`) sets its `flag` to in_cs and no other process whose index lies between `turn` and k has set its `flag` to in_cs. This process then gets to enter the critical section.

c. Bounded-waiting requirement is met: The bounded waiting requirement is satisfied by the fact that when a process k desires to enter the critical section, its `flag` is no longer set to idle. Therefore, any process whose index does not lie between `turn` and k cannot enter the critical section. In the meantime, all processes whose index falls between `turn` and k and desire to enter the critical section would indeed enter the critical section (due to the fact that the system always makes progress) and the `turn` value monotonically becomes closer to k. Eventually, either `turn` becomes k or there are no processes whose index values lie between `turn` and k, and therefore process k gets to enter the critical section.

6.3 What is the meaning of the term *busy waiting*? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

Answer: *Busy waiting* means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

6.4 Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

Answer: Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock can be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock.

6.5 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

Answer: If a user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes execute.

6.6 Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.
Answer: Interrupts are not sufficient in multiprocessor systems since disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot guarantee mutually exclusive access to program state.

6.7 Describe how the `Swap()` instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement.
Answer:

```
do { waiting[i] = TRUE; key = TRUE;
    while (waiting[i] && key) key = Swap(&lock, &key);

    waiting[i] = FALSE;

    /* critical section */

    j = (i+1) % n; while ((j != i) && !waiting[j])
    j = (j+1) % n;
    if (j == i) lock = FALSE; else waiting[j] = FALSE;

    n/* remainder section */
    while (TRUE);
}
```

6.8 Demonstrate that monitors and semaphores are equivalent insofar as they can be used to implement the same types of synchronization problems.
Answer: A semaphore can be implemented using the following monitor code:

```
monitor semaphore {
    int value = 0;
    condition c;

    semaphore_increment() {
       value++;
       c.signal();
    }

    semaphore_decrement() {
       while (value == 0)
       c.wait();
       value--;
    }
}
```

A monitor could be implemented using a semaphore in the following manner. Each condition variable is represented by a queue of threads waiting for the condition. Each thread has a semaphore associated with its queue entry. When a thread performs a wait operation, it creates a new semaphore (initialized to zero), appends the semaphore to the queue associated with the condition variable, and performs a blocking semaphore decrement operation on the newly created semaphore. When a thread performs a signal on a condition variable, the first process in the queue is awakened by performing an increment on the corresponding semaphore.

6.9 Discuss the tradeoff between fairness and throughput of operations in the readers-writers problem. Propose a method for solving the readers-writers problem without causing starvation.

Answer: Throughput in the readers-writers problem is increased by favoring multiple readers as opposed to allowing a single writer to exclusively access the shared values. On the other hand, favoring readers could result in starvation for writers. The starvation in the readers-writers problem could be avoided by keeping timestamps associated with waiting processes. When a writer is finished with its task, it would wake up the process that has been waiting for the longest duration. When a reader arrives and notices that another reader is accessing the database, then it would enter the critical section only if there are no waiting writers. These restrictions would guarantee fairness.

6.10 How does the `signal()` operation associated with monitors differ from the corresponding operation defined for semaphores?

Answer: The `signal()` operation associated with monitors is not persistent in the following sense: if a signal is performed and if there are no waiting threads, then the signal is simply ignored and the system does not remember that the signal took place. If a subsequent wait operation is performed, then the corresponding thread simply blocks. In semaphores, on the other hand, every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. A

future wait operation would immediately succeed because of the earlier increment.

6.11 Suppose the `signal()` statement can appear only as the last statement in a monitor procedure. Suggest how the implementation described in Section 6.7 can be simplified.
Answer: If the signal operation were the last statement, then the lock could be transferred from the signalling process to the process that is the recipient of the signal. Otherwise, the signalling process would have to explicitly release the lock and the recipient of the signal would have to compete with all other processes to obtain the lock to make progress.

6.12 Show that, if the `acquire()` and `release()` semaphore operations are not executed atomically, then mutual exclusion may be violated.
Answer: A acquire operation atomically decrements the value associated with a semaphore. If two acquire operations are executed on a semaphore when its value is 1, if the two operations are not performed atomically, then it is possible that both operations might proceed to decrement the semaphore value, thereby violating mutual exclusion.

6.13 Show how to implement the `acquire()` and `release()` semaphore operations in multiprocessor environments using the `TestAndSet()` instruction. The solution should exhibit minimal busy waiting.
Answer: Here is the pseudocode for implementing the operations:

```
int guard = 0;
int semaphore_value = 0;

acquire()
{
    while (TestAndSet(&guard) == 1);
    if (semaphore_value == 0) {
       atomically add process to a queue of processes
       waiting for the semaphore and set guard to 0;
    } else {
       semaphore_value--;
       guard = 0;
    }
}


release()
{
    while (TestAndSet(&guard) == 1);
    if (semaphore_value == 0 &&
            there is a process on the wait queue)
        wake up the first process in the queue
        of waiting processes
    else
       semaphore_value++;
    guard = 0;
}
```

6.14 The `wait()` statement in all Java program examples in this chapter is part of a `while` loop. Explain why you would always need to use a `while` statement when using `wait()` and why you would never use an `if` statement.

Answer: A thread may receive the notification, but the condition it is waiting for remains unchanged. (The `doWork()` method is a good example of this.) The call to `wait()` must always be placed in a `while` loop so it can re-check the condition it is waiting for when it receives a notification. (Note, regular Java synchronization does not provide a mechanism for notifying a thread a particular condition has been met. However, if condition variables are used, it is possible to notify a thread via a condition variable.)

6.15 Why do Solaris, Linux, and Windows 2000 use spinlocks as a synchronization mechanism only on multiprocessor systems and not on single-processor systems?

Answer: Solaris, Linux, and Windows 2000 use spinlocks as a synchronization mechanism only on multiprocessor systems. Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock could be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock.

6.16 In log-based systems that provide support for transactions, updates to data items cannot be performed before the corresponding entries are logged. Why is this restriction necessary?

Answer: If the transaction needs to be aborted, then the values of the updated data values need to be rolled back to the old values. This requires the old values of the data entries to be logged before the updates are performed.

6.17 Show that the two-phase locking protocol ensures conflict serializability.

Answer: A schedule refers to the execution sequence of the operations for one or more transactions. A serial schedule is the situation where each transaction of a schedule is performed atomically. If a schedule consists of two different transactions where consecutive operations from the different transactions access the same data and at least one of the operations is a write, then we have what is known as a *conflict*. If a schedule can be transformed into a serial schedule by a series of swaps on nonconflicting operations, we say that such a schedule is conflict serializable.

The two-phase locking protocol ensures conflict serializabilty because exclusive locks (which are used for write operations) must be acquired serially, without releasing any locks during the acquire (growing) phase. Other transactions that wish to acquire the same locks must wait for the first transaction to begin releasing locks. By requiring that all locks

must first be acquired before releasing any locks, we are ensuring that potential conflicts are avoided.

6.18 What are the implications of assigning a new timestamp to a transaction that is rolled back? How does the system process transactions that were issued after the rolled-back transaction but that have timestamps smaller than the new timestamp of the rolled-back transaction?
Answer: If the transactions that were issued after the rolled-back transaction had accessed variables that were updated by the rolled-back transaction, then these transactions would have to be rolled back as well. If they have not performed such operations (that is, there is no overlap with the rolled-back transaction in terms of the variables accessed), then these operations are free to commit when appropriate.

6.19 Suppose we replace the `wait()` and `signal()` operations of monitors with a single construct `await(B)`, where `B` is a general Boolean expression that causes the process executing it to wait until `B` becomes `true`.

a. Write a monitor using this scheme to implement the readers–writers problem.

b. Explain why, in general, this construct cannot be implemented efficiently.

c. What restrictions need to be put on the `await` statement so that it can be implemented efficiently? (Hint: Restrict the generality of `B`; see Kessels [1977].)

Answer:

a. The readers–writers problem could be modified with the following more generate `await` statements: A reader can perform "await(active_writers == 0 && waiting_writers == 0)" to check that there are no active writers and there are no waiting writers before it enters the critical section. The writer can perform a "await(active_writers == 0 && active_readers == 0)" check to ensure mutually exclusive access.

b. The system would have to check which one of the waiting threads have to be awakened by checking which one of their waiting conditions are satisfied after a signal. This requires considerable complexity and might require some interaction with the compiler to evaluate the conditions at different points in time. One could restrict the Boolean condition to be a disjunction of conjunctions with each component being a simple check (equality or inequality with respect to a static value) on a program variable. In that case, the Boolean condition could be communicated to the run-time system, which could perform the check every time it needs to determine which thread to be awakened.

6.20 The `HardWareData` class in Figure 6.4 abstracts the idea of the *get-and-set* and *swap* instructions. However, this class is not considered thread safe, because multiple threads may concurrently access its methods and thread safety requires that each method be performed atomically.

Rewrite the `HardwareData` class using Java synchronization so that it is thread safe.
Answer: Please refer to the supporting Web site for source code solution.

6.21 The **Singleton** design pattern ensures that only one instance of an object is created. For example, assume we have a class called `Singleton` and we only wish to allow one instance of it. Rather than creating a `Singleton` object using its constructor, we instead declare the constructor as `private` and provide a public static method—such as `getInstance()`—for object creation:

```
Singleton sole = Singleton.getInstance();
```

Figure 6.44 provides one strategy for implementing the Singleton pattern. The idea behind this approach is to use **lazy initialization**, whereby we create an instance of the object only when it is needed—that is, when `getInstance()` is first called. However, Figure 6.44 suffers from a race condition. Identify the race condition.

Figure 6.45 shows an alternative strategy that addresses the race condition by using the **double-checked locking idiom**. Using this strategy, we first check whether `instance` is `null`. If it is, we next obtain the lock for the `Singleton` class and then double-check whether `instance` is still `null` before creating the object. Does this strategy result in any race conditions? If so, identify and fix them. Otherwise, illustrate why this code example is thread safe.
Answer: Figure 6.44suffers from a race condition. Consider if thread 1 invokes the `getInstance()` method where `instance` is null. However, before a new instance can be created, thread 1 is preempted by thread 2. Thread 2 enters `getInstance()`and creates and returns the `Singleton` object. When control returns to thread 1, it resumes where it creates a second `Singleton` object. (Note that it is possible to declare `getInstance()` as `synchronized` to prevent the race condition. However, this is expensive as this method only requires thread safety for the first invocation — where the object is first created.)
The second strategy using double-checked locking works well in theory. However, because of the differences in Java memory models, there is no guarantee it works on all architectures. Consider the following scenario:

a. Thread 1 enters the `getInstance()` method.

b. Thread 1 enters the `synchronized(Singleton.class)` block as `instance` is null.

c. Thread 1 proceeds to the statement `instance = new Singleton()` and makes `instance` non-null, but before the constructor finishes execution.

d. Thread 1 is preempted by thread 2.

e. Thread 2 checks to see if `instance` is null. Because it is not, thread 2 returns the instance reference to a fully constructed — but partially initialized — `Singleton` object.

f. Thread 2 is preempted by thread 1.

g. Thread 1 now completes the initialization of the `Singleton` object by completing execution of its constructor and returning a reference to to the object.

6.22 Implement the `Channel` interface from Chapter 3 so that the `send()` and `receive()` methods are blocking. That is, a thread invoking `send()` will block if the channel is full. If the channel is empty, a thread invoking `receive()` will similarly block. To do this will require storing the messages in a fixed-length array. Ensure that your implementation is thread safe (using Java synchronization) and that the messages are stored in FIFO order.

Answer: Please refer to the supporting Web site for source code solution.

6.23 Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released.

Explain how semaphores can be used by a server to limit the number of concurrent connections. Test your solution to this problem by modifying your answer to Exercise 4.13 using Java semaphores so that the number of concurrent connections is limited.

Answer: A semaphore is initialized to the number of allowable open socket connections. When a connection is accepted, the `acquire()` method is called; when a connection is released, the `release()` method is called. If the system reaches the number of allowable socket connections, subsequent calls to `acquire()` will block until an existing connection is terminated and the release method is invoked.

6.24 **The Sleeping-Barber Problem**. A barbershop consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and the customers.

Answer: Please refer to the supporting Web site for source code solution.

6.25 A **barrier** is a thread-synchronization mechanism that allows several threads to run for a period but then forces all threads to wait until all have reached a certain point. Once all threads have reached this point (the barrier), they may all continue.

The following code segment establishes a barrier and creates ten `Worker` threads that will synchronize according to the barrier:

```
Barrier jersey = new Barrier(10);
for (int i = 0; i < 10; i++)
   (new Worker(jersey)).start();
```

Note that the barrier must be initialized to the number of threads that are being synchronized and that each thread has a reference to the same barrier object—`jersey`. Each `Worker` would run as follows:

```
// All threads have access to this barrier
Barrier jersey;
// do some work for a while . . .
// now wait for the others
jersey.waitForOthers();
// now do more work . . .
```

When a thread invokes the method `waitForOthers()`, it will block until all threads have reached this method (the barrier). Once all threads have reached the method, they may all proceed with the remainder of their code. Implement a barrier class using Java synchronization. This class will provide a constructor and the `waitForOthers()` method.
Answer: Please refer to the supporting Web site for source code solution.

6.26 Create a thread pool (see Chapter 4) using Java synchronization. Your thread pool will implement the following API:

`ThreadPool()`–Create a default-sized thread pool.

`ThreadPool(int size)`–Create a thread pool of size `size`.

`void add(Runnable task)`–Add a task to be performed by a thread in the pool.

`void stopPool()`–Stop all threads in the pool.

Your pool will first create a a number of idle threads that await work. Work will be submitted to the pool via the `add()` method, which adds a task implementing the `Runnable` interface. The `add()` method will place the `Runnable` task into a queue. Once a thread in the pool becomes available for work, it will check the queue for any `Runnable` tasks. If there are such tasks, the idle thread will remove the task from the queue and invoke its `run()` method. If the queue is empty, the idle thread will wait to be notified when work becomes available. (The `add()` method will perform a `notify()` when it places a `Runnable` task into the queue to possibly awaken an idle thread awaiting work.) The `stopPool()` method will stop all threads in the pool by invoking their `interrupt()` method (Section 4.5.2). This of course requires that `Runnable` tasks being executed by the thread pool check their interruption status.

Test your solution to this problem by modifying your answer to Exercise 4.13 so that it now uses your thread pool.
Answer: Please refer to the supporting Web site for source code solution.

6.27 Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and —once finished—will return them. As an example, many commercial software packages provide a given number of **licenses**, indicating the number of applications that may run concurrently. When the application is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will only be

granted when an existing license holder terminates the application and a license is returned.

The following program segment is used to manage a finite number of instances of an available resource. The maximum number of resources and the number of available resources are declared as follows:

```
#define MAX_RESOURCES 5
int available_resources = MAX_RESOURCES;
```

When a process wishes to obtain a number of resources, it invokes the `decrease_count()` function:

```
/* decrease available_resources by count resources */
/* return 0 if sufficient resources available, */
/* otherwise return -1 */
int decrease_count(int count) {
   if (available_resources < count)
      return -1;
   else {
      available_resources -= count;

      return 0;
   }
}
```

When a process wants to return a number of resources, it calls the `decrease_count()` function:

```
/* increase available_resources by count */
int increase_count(int count) {
   available_resources += count;

   return 0;
}
```

The preceding program segment produces a race condition. Do the following:

a. Identify the data involved in the race condition.

b. Identify the location (or locations) in the code where the race condition occurs.

c. Using Java synchronization, fix the race condition. Also modify `decreaseCount()` so that a thread blocks if there aren't sufficient resources available.

Answer:

- Identify the data involved in the race condition: The variable available_resources.
- Identify the location (or locations) in the code where the race condition occurs: The code that decrements available_resources and

the code that increments available_resources are the statements that could be involved in race conditions.

- Using a semaphore, fix the race condition: Use a semaphore to represent the available_resources variable and replace increment and decrement operations by semaphore increment and semaphore decrement operations.

6.28 Implement the `Buffer` interface (Figure 3.15) as a bounded buffer using Java's condition variables. Test your solution using Figure 6.13.
Answer: Please refer to the supporting Web site for source code solution.

6.29 Implement the `RWLock` interface (Figure 6.16) using Java's condition variables. You may find it necessary to examine the `signalAll()` method in the `Condition` API.
Answer: Please refer to the supporting Web site for source code solution.

6.30 In Section 6.7.2, we provide an outline of a solution to the dining-philosophers problem using monitors. This exercise will require implementing this solution using Java's condition variables.

Begin by creating five philosophers, each identified by a number 0..4. Each philosopher runs as a separate thread. Philosophers will alternate between thinking and eating. When a philosopher wishes to eat, it invokes the method `takeForks(philNumber)`, where `philNumber` identifies the number of the philosopher wishing to eat. When a philosopher finishes eating, it invokes `returnForks(philNumber)`. Your solution will implement the following interface:

```
public interface DiningServer
{
   // called by a philosopher when it wishes to eat
   public void takeForks(int philNumber);

   // called by a philosopher when it is finished eating
   public void returnForks(int philNumber);
}
```

The implementation of the interface follows the outline of the solution provided in Figure 6.25. Use Java's condition variables to synchronize activity of the philosophers and prevent deadlock.
Answer: Please refer to the supporting Web site for source code solution.

6.31 The outline of the solution to the dining-philosophers problem does not prevent a philosopher from starving. For example, two philosophers—say, philosopher$_1$ and philosopher$_3$—could alternate eating and thinking in such a way that philosopher$_2$ could never eat. Modify your solution to Exercise 6.30 to prevent starvation.
Answer: Please refer to the supporting Web site for source code solution.

CHAPTER 7

Deadlocks

Deadlock is a problem that can arise only in a system with multiple active asynchronous processes. It is important that the students learn the three basic approaches to deadlock: prevention, avoidance, and detection (although the terms *prevention* and *avoidance* are easy to confuse).

It can be useful to pose a deadlock problem in human terms and ask why human systems never deadlock. Can the students transfer this understanding of human systems to computer systems?

Projects can involve simulation: create a list of jobs consisting of requests and releases of resources (single type or multiple types). Ask the students to allocate the resources to prevent deadlock. This basically involves programming the Banker's Algorithm.

The survey paper by Coffman, Elphick, and Shoshani [1971] is good supplemental reading, but you might also consider having the students go back to the papers by Havender [1968], Habermann [1969], and Holt [1971a]. The last two were published in *CACM* and so should be readily available.

Exercises

7.1 Consider the traffic deadlock depicted in Figure 7.1.

a. Show that the four necessary conditions for deadlock indeed hold in this example.

b. State a simple rule for avoiding deadlocks in this system.

Answer:

a. The four necessary conditions for a deadlock are (1) mutual exclusion; (2) hold-and-wait; (3) no preemption; and (4) circular wait. The mutual exclusion condition holds since only one car can occupy a space in the roadway. Hold-and-wait occurs where a car holds onto its place in the roadway while it waits to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly, there is indeed a circular wait as

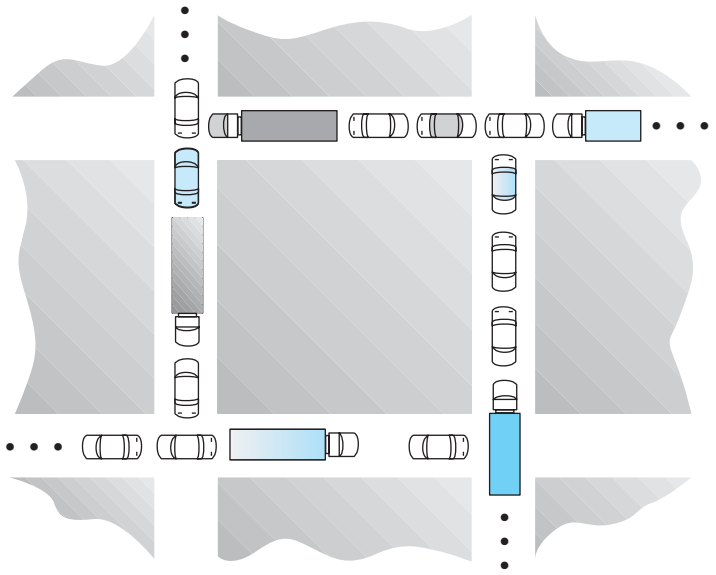


Figure 7.1 Traffic deadlock for Exercise 7.1.

each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.

b. A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear it will not be able immediately to clear the intersection.

7.2 Consider the deadlock situation that could occur in the dining-philosophers problem when the philosophers obtain the chopsticks one at a time. Discuss how the four necessary conditions for deadlock indeed hold in this setting. Discuss how deadlocks could be avoided by eliminating any one of the four conditions.

Answer: Deadlock is possible because the four necessary conditions hold in the following manner: 1) mutual exclusion is required for chopsticks, 2) the philosophers hold onto the chopstick in hand while they wait for the other chopstick, 3) there is no preemption of chopsticks in the sense that a chopstick allocated to a philosopher cannot be forcibly taken away, and 4) there is a possibility of circular wait. Deadlocks could be avoided by overcoming the conditions in the following manner: 1) allow simultaneous sharing of chopsticks, 2) have the philosophers relinquish the first chopstick if they are unable to obtain the other chopstick, 3) allow chopsticks to be forcibly taken away if a philosopher has had a chopstick for a long period of time, and 4) enforce a numbering of the chopsticks and always obtain the lower numbered chopstick before obtaining the higher numbered one.

7.3 A possible solution for preventing deadlocks is to have a single, higher-order resource that must be requested before any other resource. For example, if multiple threads attempt to access the synchronization ob-

jects $A \cdots E$, deadlock is possible. (Such synchronization objects may include mutexes, semaphores, condition variables, etc.) We can prevent the deadlock by adding a sixth object F. Whenever a thread wants to acquire the synchronization lock for any object $A \cdots E$, it must first acquire the lock for object F. This solution is known as **containment**: The locks for objects $A \cdots E$ are contained within the lock for object F. Compare this scheme with the circular-wait scheme of Section 7.4.4.
Answer: This is probably not a good solution because it yields too large a scope. It is better to define a locking policy with as narrow a scope as possible.

7.4 Compare the circular-wait scheme with the deadlock-avoidance schemes (like the banker's algorithm) with respect to the following issues:

a. Runtime overheads

b. System throughput

Answer: A deadlock-avoidance scheme tends to increase the runtime overheads due to the cost of keep track of the current resource allocation. However, a deadlock-avoidance scheme allows for more concurrent use of resources than schemes that statically prevent the formation of deadlock. In that sense, a deadlock-avoidance scheme could increase system throughput.

7.5 Java's locking mechanism (the `synchronized` statement) is considered *r*eentrant. That is, if a thread acquires the lock for an object (by invoking a `synchronized` method or block), it can enter other `synchronized` methods or blocks for the same object. Explain how deadlock would be possible if Java's locking mechanism were not reentrant.
Answer: A thread could simply deadlock itself by first acquiring an object lock (say via a `synchronized` method) and then attempting to acquire the lock again (perhaps in another `synchronized` method or block.) This second attempt would fail if the lock were not reentrant.

7.6 In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?

a. Increase *Available* (new resources added)

b. Decrease *Available* (resource permanently removed from system)

c. Increase *Max* for one process (the process needs more resources than allowed, it may want more)

d. Decrease *Max* for one process (the process decides it does not need that many resources)

e. Increase the number of processes

f. Decrease the number of processes

Answer:

a. Increase *Available* (new resources added)—This could safely be changed without any problems.

b. Decrease *Available* (resource permanently removed from system)—This could have an effect on the system and introduce the possibility of deadlock as the safety of the system assumed there were a certain number of available resources.

c. Increase *Max* for one process (the process needs more resources than allowed, it may want more)—This could have an effect on the system and introduce the possibility of deadlock.

d. Decrease *Max* for one process (the process decides it does not need that many resources)—This could safely be changed without any problems.

e. Increase the number of processes—This could be allowed assuming that resources were allocated to the new process(es) such that the system does not enter an unsafe state.

f. Decrease the number of processes—This could safely be changed without any problems.

7.7 Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free.
Answer: Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

7.8 Consider a system consisting of m resources of the same type, being shared by n processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and m resources

b. The sum of all maximum needs is less than $m + n$

Answer: Using the terminology of Section 7.6.2, we have:

a. $\sum_{i=1}^{n} Max_i < m + n$

b. $Max_i \geq 1$ for all i
Proof: $Need_i = Max_i - Allocation_i$
If there exists a deadlock state then:

c. $\sum_{i=1}^{n} Allocation_i = m$

Use a. to get: $\sum Need_i + \sum Allocation_i = \sum Max_i < m + n$
Use c. to get: $\sum Need_i + m < m + n$
Rewrite to get: $\sum_{i=1}^{n} Need_i < n$

This implies that there exists a process P_i such that $Need_i = 0$. Since $Max_i \geq 1$ it follows that P_i has at least one resource that it can release. Hence the system cannot be in a deadlock state.

7.9 Consider the dining-philosophers problem where the chopsticks are placed at the center of the table and any two of them could be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.
Answer: The following rule prevents deadlock: when a philosopher makes a request for the first chopstick, do not grant the request if there is no other philosopher with two chopsticks and if there is only one chopstick remaining.

7.10 Consider again the setting in the preceding exercise. Assume now that each philosopher requires three chopsticks to eat and that resource requests are still issued separately. Describe some simple rules for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.
Answer: When a philosopher makes a request for a chopstick, allocate the request if: 1) the philosopher has two chopsticks and there is at least one chopstick remaining, 2) the philosopher has one chopstick and there are at least two chopsticks remaining, 3) there is at least one chopstick remaining, and there is at least one philosopher with three chopsticks, 4) the philosopher has no chopsticks, there are two chopsticks remaining, and there is at least one other philosopher with two chopsticks assigned.

7.11 We can obtain the banker's algorithm for a single resource type from the general banker's algorithm simply by reducing the dimensionality of the various arrays by 1. Show through an example that the multiple-resource-type banker's scheme cannot be implemented by individual application of the single-resource-type scheme to each resource type.
Answer: Consider a system with resources A, B, and C and processes P_0, P_1, P_2, P_3, and P_4 with the following values of *Allocation*:

Allocation			
	A	B	C
P_0	0	1	0
P_1	3	0	2
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

and the following value of *Need*:

Need			
	A	B	C
P_0	7	4	3
P_1	0	2	0
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

If the value of *Available* is (2 3 0), we can see that a request from process P_0 for (0 2 0) cannot be satisfied as this lowers *Available* to (2 1 0) and no process could safely finish.

However, if we treat the three resources as three single-resource types of the banker's algorithm, we get the following:
For resource *A* (of which we have 2 available),

	Allocated	Need
P_0	0	7
P_1	3	0
P_2	3	6
P_3	2	0
P_4	0	4

Processes could safely finish in the order P_1, P_3, P_4, P_2, P_0.

For resource *B* (of which we now have 1 available as 2 were assumed assigned to process P_0),

	Allocated	Need
P_0	3	2
P_1	0	2
P_2	0	0
P_3	1	1
P_4	0	3

Processes could safely finish in the order P_2, P_3, P_1, P_0, P_4.

And finally, for For resource *C* (of which we have 0 available),

	Allocated	Need
P_0	0	3
P_1	2	0
P_2	2	0
P_3	1	1
P_4	2	1

Processes could safely finish in the order P_1, P_2, P_0, P_3, P_4.

As we can see, if we use the banker's algorithm for multiple resource types, the request for resources (0 2 0) from process P_0 is denied as it leaves the system in an unsafe state. However, if we consider the banker's algorithm for the three separate resources where we use a single resource

type, the request is granted. Therefore, if we have multiple resource types, we must use the banker's algorithm for multiple resource types.

7.12 Consider the following snapshot of a system:

	Allocation	*Max*	*Available*
	A B C D	*A B C D*	*A B C D*
P_0	0 0 1 2	0 0 1 2	1 5 2 0
P_1	1 0 0 0	1 7 5 0	
P_2	1 3 5 4	2 3 5 6	
P_3	0 6 3 2	0 6 5 2	
P_4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix *Need*?

b. Is the system in a safe state?

c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately?

Answer:

a. What is the content of the matrix *Need*? The values of *Need* for processes P_0 through P_4 respectively are (0, 0, 0, 0), (0, 7, 5, 0), (1, 0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. Is the system in a safe state? Yes. With *Available* being equal to (1, 5, 2, 0), either process P_0 or P_3 could run. Once process P_3 runs, it releases its resources, which allow all other existing processes to run.

c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately? Yes, it can. This results in the value of *Available* being (1, 1, 0, 0). One ordering of processes that can finish is P_0, P_2, P_3, P_1, and P_4.

7.13 What is the optimistic assumption made in the deadlock-detection algorithm? How could this assumption be violated?
Answer: The optimistic assumption is that there will not be any form of circular wait in terms of resources allocated and processes making requests for them. This assumption could be violated if a circular wait does indeed occur in practice.

7.14 Write a Java program that illustrates deadlock by having separate threads attempt to perform operations on different semaphores.
Answer: Please refer to the supporting web page for the source code solution.

7.15 A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a northbound and a southbound farmer get on the bridge at the same time (Vermont farmers are stubborn and are unable to back up). Using semaphores, design an algorithm that prevents

deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, and vice versa).

Answer:

```
semaphore ok_to_cross = 1;

void enter_bridge() {
    ok_to_cross.wait();
}

void exit_bridge() {
    ok_to_cross.signal();
}
```

7.16 Modify your solution to Exercise 7.15 so that it is starvation-free.

Answer:

```
monitor bridge {
    int num_waiting_north = 0;
    int num_waiting_south = 0;
    int on_bridge = 0;
    condition ok_to_cross;
    int prev = 0;

    void enter_bridge_north() {
      num_waiting_north++;
      while (on_bridge ||
             (prev == 0 && num_waiting_south > 0))
      ok_to_cross.wait();
      num_waiting_north--;
      prev = 0;
    }

    void exit_bridge_north() {
      on_bridge = 0;
      ok_to_cross.broadcast();
    }

    void enter_bridge_south() {
      num_waiting_south++;
      while (on_bridge ||
             (prev == 1 && num_waiting_north > 0))
      ok_to_cross.wait();
      num_waiting_south--;
      prev = 1;
    }

    void exit_bridge_south() {
      on_bridge = 0;
      ok_to_cross.broadcast();
    }
}
```

8 Memory Management

Although many systems are demand paged (discussed in Chapter 10), there are still many that are not, and in many cases the simpler memory-management strategies may be better, especially for small dedicated systems. We want the student to learn about all of them: resident monitor, swapping, partitions, paging, and segmentation.

Exercises

8.1 Explain the difference between internal and external fragmentation.
Answer: Internal fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released.

8.2 Consider the following process for generating binaries. A compiler is used to generate the object code for individual modules, and a linkage editor is used to combine multiple object modules into a single program binary. How does the linkage editor change the binding of instructions and data to memory addresses? What information needs to be passed from the compiler to the linkage editor to facilitate the memory binding tasks of the linkage editor?
Answer: The linkage editor has to replace unresolved symbolic addresses with the actual addresses associated with the variables in the final program binary. In order to perform this, the modules should keep track of instructions that refer to unresolved symbols. During linking, each module is assigned a sequence of addresses in the overall program binary and when this has been performed, unresolved references to symbols exported by this binary could be patched in other modules since every other module would contain the list of instructions that need to be patched.

8.3 Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Answer:

a. First-fit:

b. 212K is put in 500K partition

c. 417K is put in 600K partition

d. 112K is put in 288K partition (new partition 288K = 500K − 212K)

e. 426K must wait

f. Best-fit:

g. 212K is put in 300K partition

h. 417K is put in 500K partition

i. 112K is put in 200K partition

j. 426K is put in 600K partition

k. Worst-fit:

l. 212K is put in 600K partition

m. 417K is put in 500K partition

n. 112K is put in 388K partition

o. 426K must wait

In this example, best-fit turns out to be the best.

8.4 Most systems allow programs to allocate more memory to its address space during execution. Data allocated in the heap segments of programs are an example of such allocated memory. What is required to support dynamic memory allocation in the following schemes:

a. contiguous-memory allocation

b. pure segmentation

c. pure paging

Answer:

a. contiguous-memory allocation: might require relocation of the entire program since there is not enough space for the program to grow its allocated memory space.

b. pure segmentation: might also require relocation of the segment that needs to be extended since there is not enough space for the segment to grow its allocated memory space.

c. pure paging: incremental allocation of new pages is possible in this scheme without requiring relocation of the program's address space.

8.5 Compare the main memory organization schemes of contiguous-memory allocation, pure segmentation, and pure paging with respect to the following issues:

a. external fragmentation

b. internal fragmentation

c. ability to share code across processes

Answer: The contiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes die and new processes are initiated. It also does not allow processes to share code, since a process's virtual memory segment is not broken into noncontiguous finegrained segments. Pure segmentation also suffers from external fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes. Segmentation, however, enables processes to share code; for instance, two different processes could share a code segment but have distinct data segments. Pure paging does not suffer from external fragmentation, but instead suffers from internal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space. Paging also enables processes to share code at the granularity of pages.

8.6 On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

Answer: An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process's page table. This is useful when two or more processes need to exchange data—they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.

8.7 Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.

Answer: Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

8.8 Program binaries in many systems are typically structured as follows. Code is stored starting with a small fixed virtual address such as 0. The code segment is followed by the data segment that is used for storing the program variables. When the program starts executing, the stack is allocated at the other end of the virtual address space and is allowed to

grow towards lower virtual addresses. What is the significance of the above structure for the following schemes:

a. contiguous-memory allocation

b. pure segmentation

c. pure paging

Answer: 1) Contiguous-memory allocation requires the operating system to allocate the entire extent of the virtual address space to the program when it starts executing. This could be much larger than the actual memory requirements of the process. 2) Pure segmentation gives the operating system flexibility to assign a small extent to each segment at program startup time and extend the segment if required. 3) Pure paging does not require the operating system to allocate the maximum extent of the virtual address space to a process at startup time, but it still requires the operating system to allocate a large page table spanning all of the program's virtual address space. When a program needs to extend the stack or the heap, it needs to allocate a new page but the corresponding page table entry is preallocated.

8.9 Consider a paging system with the page table stored in memory.

a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time if the entry is there.)

Answer:

a. 400 nanoseconds: 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.

b. Effective access time = 0.75 $\times$ (200 nanoseconds) + 0.25 $\times$ (400 nanoseconds) = 250 nanoseconds.

8.10 Why are segmentation and paging sometimes combined into one scheme?
Answer: Segmentation and paging are often combined in order to improve upon each other. Segmented paging is helpful when the page table becomes very large. A large contiguous section of the page table that is unused can be collapsed into a single-segment table entry with a page-table address of zero. Paged segmentation handles the case of having very long segments that require a lot of time for allocation. By paging the segments, we reduce wasted memory due to external fragmentation as well as simplify the allocation.

8.11 Explain why it is easier to share a reentrant module using segmentation than it is to do so when pure paging is used.
Answer: Since segmentation is based on a logical division of memory rather than a physical one, segments of any size can be shared with only one entry in the segment tables of each user. With paging there must be a common entry in the page tables for each page that is shared.

8.12 Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?

a. 0,430

b. 1,10

c. 2,500

d. 3,400

e. 4,112

Answer:

a. 219 + 430 = 649

b. 2300 + 10 = 2310

c. illegal reference, trap to operating system

d. 1327 + 400 = 1727

e. illegal reference, trap to operating system

8.13 What is the purpose of paging the page tables?
Answer: In certain situations the page tables could become large enough that by paging the page tables, one could simplify the memory allocation problem (by ensuring that everything is allocated as fixed-size pages as opposed to variable-sized chunks) and also enable the swapping of portions of page table that are not currently used.

8.14 Consider the hierarchical paging scheme used by the VAX architecture. How many memory operations are performed when an user program executes a memory load operation?
Answer: When a memory load operation is performed, there are three memory operations that might be performed. One is to translate the position where the page table entry for the page could be found (since page tables themselves are paged). The second access is to access the page table entry itself, while the third access is the actual memory load operation.

8.15 Compare the segmented paging scheme with the hashed page tables scheme for handling large address spaces. Under what circumstances is one scheme preferable to the other?
Answer: When a program occupies only a small portion of its large virtual address space, a hashed page table might be preferred due to its smaller size. The disadvantage with hashed page tables however is

the problem that arises due to conflicts in mapping multiple pages onto the same hashed page table entry. If many pages map to the same entry, then traversing the list corresponding to that hash table entry could incur a significant overhead; such overheads are minimal in the segmented paging scheme where each page table entry maintains information regarding only one page.

8.16 Consider the Intel address-translation scheme shown in Figure 8.22.

a. Describe all the steps that the Intel 80386 takes in translating a logical address into a physical address.

b. What are the advantages to the operating system of hardware that provides such complicated memory-translation hardware?

c. Are there any disadvantages to this address-translation system? If so, what are they? If not, why is it not used by every manufacturer?

Answer:

a. The selector is an index into the segment descriptor table. The segment descriptor result plus the original offset is used to produce a linear address with a dir, page, and offset. The dir is an index into a page directory. The entry from the page directory selects the page table, and the page field is an index into the page table. The entry from the page table, plus the offset, is the physical address.

b. Such a page-translation mechanism offers the flexibility to allow most operating systems to implement their memory scheme in hardware, instead of having to implement some parts in hardware and some in software. Because it can be done in hardware, it is more efficient (and the kernel is simpler).

c. Address translation can take longer due to the multiple table lookups it can invoke. Caches help, but there will still be cache misses.

Virtual Memory

Virtual memory can be a very interesting subject since it has so many different aspects: page faults, managing the backing store, page replacement, frame allocation, thrashing, page size. The objectives of this chapter are to explain these concepts and show how paging works.

A simulation is probably the easiest way to allow the students to program several of the page-replacement algorithms and see how they really work. If an interactive graphics display can be used to display the simulation as it works, the students may be better able to understand how paging works. We also present an exercise that asks the student to develop a Java program that implements the FIFO and LRU page-replacement algorithms.

Exercises

9.1 Give an example that illustrates the problem with restarting the block move instruction (MVC) on the IBM 360/370 when the source and destination regions are overlapping.
Answer: Assume that the page boundary is at 1024 and the move instruction is moving values from a source region of 800:1200 to a target region of 700:1100. Assume that a page fault occurs while accessing location 1024. By this time the locations of 800:923 have been overwritten with the new values and therefore restarting the block move instruction would result in copying the new values in 800:923 to locations 700:823, which is incorrect.

9.2 Discuss the hardware support required to support demand paging.
Answer: For every memory-access operation, the page table needs to be consulted to check whether the corresponding page is resident or not and whether the program has read or write privileges for accessing the page. These checks have to be performed in hardware. A TLB could serve as a cache and improve the performance of the lookup operation.

9.3 What is the copy-on-write feature and under what circumstances is it beneficial to use this feature? What is the hardware support required to implement this feature?

Answer: When two processes are accessing the same set of program values (for instance, the code segment of the source binary), then it is useful to map the corresponding pages into the virtual address spaces of the two programs in a write-protected manner. When a write does indeed take place, then a copy must be made to allow the two programs to individually access the different copies without interfering with each other. The hardware support required to implement is simply the following: on each memory access, the page table needs to be consulted to check whether the page is write protected. If it is indeed write protected, a trap would occur and the operating system could resolve the issue.

9.4 A certain computer provides its users with a virtual-memory space of 2^{32} bytes. The computer has 2^{18} bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.
Answer: The virtual address in binary form is

0001 0001 0001 0010 0011 0100 0101 0110

Since the page size is 2^{12}, the page table size is 2^{20}. Therefore the low-order 12 bits "0100 0101 0110" are used as the displacement into the page, while the remaining 20 bits "0001 0001 0001 0010 0011" are used as the displacement in the page table.

9.5 Assume we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified, and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?
Answer:

$$\begin{aligned} 0.2\ \mu\text{sec} &= (1-P) \times 0.1\ \mu\text{sec} + (0.3P) \times 8\ \text{millisec} + (0.7P) \times 20\ \text{millisec} \\ 0.1 &= -0.1P + 2400\,P + 14000\,P \\ 0.1 &\simeq 16{,}400\,P \\ P &\simeq 0.000006 \end{aligned}$$

9.6 Assume that you are monitoring the rate at which the pointer in the clock algorithm (which indicates the candidate page for replacement) moves. What can you say about the system if you notice the following behavior:

a. pointer is moving fast

b. pointer is moving slow

Answer: If the pointer is moving fast, then the program is accessing a large number of pages simultaneously. It is most likely that during the

period between the point at which the bit corresponding to a page is cleared and it is checked again, the page is accessed again and therefore cannot be replaced. This results in more scanning of the pages before a victim page is found. If the pointer is moving slow, then the virtual memory system is finding candidate pages for replacement extremely efficiently, indicating that many of the resident pages are not being accessed.

9.7 Discuss situations under which the least frequently used page-replacement algorithm generates fewer page faults than the least recently used page-replacement algorithm. Also discuss under what circumstance the opposite holds.

Answer: Consider the following sequence of memory accesses in a system that can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence "1 2 3 4 5 2," the least recently used algorithm performs better.

9.8 Discuss situations under which the most frequently used page-replacement algorithm generates fewer page faults than the least recently used page-replacement algorithm. Also discuss under what circumstance the opposite holds.

Answer: Consider the sequence in a system that holds four pages in memory: 1 2 3 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4 while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence "1 2 3 4 4 4 5 1," the LRU algorithm makes the right decision.

9.9 The VAX/VMS system uses a FIFO replacement algorithm for resident pages and a free-frame pool of recently used pages. Assume that the free-frame pool is managed using the least recently used replacement policy. Answer the following questions:

a. If a page fault occurs and if the page does not exist in the free-frame pool, how is free space generated for the newly requested page?

b. If a page fault occurs and if the page exists in the free-frame pool, how is the resident page set and the free-frame pool managed to make space for the requested page?

c. What does the system degenerate to if the number of resident pages is set to one?

d. What does the system degenerate to if the number of pages in the free-frame pool is zero?

Answer:

a. When a page fault occurs and if the page does not exist in the free-frame pool, then one of the pages in the free-frame pool is evicted to disk, creating space for one of the resident pages to be

moved to the free-frame pool. The accessed page is then moved to the resident set.

b. When a page fault occurs and if the page exists in the free-frame pool, then it is moved into the set of resident pages, while one of the resident pages is moved to the free-frame pool.

c. When the number of resident pages is set to one, then the system degenerates into the page replacement algorithm used in the free-frame pool, which is typically managed in a LRU fashion.

d. When the number of pages in the free-frame pool is zero, then the system degenerates into a FIFO page-replacement algorithm.

9.10 Consider a demand-paging system with the following time-measured utilizations:

CPU utilization	20%
Paging disk	97.7%
Other I/O devices	5%

Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.

a. Install a faster CPU.

b. Install a bigger paging disk.

c. Increase the degree of multiprogramming.

d. Decrease the degree of multiprogramming.

e. Install more main memory.

f. Install a faster hard disk or multiple controllers with multiple hard disks.

g. Add prepaging to the page fetch algorithms.

h. Increase the page size.

Answer: The system obviously is spending most of its time paging, indicating over-allocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging drum.

a. Install a faster CPU—No.

b. Install a bigger paging disk—No.

c. Increase the degree of multiprogramming—No.

d. Decrease the degree of multiprogramming—Yes.

e. Install more main memory—Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.

f. Install a faster hard disk or multiple controllers with multiple hard disks—Also an improvement, for as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.

g. Add prepaging to the page fetch algorithms—Again, the CPU will get more data faster, so it will be more in use. This is only the case if the paging action is amenable to prefetching (i.e., some of the access is sequential).

h. Increase the page size—Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is more or less random, more paging action could ensue because fewer pages can be kept in memory and more data is transferred per page fault. So this change is as likely to decrease utilization as it is to increase it.

9.11 Suppose that a machine provides instructions that can access memory locations using the one-level indirect addressing scheme. What is the sequence of page faults incurred when all of the pages of a program are currently non resident and the first instruction of the program is an indirect memory load operation? What happens when the operating system is using a per-process frame allocation technique and only two pages are allocated to this process?

Answer: The following page faults take place: page fault to access the instruction, a page fault to access the memory location that contains a pointer to the target memory location, and a page fault when the target memory location is accessed. The operating system will generate three page faults with the third page replacing the page containing the instruction. If the instruction needs to be fetched again to repeat the trapped instruction, then the sequence of page faults will continue indefinitely. If the instruction is cached in a register, then it will be able to execute completely after the third page fault.

9.12 Suppose that your replacement policy (in a paged system) is to examine each page regularly and to discard that page if it has not been used since the last examination. What would you gain and what would you lose by using this policy rather than LRU or second-chance replacement?

Answer: Such an algorithm could be implemented with the use of a reference bit. After every examination, the bit is set to zero; set back to one if the page is referenced. The algorithm would then select an arbitrary page for replacement from the set of unused pages since the last examination.

The advantage of this algorithm is its simplicity—nothing other than a reference bit need be maintained. The disadvantage of this algorithm is that it ignores locality by using only a short time frame for determining whether to evict a page or not. For example, a page may be part of the working set of a process, but may be evicted because it was not referenced since the last examination (that is, not all pages in the working set may be referenced between examinations).

9.13 A page-replacement algorithm should minimize the number of page faults. We can do this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages that are associated with that frame. Then, to replace a page, we search for the page frame with the smallest counter.

a. Define a page-replacement algorithm using this basic idea. Specifically address the problems of (1) what the initial value of the counters is, (2) when counters are increased, (3) when counters are decreased, and (4) how the page to be replaced is selected.

b. How many page faults occur for your algorithm for the following reference string, for four page frames?

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

c. What is the minimum number of page faults for an optimal page-replacement strategy for the reference string in part b with four page frames?

Answer:

a. Define a page-replacement algorithm addressing the problems of:
   1. Initial value of the counters—0.
   2. Counters are increased—whenever a new page is associated with that frame.
   3. Counters are decreased—whenever one of the pages associated with that frame is no longer required.
   4. How the page to be replaced is selected—find a frame with the smallest counter. Use FIFO for breaking ties.

b. 14 page faults

c. 11 page faults

9.14 Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory.

Assume that 80 percent of the accesses are in the associative memory and that, of those remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Answer:

$$\begin{aligned} \text{effective access time} &= (0.8) \times (1\ \mu\text{sec}) \\ &\quad + (0.1) \times (2\ \mu\text{sec}) + (0.1) \times (5002\ \mu\text{sec}) \\ &= 501.2\ \mu\text{sec} \\ &= 0.5\ \text{millisec} \end{aligned}$$

9.15 What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?
Answer: Thrashing is caused by underallocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

9.16 Is it possible for a process to have two working sets, one representing data and another representing code? Explain.
Answer: Yes, in fact many processors provide two TLBs for this very reason. As an example, the code being accessed by a process may retain the same working set for a long period of time. However, the data the code accesses may change, thus reflecting a change in the working set for data accesses.

9.17 Consider the parameter Δ used to define the working-set window in the working-set model. What is the effect of setting Δ to a small value on the page fault frequency and the number of active (non-suspended) processes currently executing in the system? What is the effect when Δ is set to a very high value?
Answer: When Δ is set to a small value, then the set of resident pages for a process might be underestimated, allowing a process to be scheduled even though all of its required pages are not resident. This could result in a large number of page faults. When Δ is set to a large value, then a process's resident set is overestimated and this might prevent many processes from being scheduled even though their required pages are resident. However, once a process is scheduled, it is unlikely to generate page faults since its resident set has been overestimated.

9.18 Assume there is an initial 1024 KB segment where memory is allocated using the Buddy system. Using Figure 9.27 as a guide, draw the tree illustrating how the following memory requests are allocated:

- request 240 bytes
- request 120 bytes
- request 60 bytes
- request 130 bytes

Next, modify the tree for the following releases of memory. Perform coalescing whenever possible:

- release 240 bytes
- release 60 bytes
- release 120 bytes

Answer: The following allocation is made by the Buddy system: The 240-byte request is assigned a 256-byte segment. The 120-byte request is assigned a 128-byte segement, the 60-byte request is assigned a 64-byte segment and the 130-byte request is assigned a 256-byte segment.

After the allocation, the following segment sizes are available: 64-bytes, 256-bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

After the releases of memory, the only segment in use would be a 256-byte segment containing 130 bytes of data. The following segments will be free: 256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

9.19 The slab allocation algorithm uses a separate cache for each different object type. Assuming there is one cache per object type, explain why this doesn't scale well with multiple CPUs. What could be done to address this scalability issue?

Answer: This has long been a problem with the slab allocator—poor scalability with multiple CPUs. The issue comes from having to lock the global cache when it is being access. This has the effect of serializing cache accesses on multiprocessor systems. Solaris has addressed this by introducing a per-CPU cache, rather than a single global cache.

9.20 Consider a system that allocates pages of different sizes to its processes. What are the advantages of such a paging scheme? What modifications to the virtual memory system provide this functionality?

Answer: The program could have a large code segment or use large-sized arrays as data. These portions of the program could be allocated to larger pages, thereby decreasing the memory overheads associated with a page table. The virtual memory system would then have to maintain multiple free lists of pages for the different sizes and also needs to have more complex code for address translation to take into account different page sizes.

9.21 Write a program that implements the FIFO and LRU page-replacement algorithms presented in this chapter. First, generate a random page-reference string where page numbers range from 0 to 9. Apply the random page-reference string to each algorithm, and record the number of page faults incurred by each algorithm. Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used. Your algorithms will be based on the abstract class depicted in Figure 9.30.

Design and implement two classes—`LRU` and `FIFO`—that extend `ReplacementAlgorithm`. Each of these classes will implement the `insert()` method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm.

There are two classes available online to test your algorithm:

a. `PageGenerator`—a class that generates page-reference strings with page numbers ranging from 0 to 9. The size of the reference string is passed to the `PageGenerator` constructor. Once a `PageGenerator` object is constructed, the `getReferenceString()` method returns the reference string as an array of integers.

b. `Test`—used to test your FIFO and LRU implementations of the `ReplacementAlgorithm` abstract class. `Test` is invoked as

```
java Test <reference string #> <# of page frames>.
```

nce you have implemented the FIFO and LRU algorithms, experiment with a different number of page frames for a given reference string and record the number of page faults. Does one algorithm perform better than the other? For a given reference-string size, what is the optimal number of page frames?

Answer: Please refer to the supporting Web site for source code solution.

File-System Interface

Files are the most obvious object that operating systems manipulate. Everything is typically stored in files: programs, data, output, etc. The student should learn what a file is to the operating system and what the problems are (providing naming conventions to allow files to be found by user programs, protection).

Two problems can crop up in this chapter. First, terminology may be different between your system and the book. This can be used to drive home the point that concepts are important and terms must be clearly defined when you get to a new system. Second, it may be difficult to motivate students to learn about directory structures that are not the ones on the system they are using. This can best be overcome if the students have two very different systems to consider, such as a single-user system for a microcomputer and a large, university time-shared system.

Projects might include a report about the details of the file system for the local system. It is also possible to write programs to implement a simple file system either in memory (allocate a large block of memory that is used to simulate a disk) or on top of an existing file system. In many cases, the design of a file system is an interesting project of its own.

Exercises

10.1 Consider a file system where a file can be deleted and its disk space reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?

Answer: Let F1 be the old file and F2 be the new file. A user wishing to access F1 through an existing link will actually access F2. Note that the access protection for file F1 is used rather than the one associated with F2.

This problem can be avoided by insuring that all links to a deleted file are deleted also. This can be accomplished in several ways:

a. maintain a list of all links to a file, removing each of them when the file is deleted

b. retain the links, removing them when an attempt is made to access a deleted file

c. maintain a file reference list (or counter), deleting the file only after all links or references to that file have been deleted

10.2 The open-file table is used to maintain information about files that are currently open. Should the operating system maintain a separate table for each user or just maintain one table that contains references to files that are being accessed by all users at the current time? If the same file is being accessed by two different programs or users, should there be separate entries in the open file table?

Answer: By keeping a central open-file table, the operating system can perform the following operation that would be infeasible otherwise. Consider a file that is currently being accessed by one or more processes. If the file is deleted, then it should not be removed from the disk until all processes accessing the file have closed it. This check can be performed only if there is centralized accounting of number of processes accessing the file. On the other hand, if two processes are accessing the file, then two separate states need to be maintained to keep track of the current location of which parts of the file are being accessed by the two processes. This requires the operating system to maintain separate entries for the two processes.

10.3 What are the advantages and disadvantages of a system providing mandatory locks instead of providing advisory locks whose usage is left to the users' discretion?

Answer: In many cases, separate programs might be willing to tolerate concurrent access to a file without requiring the need to obtain locks and thereby guaranteeing mutual exclusion to the files. Mutual exclusion could be guaranteed by other program structures such as memory locks or other forms of synchronization. In such situations, the mandatory locks would limit the flexibility in how files could be accessed and might also increase the overheads associated with accessing files.

10.4 What are the advantages and disadvantages of recording the name of the creating program with the file's attributes (as is done in the Macintosh Operating System)?

Answer: By recording the name of the creating program, the operating system is able to implement features (such as automatic program invocation when the file is accessed) based on this information. It does add overhead in the operating system and require space in the file descriptor, however.

10.5 Some systems automatically open a file when it is referenced for the first time, and close the file when the job terminates. Discuss the advantages and disadvantages of this scheme as compared to the more traditional one, where the user has to open and close the file explicitly.

Answer: Automatic opening and closing of files relieves the user from the invocation of these functions, and thus makes it more convenient to the user; however, it requires more overhead than the case where explicit opening and closing is required.

10.6 If the operating system were to know that a certain application is going to access the file data in a sequential manner, how could it exploit this information to improve performance?
Answer: When a block is accessed, the file system could prefetch the subsequent blocks in anticipation of future requests to these blocks. This prefetching optimization would reduce the waiting time experienced by the process for future requests.

10.7 Give an example of an application that could benefit from operating system support for random access to indexed files.
Answer: An application that maintains a database of entries could benefit from such support. For instance, if a program is maintaining a student database, then accesses to the database cannot be modeled by any predetermined access pattern. The access to records are random and locating the records would be more efficient if the operating system were to provide some form of tree-based index.

10.8 Discuss the merits and demerits of supporting links to files that cross mount points (that is, the file link refers to a file that is stored in a different volume).
Answer: The advantage is that there is greater transparency in the sense that the user does not need to be aware of mount points and create links in all scenarios. The disadvantage however is that the file system containing the link might be mounted while the file system containing the target file might not be, and therefore one cannot provide transparent access to the file in such a scenario; the error condition would expose to the user that a link is a dead link and that the link does indeed cross file system boundaries.

10.9 Some systems provide file sharing by maintaining a single copy of a file; other systems maintain several copies, one for each of the users sharing the file. Discuss the relative merits of each approach.
Answer: With a single copy, several concurrent updates to a file may result in user obtaining incorrect information, and the file being left in an incorrect state. With multiple copies, there is storage waste and the various copies may not be consistent with respect to each other.

10.10 Discuss the advantages and disadvantages of associating with remote file systems (stored on file servers) a different set of failure semantics from that associated with local file systems.
Answer: The advantage is that the application can deal with the failure condition in a more intelligent manner if it realizes that it incurred an error while accessing a file stored in a remote file system. For instance, a file open operation could simply fail instead of hanging when accessing a remote file on a failed server and the application could deal with the failure in the best possible manner; if the operation were simply to hang, then the entire application hangs, which is not desirable. The disadvantage however is the lack of uniformity in failure semantics and the resulting complexity in application code.

10.11 What are the implications of supporting UNIX consistency semantics for shared access for those files that are stored on remote file systems?

Answer: UNIX consistency semantics requires updates to a file to be immediately available to other processes. Supporting such a semantics for shared files on remote file systems could result in the following inefficiencies: all updates by a client have to be immediately reported to the file server instead of being batched (or even ignored if the updates are to a temporary file), and updates have to be communicated by the file server to clients caching the data immediately, again resulting in more communication.

11 File-System Implementation

In this chapter we discuss various methods for storing information on secondary storage. The basic issues are device directory, free space management, and space allocation on a disk.

Exercises

11.1 Consider a file system that uses a modifed contiguous-allocation scheme with support for extents. A file is a collection of extents, with each extent corresponding to a contiguous set of blocks. A key issue in such systems is the degree of variability in the size of the extents. What are the advantages and disadvantages of the following schemes?

a. All extents are of the same size, and the size is predetermined.

b. Extents can be of any size and are allocated dynamically.

c. Extents can be of a few fixed sizes, and these sizes are predetermined.

Answer: If all extents are of the same size, and the size is predetermined, then it simplifies the block allocation scheme. A simple bit map or free list for extents would suffice. If the extents can be of any size and are allocated dynamically, then more complex allocation schemes are required. It might be difficult to find an extent of the appropriate size and there might be external fragmentation. One could use the Buddy system allocator discussed in the previous chapters to design an appropriate allocator. When the extents can be of a few fixed sizes, and these sizes are predetermined, one would have to maintain a separate bitmap or free list for each possible size. This scheme is of intermediate complexity and of intermediate flexibility in comparison to the earlier schemes.

11.2 What are the advantages of the variation of linked allocation that uses a FAT to chain together the blocks of a file?

Answer: The advantage is that while accessing a block that is stored at the middle of a file, its location can be determined by chasing the pointers stored in the FAT as opposed to accessing all of the individual blocks of the file in a sequential manner to find the pointer to the target block. Typically, most of the FAT can be cached in memory and therefore the pointers can be determined with just memory accesses instead of having to access the disk blocks.

11.3 Consider a system where free space is kept in a free-space list.

a. Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.

b. Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at */a/b/c*? Assume that none of the disk blocks is currently being cached.

c. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

Answer:

a. In order to reconstruct the free list, it would be necessary to perform "garbage collection." This would entail searching the entire directory structure to determine which pages are already allocated to jobs. Those remaining unallocated pages could be relinked as the free-space list.

b. The free-space list pointer could be stored on the disk, perhaps in several places.

c. The pointer could be stored in an on-disk data structure or perhaps in non-volatile RAM (NVRAM.)

11.4 Some file systems allow disk storage to be allocated at different levels of granularity. For instance, a file system could allocate 4 KB of disk space as a single 4-KB block or as eight 512-byte blocks. How could we take advantage of this flexibility to improve performance? What modifications would have to be made to the free-space management scheme in order to support this feature?
Answer: Such a scheme would decrease internal fragmentation. If a file is 5 KB, then it could be allocated a 4 KB block and two contiguous 512-byte blocks. In addition to maintaining a bitmap of free blocks, one would also have to maintain extra state regarding which of the subblocks are currently being used inside a block. The allocator would then have to examine this extra state to allocate subblocks and coalesce the subblocks to obtain the larger block when all of the subblocks become free.

11.5 Discuss how performance optimizations for file systems might result in difficulties in maintaining the consistency of the systems in the event of computer crashes.
Answer: The primary difficulty that might arise is due to delayed updates of data and metadata. Updates could be delayed in the hope

that the same data might be updated in the future or that the updated data might be temporary and might be deleted in the near future. However, if the system were to crash without having committed the delayed updates, then the consistency of the file system is destroyed.

11.6 Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of the three allocation strategies (contiguous, linked, and indexed), answer these questions:

a. How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)

b. If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

Answer: Let Z be the starting file address (block number).

a. Contiguous. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.
   1. Add X to Z to obtain the physical block number. Y is the displacement into that block.
   2. 1

b. Linked. Divide the logical physical address by 511 with X and Y the resulting quotient and remainder respectively.
   1. Chase down the linked list (getting $X + 1$ blocks). $Y + 1$ is the displacement into the last physical block.
   2. 4

c. Indexed. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.
   1. Get the index block into memory. Physical block address is contained in the index block at location X. Y is the displacement into the desired physical block.
   2. 2

11.7 Fragmentation on a storage device could be eliminated by recompaction of the information. Typical disk devices do not have relocation or base registers (such as are used when memory is to be compacted), so how can we relocate files? Give three reasons why recompacting and relocation of files often are avoided.

Answer: Relocation of files on secondary storage involves considerable overhead—data blocks have to be read into main memory and written back out to their new locations. Furthermore, relocation registers apply only to *sequential* files, and many disk files are not sequential. For this same reason, many new files will not require contiguous disk space; even sequential files can be allocated noncontiguous blocks if links between logically sequential blocks are maintained by the disk system.

11.8 In what situations would using memory as a RAM disk be more useful than using it as a disk cache?
Answer: In cases where the user (or system) knows exactly what data is going to be needed. Caches are algorithm-based, while a RAM disk is user-directed.

11.9 Consider the following augmentation of a remote-file-access protocol. Each client maintains a name cache that caches translations from file names to corresponding file handles. What issues should we take into account in implementing the name cache?
Answer: One issue is maintaining consistency of the name cache. If the cache entry becomes inconsistent, then either it should be updated or its inconsistency should be detected when it is used next. If the inconsistency is detected later, then there should be a fallback mechanism for determining the new translation for the name. Also, another related issue is whether a name lookup is performed one element at a time for each subdirectory in the pathname or whether it is performed in a single shot at the server. If it is perfomed one element at a time, then the client might obtain more information regarding the translations for all of the intermediate directories. On the other hand, it increases the network traffic as a single name lookup causes a sequence of partial name lookups.

11.10 Explain why logging metadata updates ensures recovery of a file system after a file system crash.
Answer: For a file system to be recoverable after a crash, it must be consistent or must be able to be made consistent. Therefore, we have to prove that logging metadata updates keeps the file system in a consistent or able-to-be-consistent state. For a file system to become inconsistent, the metadata must be written incompletely or in the wrong order to the file system data structures. With metadata logging, the writes are made to a sequential log. The complete transaction is written there before it is moved to the file system structures. If the system crashes during file system data updates, the updates can be completed based on the information in the log. Thus, logging ensures that file system changes are made completely (either before or after a crash). The order of the changes is guaranteed to be correct because of the sequential writes to the log. If a change was made incompletely to the log, it is discarded, with no changes made to the file system structures. Therefore, the structures are either consistent or can be trivially made consistent via metadata logging replay.

11.11 Consider the following backup scheme:

- **Day 1**. Copy to a backup medium all files from the disk.
- **Day 2**. Copy to another medium all files changed since day 1.
- **Day 3**. Copy to another medium all files changed since day 1.

This contrasts to the schedule given in Section 11.7.2 by having all subsequent backups copy all files modified since the first full backup. What are the benefits of this system over the one in Section 11.7.2?

What are the drawbacks? Are restore operations made easier or more difficult? Explain your answer.

Answer: Restores are easier because you can go to the last backup tape, rather than the full tape. No intermediate tapes need be read. More tape is used as more files change.

Mass Storage Structure

In this chapter we describe the internal data structures and algorithms used by the operating system to implement the file system. We also discuss the lowest level of the file system, the secondary storage structure. We first describe disk-head-scheduling algorithms. Next we discuss disk formatting and management of boot blocks, damaged blocks, and swap space. We end with coverage of disk reliability and stable storage.

The basic implementation of disk scheduling should be fairly clear: requests, queues, servicing; so the main new consideration is the actual algorithms: FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK. Simulation may be the best way to involve the student with the algorithms. Exercise 12.7 provides a question amenable to a small but open-ended simulation study.

The paper by Worthington et al. [1994] gives a good presentation of the disk-scheduling algorithms and their evaluation. Be suspicious of the results of the disk-scheduling papers from the 1970s, such as Teory and Pinkerton [1972], because they generally assume that the seek time function is linear, rather than a square root. The paper by Lynch [1972b] shows the importance of keeping the overall system context in mind when choosing scheduling algorithms. Unfortunately, it is fairly difficult to find.

Chapter 2 introduced the concept of primary, secondary, and tertiary storage. In this chapter, we discuss tertiary storage in more detail. First, we describe the types of storage devices used for tertiary storage. Next, we discuss the issues that arise when an operating system uses tertiary storage. Finally, we consider some performance aspects of tertiary storage systems.

Exercises

12.1 None of the disk-scheduling disciplines, except FCFS, is truly fair (starvation may occur).

a. Explain why this assertion is true.

b. Describe a way to modify algorithms such as SCAN to ensure fairness.

c. Explain why fairness is an important goal in a time-sharing system.

d. Give three or more examples of circumstances in which it is important that the operating system be *unfair* in serving I/O requests.

Answer:

a. New requests for the track over which the head currently resides can theoretically arrive as quickly as these requests are being serviced.

b. All requests older than some predetermined age could be "forced" to the top of the queue, and an associated bit for each could be set to indicate that no new request could be moved ahead of these requests. For SSTF, the rest of the queue would have to be reorganized with respect to the last of these "old" requests.

c. To prevent unusually long response times.

d. Paging and swapping should take priority over user requests. It may be desirable for other kernel-initiated I/O, such as the writing of file system metadata, to take precedence over user I/O. If the kernel supports real-time process priorities, the I/O requests of those processes should be favored.

12.2 Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

a. FCFS

b. SSTF

c. SCAN

d. LOOK

e. C-SCAN

Answer:

a. The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. The total seek distance is 7081.

b. The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774. The total seek distance is 1745.

c. The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86. The total seek distance is 9769.

d. The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86. The total seek distance is 3319.

e. The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 86, 130. The total seek distance is 9813.

f. (Bonus.) The C-LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130. The total seek distance is 3363.

12.3 From elementary physics, we know that when an object is subjected to a constant acceleration a, the relationship between distance d and time t is given by $d = \frac{1}{2}at^2$. Suppose that, during a seek, the disk in Exercise 12.2 accelerates the disk arm at a constant rate for the first half of the seek, then decelerates the disk arm at the same rate for the second half of the seek. Assume that the disk can perform a seek to an adjacent cylinder in 1 millisecond and a full-stroke seek over all 5000 cylinders in 18 milliseconds.

a. The distance of a seek is the number of cylinders that the head moves. Explain why the seek time is proportional to the square root of the seek distance.

b. Write an equation for the seek time as a function of the seek distance. This equation should be of the form $t = x + y\sqrt{L}$, where t is the time in milliseconds and L is the seek distance in cylinders.

c. Calculate the total seek time for each of the schedules in Exercise 12.2. Determine which schedule is the fastest (has the smallest total seek time).

d. The *percentage speedup* is the time saved divided by the original time. What is the percentage speedup of the fastest schedule over FCFS?

Answer:

a. Solving $d = \frac{1}{2}at^2$ for t gives $t = \sqrt{(2d/a)}$.

b. Solve the simultaneous equations $t = x + y\sqrt{L}$ that result from $(t = 1, L = 1)$ and $(t = 18, L = 4999)$ to obtain $t = 0.7561 + 0.2439\sqrt{L}$.

c. The total seek times are: FCFS 65.20; SSTF 31.52; SCAN 62.02; LOOK 40.29; C-SCAN 62.10 (and C-LOOK 40.42). Thus, SSTF is fastest here.

d. $(65.20 - 31.52)/65.20 = 0.52$. The percentage speedup of SSTF over FCFS is 52%, with respect to the seek time. If we include the overhead of rotational latency and data transfer, the percentage speedup will be less.

12.4 Suppose that the disk in Exercise 12.3 rotates at 7,200 RPM.

a. What is the average rotational latency of this disk drive?

b. What seek distance can be covered in the time that you found for part a?

Answer:

a. 7,200 rpm gives 120 rotations per second. Thus, a full rotation takes 8.33 ms, and the average rotational latency (a half rotation) takes 4.167 ms.

b. Solving $t = 0.7561 + 0.2439\sqrt{L}$ for $t = 4.167$ gives $L = 195.58$, so we can seek over 195 tracks (about 4% of the disk) during an average rotational latency.

12.5 One possible implementation of the SSTF algorithm uses a binary tree data structure. We initialize the binary tree by storing the position of the head before any requests are inserted into the tree. Requests are stored in the tree as they arrive, using the cylinder of the request as the key of the binary tree.

When a request has been serviced, the scheduler compares the cylinder position of the root node with the cylinder positions of its predecessor and successor, servicing the closer of the two. The root node, which has already been serviced, is then removed and replaced by the next request to be serviced—either the predecessor or the successor. This process continues until the tree has a single element, which represents the last request to have been serviced.

Based on this proposed scheme, answer the following:

a. Assuming the initial position of the head is at cylinder 53, insert the following request sequence into the binary tree:

98, 183, 37, 122, 14, 124, 65, 67

b. Illustrate execution of the proposed algorithm. After each node is serviced, draw the binary tree, highlighting the predecessor and successor nodes.

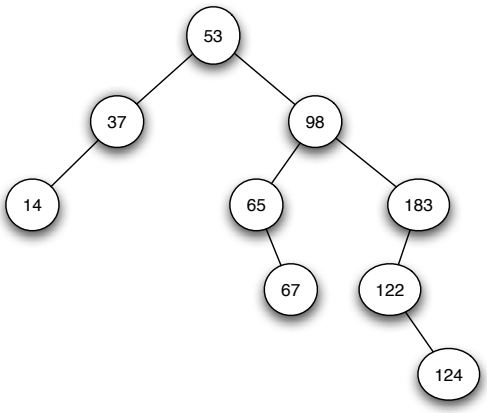


Figure 12.1 Figure 1.

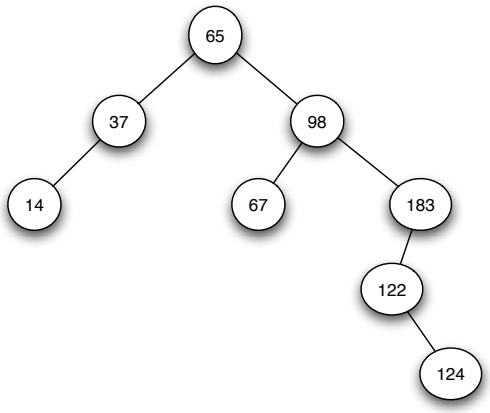


Figure 12.2 Figure 2.

c. Outline an algorithm for the SSTF disk scheduler. Assume the binary tree provides `insert()`, `remove()`, `size()`, `successor()`, and `predecessor()` operations.

d. Are there any problems with this proposed scheme? If so, how might they be fixed?

Answer:

a. The initial tree appears as shown in Figure 12.1

b. The algorithm begins with *root* = *53*, *predecessor* = *37*, and *successor* = *65*. Since the successor is closest to the root, the root is replaced with the successor, yielding the tree that is shown in Figure 12.2

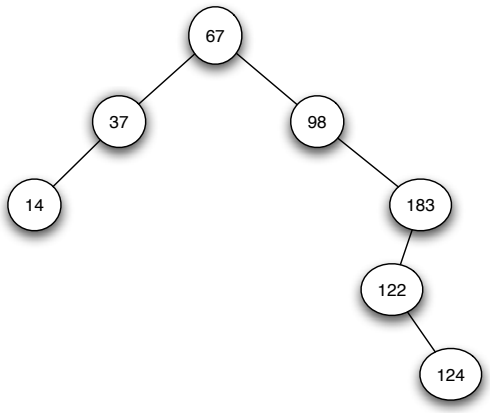


Figure 12.3 Figure 3.

At this point, *root* = *65*, *predecessor* = *37*, and *successor* = *67*. As the successor is closest, we remove the root and replace it with the successor. The tree now appears as shown in Figure 12.3.
We continue this process until the tree has a single root element.

c. The pseudo-code algorithm appears as shown in Figure 12.4.

d. The structure will lose balance quickly, degrading to near linear performance. One possible solution to this problem would be to use a balanced tree structure such as a red-black or AVL tree.

12.6 Write a Java program that simulates the disk-scheduling algorithms discussed in Section 12.4. In particular, design separate classes that implement the following scheduling algorithms:

```
SSTF_init(int actualPosition) {
   binaryTree T = new BinaryTree();
}

SSTF_insert_request(int request) {
   T.insert(request);
}

SSTF_next_task() {
   pred = T.root.predecessor();
   succ = T.root.successor();
   if (abs(root - pred) < abs(root - succ)) {
      // this services the predecessor
      T.remove(pred);
      T.root = pred;
   }
   else {
      // this services the successor
      T.remove(succ);
      T.root = succ;
   }
}

SSTF_schedule() {
   SSTF_init(0);

   while (true) {
      if (T.size() > 1)
         next = SSTF_next_task();
   }
}
```

Figure 12.4 Figure 12.4

a. FCFS

b. SSTF

c. SCAN

d. C-SCAN

e. LOOK

Each algorithm will implement the following interface:

```
public interface DiskScheduler
{
   // service the requests
   // return the amount of head movement
   // for the particular algorithm
   public int serviceRequests();
}
```

The `serviceRequests()` method will return the amount of head movement required by the disk-scheduling algorithm.

Reference strings consisting of request for disk cylinders will be provided by the `Generator` class, which is available online (www.os-book.com) The `Generator` class produces random requests for cylinders numbered between 0 and 99. The API for the `Generator` class appears as follows:

```
// produce a default-sized list of cylinder requests
public Generator()
// produce a list of cylinder requests of size count
public Generator(int count)
// return the list of cylinder requests
public int[] getCylinders()
```

Each algorithm implementing the `DiskScheduler` interface must supply a constructor that is passed (1) an integer array of cylinder requests and (2) the initial cylinder position of the disk head. Assuming the `FCFS` class implements `DiskScheduler` according to the FCFS policy, an example illustrating its usage is shown below:

```
Generator ref = new Generator(1000);

int[] referenceString = ref.getCylinders();
DiskScheduler fcfs = new FCFS(referenceString, 13);
System.out.println("FCFS = " + fcfs.serviceRequests());
```

This example constructs 1,000 random cylinder requests and begins the FCFS algorithm at cylinder 13.

When you have finished, compare the amounts of head movement required by the various disk-scheduling algorithms.

Answer: As the cylinder requests are random, answers will vary widely.

12.7 Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request's service), the variation in response time, and the effective bandwidth. How does performance depend on the relative sizes of seek time and rotational latency?

Answer: There is no simple analytical argument to answer the first part of this question. It would make a good small simulation experiment for the students. The answer can be found in Figure 2 of Worthington et al. [1994]. (Worthington et al. studied the LOOK algorithm, but similar results obtain for SCAN.) Figure 2 in Worthington et al. shows that C-LOOK has an average response time just a few percent higher than LOOK but that C-LOOK has a significantly lower variance in response time for medium and heavy workloads. The intuitive reason for the difference in variance is that LOOK (and SCAN) tend to favor requests near the middle cylinders, whereas the C-versions do not have this imbalance. The intuitive reason for the slower response time of C-LOOK is the "circular" seek from one end of the disk to the farthest request at the other end. This seek satisfies no requests. It causes only a small performance degradation because the square-root dependency of seek time on distance implies that a long seek isn't terribly expensive by comparison with moderate-length seeks.

For the second part of the question, we observe that these algorithms do not schedule to improve rotational latency; therefore, as seek times decrease relative to rotational latency, the performance differences between the algorithms will decrease.

12.8 Requests are not usually uniformly distributed. For example, a cylinder containing the file system FAT or inodes can be expected to be accessed more frequently than a cylinder that contains only files. Suppose you know that 50 percent of the requests are for a small, fixed number of cylinders.

a. Would any of the scheduling algorithms discussed in this chapter be particularly good for this case? Explain your answer.

b. Propose a disk-scheduling algorithm that gives even better performance by taking advantage of this "hot spot" on the disk.

c. File systems typically find data blocks via an indirection table, such as a FAT in DOS or inodes in UNIX. Describe one or more ways to take advantage of this indirection to improve the disk performance.

Answer:

a. SSTF would take greatest advantage of the situation. FCFS could cause unnecessary head movement if references to the "high-demand" cylinders were interspersed with references to cylinders far away.

b. Here are some ideas. Place the hot data near the middle of the disk. Modify SSTF to prevent starvation. Add the policy that if

the disk becomes idle for more than, say, 50 ms, the operating system generates an *anticipatory seek* to the hot region, since the next request is more likely to be there.

c. Cache the metadata in primary memory, and locate a file's data and metadata in close physical proximity on the disk. (UNIX accomplishes the latter goal by allocating data and metadata in regions called *cylinder groups*.)

12.9 Could a RAID Level 1 organization achieve better performance for read requests than a RAID Level 0 organization (with nonredundant striping of data)? If so, how?

Answer: Yes, a RAID Level 1 organization could achieve better performance for read requests. When a read operation is performed, a RAID Level 1 system can decide which of the two copies of the block should be accessed to satisfy the request. This choice could be based on the current location of the disk head and could therefore result in performance optimizations by choosing a disk head that is closer to the target data.

12.10 Consider a RAID Level 5 organization comprising five disks, with the parity for sets of four blocks on four disks stored on the fifth disk. How many blocks are accessed in order to perform the following?

a. A write of one block of data

b. A write of seven continuous blocks of data

Answer: a) A write of one block of data requires the following: read of the parity block, read of the old data stored in the target block, computation of the new parity based on the differences between the new and old contents of the target block, and write of the parity block and the target block. b) Assume that the seven contiguous blocks begin at a four-block boundary. A write of seven contiguous blocks of data could be performed by writing the seven contiguous blocks, writing the parity block of the first four blocks, reading the eight block, computing the parity for the next set of four blocks and writing the corresponding parity block onto disk.

12.11 Compare the throughput achieved by a RAID Level 5 organization with that achieved by a RAID Level 1 organization for the following:

a. Read operations on single blocks

b. Read operations on multiple contiguous blocks

Answer: a) The amount of throughput depends on the number of disks in the RAID system. A RAID Level 5 comprising of a parity block for every set of four blocks spread over five disks can support four to five operations simultaneously. A RAID Level 1 comprising of two disks can support two simultaneous operations. Of course, there is greater flexibility in RAID Level 1 as to which copy of a block could be accessed and that could provide performance benefits by taking into account position of disk head. b) RAID Level 5 organization achieves greater bandwidth for accesses to multiple contiguous blocks since the

adjacent blocks could be simultaneously accessed. Such bandwidth improvements are not possible in RAID Level 1.

12.12 Compare the performance of write operations achieved by a RAID Level 5 organization with that achieved by a RAID Level 1 organization.
Answer: RAID Level 1 organization can perform writes by simply issuing the writes to mirrored data concurrently. RAID Level 5, on the other hand, would require the old contents of the parity block to be read before it is updated based on the new contents of the target block. This results in more overhead for the write operations on a RAID Level 5 system.

12.13 Assume that you have a mixed configuration comprising disks organized as RAID Level 1 and as RAID Level 5 disks. Assume that the system has flexibility in deciding which disk organization to use for storing a particular file. Which files should be stored in the RAID Level 1 disks and which in the RAID Level 5 disks in order to optimize performance?
Answer: Frequently updated data need to be stored on RAID Level 1 disks while data that is more frequently read as opposed to being written should be stored in RAID Level 5 disks.

12.14 Is there any way to implement truly stable storage? Explain your answer.
Answer: Truly stable storage would never lose data. The fundamental technique for stable storage is to maintain multiple copies of the data, so that if one copy is destroyed, some other copy is still available for use. But for any scheme, we can imagine a large enough disaster that all copies are destroyed.

12.15 The reliability of a hard-disk drive is typically described in terms of a quantity called *mean time between failures* (*MTBF*). Although this quantity is called a "time," the MTBF actually is measured in drive-hours per failure.

a. If a disk farm contains 1000 drives, each of which has a 750,000-hour MTBF, which of the following best describes how often a drive failure will occur in that disk farm: once per thousand years, once per century, once per decade, once per year, once per month, once per week, once per day, once per hour, once per minute, or once per second?

b. Mortality statistics indicate that, on the average, a U.S. resident has about 1 chance in 1000 of dying between ages 20 and 21 years. Deduce the MTBF hours for 20 year olds. Convert this figure from hours to years. What does this MTBF tell you about the expected lifetime of a 20 year old?

c. The manufacturer claims a 1-million hour MTBF for a certain model of disk drive. What can you say about the number of years that one of those drives can be expected to last?

Answer:

a. 750,000 drive-hours per failure divided by 1000 drives gives 750 hours per failure—about 31 days or once per month.

b. The human-hours per failure is 8760 (hours in a year) divided by 0.001 failure, giving a value of 8,760,000 "hours" for the MTBF. 8760,000 hours equals 1000 years. This tells us nothing about the expected lifetime of a person of age 20.

c. The MTBF tells nothing about the expected lifetime. Hard disk drives are generally designed to have a lifetime of five years. If such a drive truly has a million-hour MTBF, it is very unlikely that the drive will fail during its expected lifetime.

12.16 Discuss the relative advantages and disadvantages of sector sparing and sector slipping.
Answer:
Sector sparing can cause an extra track switch and rotational latency, causing an unlucky request to require an extra 8 ms of time. Sector slipping has less impact during future reading, but at sector remapping time it can require the reading and writing of an entire track's worth of data to slip the sectors past the bad spot.

12.17 Discuss the reasons why the operating system might require accurate information on how blocks are stored on a disk. How could the operating system improve file system performance with this knowledge?
Answer: While allocating blocks for a file, the operating system could allocate blocks that are geometrically close by on the disk if it had more information regarding the physical location of the blocks on the disk. In particular, it could allocate a block of data and then allocate the second block of data in the same cylinder but on a different surface at a rotationally optimal place so that the access to the next block could be made with minimal cost.

12.18 The operating system generally treats removable disks as shared file systems but assigns a tape drive to only one application at a time. Give three reasons that could explain this difference in treatment of disks and tapes. Describe additional features that would be required of the operating system to support shared file-system access to a tape jukebox. Would the applications sharing the tape jukebox need any special properties, or could they use the files as though the files were disk-resident? Explain your answer.
Answer:

a. Disks have fast random-access times, so they give good performance for interleaved access streams. By contrast, tapes have high positioning times. Consequently, if two users attempt to share a tape drive for reading, the drive will spend most of its time switching tapes and positioning to the desired data, and relatively little time performing data transfers. This performance problem is similar to the thrashing of a virtual memory system that has insufficient physical memory.

b. Tape cartridges are removable. The owner of the data may wish to store the cartridge off-site (far away from the computer) to keep a copy of the data safe from a fire at the location of the computer.

c. Tape cartridges are often used to send large volumes of data from a producer of data to the consumer. Such a tape cartridge is reserved for that particular data transfer and cannot be used for general-purpose shared storage space.

To support shared file-system access to a tape jukebox, the operating system would need to perform the usual file-system duties, including

- Manage a file-system name space over the collection of tapes
- Perform space allocation
- Schedule the I/O operations

The applications that access a tape-resident file system would need to be tolerant of lengthy delays. For improved performance, it would be desirable for the applications to be able to disclose a large number of I/O operations so that the tape-scheduling algorithms could generate efficient schedules.

12.19 What would be the effect on cost and performance if tape storage were to achieve the same areal density as disk storage? (**Areal density** is the number of gigabits per square inch.)
Answer: To achieve the same areal density as a magnetic disk, the areal density of a tape would need to improve by two orders of magnitude. This would cause tape storage to be much cheaper than disk storage. The storage capacity of a tape would increase to more than 1 terabyte, which could enable a single tape to replace a jukebox of tapes in today's technology, further reducing the cost. The areal density has no direct bearing on the data transfer rate, but the higher capacity per tape might reduce the overhead of tape switching.

12.20 You can use simple estimates to compare the cost and performance of a terabyte storage system made entirely from disks with one that incorporates tertiary storage. Suppose that magnetic disks each hold 10 gigabytes, cost $1000, transfer 5 megabytes per second, and have an average access latency of 15 milliseconds. Suppose that a tape library costs $10 per gigabyte, transfers 10 megabytes per second, and has an average access latency of 20 seconds. Compute the total cost, the maximum total data rate, and the average waiting time for a pure disk system. If you make any assumptions about the workload, describe and justify them. Now, suppose that 5 percent of the data are frequently used, so they must reside on disk, but the other 95 percent are archived in the tape library. Further suppose that 95 percent of the requests are handled by the disk system and the other 5 percent are handled by the library. What are the total cost, the maximum total data rate, and the average waiting time for this hierarchical storage system?
Answer: First let's consider the pure disk system. One terabyte is 1024 GB. To be correct, we need 103 disks at 10 GB each. But since this question

is about approximations, we will simplify the arithmetic by rounding off the numbers. The pure disk system will have 100 drives. The cost of the disk drives would be $100,000, plus about 20% for cables, power supplies, and enclosures; that is, around $120,000. The aggregate data rate would be 100 × 5 MB/s, or 500 MB/s. The average waiting time depends on the workload. Suppose that the requests are for transfers of size 8 KB, and suppose that the requests are randomly distributed over the disk drives. If the system is lightly loaded, a typical request will arrive at an idle disk, so the response time will be 15 ms access time plus about 2 ms transfer time. If the system is heavily loaded, the delay will increase, roughly in proportion to the queue length.

Now let's consider the hierarchical storage system. The total disk space required is 5% of 1 TB, which is 50 GB. Consequently, we need 5 disks, so the cost of the disk storage is $5,000 (plus 20%; that is, $6,000). The cost of the 950 GB tape library is $9500. Thus the total storage cost is $15,500. The maximum total data rate depends on the number of drives in the tape library. We suppose there is only one drive. Then the aggregate data rate is 6 × 10 MB/s; that is, 60 MB/s. For a lightly loaded system, 95% of the requests will be satisfied by the disks with a delay of about 17 ms. The other 5% of the requests will be satisfied by the tape library, with a delay of slightly more than 20 seconds. Thus the average delay will be $(95 \times 0.017 + 5 \times 20)/100$, or about 1 second. Even with an empty request queue at the tape library, the latency of the tape drive is responsible for almost all of the system's response latency, because 1/20th of the workload is sent to a device that has a 20-second latency. If the system is more heavily loaded, the average delay will increase in proportion to the length of the queue of requests waiting for service from the tape drive.

The hierarchical system is much cheaper. For the 95% of the requests that are served by the disks, the performance is as good as a pure-disk system. But the maximum data rate of the hierarchical system is much worse than for the pure-disk system, as is the average response time.

12.21 Imagine that a holographic storage drive has been invented. Suppose that a holographic drive costs $10,000 and has an average access time of 40 milliseconds. Suppose that it uses a $100 cartridge the size of a CD. This cartridge holds 40,000 images, and each image is a square black-and-white picture with resolution 6000 × 6000 pixels (each pixel stores 1 bit). Suppose that the drive can read or write 1 picture in 1 millisecond. Answer the following questions.

a. What would be some good uses for this device?

b. How would this device affect the I/O performance of a computing system?

c. Which other kinds of storage devices, if any, would become obsolete as a result of this device being invented?

Answer: First, calculate performance of the device. 6000 × 6000 bits per millisecond = 4394 KB per millisecond = 4291 MB/sec(!). Clearly this is orders of magnitude greater than current hard disk technology, as

the best production hard drives do less than 40 MB/sec. The following answers assume that the device cannot store data in smaller chunks than 4 MB.

a. This device would find great demand in the storage of images, audio files, and other digital media.

b. Assuming that interconnection speed to this device would equal its throughput ability (that is, the other components of the system could keep it fed), large-scale digital load and store performance would be greatly enchanced. Manipulation time of the digital object would stay the same, of course. The result would be greatly enhanced overall performance.

c. Currently, objects of that size are stored on optical media, tape media, and disk media. Presumably, demand for those would decrease as the holographic storage became available. There are likely to be uses for all of those media even in the presence of holographic storage, so it is unlikely that any would become obsolete. Hard disks would still be used for random access to smaller items (such as user files). Tapes would still be used for off-site archiving, and disaster-recovery uses, and optical disks (CDRW for instance) for easy interchange with other computers, and low-cost bulk storage.

   Depending on the size of the holographic device, and its power requirements, it would also find use in replacing solid-state memory for digital cameras, MP3 players, and hand-held computers.

12.22 Suppose that a one-sided 5.25-inch optical-disk cartridge has an areal density of 1 gigabit per square inch. Suppose that a magnetic tape has an areal density of 20 megabits per square inch, and is 1/2 inch wide and 1800 feet long. Calculate an estimate of the storage capacities of these two kinds of storage cartridges. Suppose that an optical tape exists that has the same physical size as the tape, but the same storage density as the optical disk. What volume of data could the optical tape hold? What would be a marketable price for the optical tape if the magnetic tape cost $25?

Answer: The area of a 5.25-inch disk is about 19.625 square inches. If we suppose that the diameter of the spindle hub is 1.5 inches, the hub occupies an area of about 1.77 square inches, leaving 17.86 square inches for data storage. Therefore, we estimate the storage capacity of the optical disk to be 2.2 gigabytes.

The surface area of the tape is 10,800 square inches, so its storage capacity is about 26 gigabytes.

If the 10,800 square inches of tape had a storage density of 1 gigabit per square inch, the capacity of the tape would be about 1,350 gigabytes, or 1.3 terabytes. If we charge the same price per gigabyte for the optical tape as for magnetic tape, the optical tape cartridge would cost about 50 times more than the magnetic tape; that is, $1,250.

12.23 Discuss how an operating system could maintain a free-space list for a tape-resident file system. Assume that the tape technology is append-only, and that it uses the EOT mark and `locate`, `space`, and `read` position commands as described in Section 12.9.2.1.

Answer: Since this tape technology is append-only, all the free space is at the end of the tape. The location of this free space does not need to be stored at all, because the `space` command can be used to position to the EOT mark. The amount of available free space after the EOT mark can be represented by a single number. It may be desirable to maintain a second number to represent the amount of space occupied by files that have been logically deleted (but their space has not been reclaimed since the tape is append-only) so that we can decide when it would pay to copy the nondeleted files to a new tape in order to reclaim the old tape for reuse. We can store the free and deleted space numbers on disk for easy access. Another copy of these numbers can be stored at the end of the tape as the last data block. We can overwrite this last data block when we allocate new storage on the tape.

I/O Systems

The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices. Although related topics appear in other chapters, here we bring together the pieces to paint a complete picture. In this chapter we describe I/O structure, devices, device drivers, caching, and terminal I/O.

Exercises

13.1 When multiple interrupts from different devices appear at about the same time, a priority scheme could be used to determine the order in which the interrupts would be serviced. Discuss what issues need to be considered in assigning priorities to different interrupts.

Answer: A number of issues need to be considered in order to determine the priority scheme to be used to determine the order in which the interrupts need to be serviced. First, interrupts raised by devices should be given higher priority than traps generated by the user program; a device interrupt can therefore interrupt code used for handling system calls. Second, interrupts that control devices might be given higher priority than interrupts that simply perform tasks such as copying data served up a device to user/kernel buffers, since such tasks can always be delayed. Third, devices that have real-time constraints on when their data is handled should be given higher priority than other devices. Also, devices that do not have any form of buffering for its data would have to be assigned higher priority since the data could be available only for a short period of time.

13.2 What are the advantages and disadvantages of supporting memory-mapped I/O to device-control registers?

Answer: The advantage of supporting memory-mapped I/O to device-control registers is that it eliminates the need for special I/O instructions from the instruction set and therefore also does not require the enforcement of protection rules that prevent user programs from executing these I/O instructions. The disadvantage is that the resulting flexibility needs to be handled with care; the memory translation units need to ensure that the memory addresses associated with the device con-

trol registers are not accessible by user programs in order to ensure protection.

13.3 Consider the following I/O scenarios on a single-user PC.

a. A mouse used with a graphical user interface

b. A tape drive on a multitasking operating system (assume no device preallocation is available)

c. A disk drive containing user files

d. A graphics card with direct bus connection, accessible through memory-mapped I/O

For each of these I/O scenarios, would you design the operating system to use buffering, spooling, caching, or a combination? Would you use polled I/O, or interrupt-driven I/O? Give reasons for your choices.
Answer:

a. A mouse used with a graphical user interface
Buffering may be needed to record mouse movement during times when higher-priority operations are taking place. Spooling and caching are inappropriate. Interrupt-driven I/O is most appropriate.

b. A tape drive on a multitasking operating system (assume no device preallocation is available)
Buffering may be needed to manage throughput difference between the tape drive and the source or destination of the I/O. Caching can be used to hold copies of data that resides on the tape, for faster access. Spooling could be used to stage data to the device when multiple users desire to read from or write to it. Interrupt-driven I/O is likely to allow the best performance.

c. A disk drive containing user files
Buffering can be used to hold data while in transit from user space to the disk, and visa versa. Caching can be used to hold disk-resident data for improved performance. Spooling is not necessary because disks are shared-access devices. Interrupt-driven I/O is best for devices such as disks that transfer data at slow rates.

d. A graphics card with direct bus connection, accessible through memory-mapped I/O
Buffering may be needed to control multiple access and for performance (double-buffering can be used to hold the next screen image while displaying the current one). Caching and spooling are not necessary due to the fast and shared-access natures of the device. Polling and interrupts are useful only for input and for I/O completion detection, neither of which is needed for a memory-mapped device.

13.4 In most multiprogrammed systems, user programs access memory through virtual addresses, while the operating system uses raw physi-

cal addresses to access memory. What are the implications of this design on the initiation of I/O operations by the user program and their execution by the operating system?

Answer: The user program typically specifies a buffer for data to be transmitted to or from a device. This buffer exists in user space and is specified by a virtual address. The kernel needs to issue the I/O operation and needs to copy data between the user buffer and its own kernel buffer before or after the I/O operation. In order to access the user buffer, the kernel needs to translate the virtual address provided by the user program to the corresponding physical address within the context of the user program's virtual address space. This translation is typically performed in software and therefore incurs overhead. Also, if the user buffer is not currently present in physical memory, the corresponding page(s) need to obtained from the swap space. This operation might require careful handling and might delay the data copy operation.

13.5 What are the various kinds of performance overheads associated with servicing an interrupt?

Answer: When an interrupt occurs the currently executing process is interrupted and its state is stored in the appropriate process control block. The interrupt service routine is then dispatched in order to deal with the interrupt. On completion of handling of the interrupt, the state of the process is restored and the process is resumed. Therefore, the performance overheads include the cost of saving and restoring process state and the cost of flushing the instruction pipeline and restoring the instructions into the pipeline when the process is restarted.

13.6 Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their device is ready?

Answer: Generally, blocking I/O is appropriate when the process will be waiting only for one specific event. Examples include a disk, tape, or keyboard read by an application program. Non-blocking I/O is useful when I/O may come from more than one source and the order of the I/O arrival is not predetermined. Examples include network daemons listening to more than one network socket, window managers that accept mouse movement as well as keyboard input, and I/O-management programs, such as a copy command that copies data between I/O devices. In the last case, the program could optimize its performance by buffering the input and output and using non-blocking I/O to keep both devices fully occupied.

Non-blocking I/O is more complicated for programmers, because of the asynchronous rendezvous that is needed when an I/O occurs. Also, busy waiting is less efficient than interrupt-driven I/O so the overall system performance would decrease.

13.7 Typically, at the completion of a device I/O, a single interrupt is raised and appropriately handled by the host processor. In certain settings, however, the code that is to be executed at the completion of the I/O can be broken into two separate pieces, one of which executes imme-

diately after the I/O completes and schedules a second interrupt for the remaining piece of code to be executed at a later time. What is the purpose of using this strategy in the design of interrupt handlers?
Answer: The purpose of this strategy is to ensure that the most critical aspect of the interrupt handling code is performed first and the less critical portions of the code are delayed for the future. For instance, when a device finishes an I/O operation, the device-control operations corresponding to declaring the device as no longer being busy are more important in order to issue future operations. However, the task of copying the data provided by the device to the appropriate user or kernel memory regions can be delayed for a future point when the CPU is idle. In such a scenario, a lower-priority interrupt handler is used to perform the copy operation.

13.8 Some DMA controllers support direct virtual memory access, where the targets of I/O operations are specified as virtual addresses and a translation from virtual to physical address is performed during the DMA. How does this design complicate the design of the DMA controller? What are the advantages of providing such a functionality?
Answer: Direct virtual memory access allows a device to perform a transfer from two memory-mapped devices without the intervention of the CPU or the use of main memory as a staging ground; the device simply issues memory operations to the memory-mapped addresses of a target device and the ensuing virtual address translation guarantees that the data is transferred to the appropriate device. This functionality, however, comes at the cost of having to support virtual address translation on addresses accessed by a DMA controller and requires the addition of an address-translation unit to the DMA controller. The address translation results in both hardware and software costs and might also result in coherence problems between the data structures maintained by the CPU for address translation and corresponding structures used by the DMA controller. These coherence issues would also need to be dealt with and results in further increase in system complexity.

13.9 UNIX coordinates the activities of the kernel I/O components by manipulating shared in-kernel data structures, whereas Windows NT uses object-oriented message passing between kernel I/O components. Discuss three pros and three cons of each approach.
Answer:
Three pros of the UNIX method: Very efficient, low overhead and low amount of data movement. Fast implementation—no coordination needed with other kernel components. Simple, so less chance of data loss

Three cons: No data protection, and more possible side effects from changes so more difficult to debug. Difficult to implement new I/O methods: new data structures needed rather than just new objects. Complicated kernel I/O subsystem, full of data structures, access routines, and locking mechanisms

13.10 Write (in pseudocode) an implementation of virtual clocks, including the queuing and management of timer requests for the kernel and applications. Assume that the hardware provides three timer channels.
Answer: Each channel would run the following algorithm:

```
        /** data definitions **/

// a list of interrupts sorted in earliest-time-first order
List interruptList

// the list that associates a request with an entry in
interruptList
List requestList

// an interrupt-based timer
Timer timer

while (true) {
    /** Get the next earliest time in the list **/
    timer.setTime = interruptList.next();

    /** An interrupt will occur at time timer.setTime **/

    /** now wait for the timer interrupt
    i.e. for the timer to expire **/

    notify( requestList.next() );
}
```

13.11 Discuss the advantages and disadvantages of guaranteeing reliable transfer of data between modules in the STREAMS abstraction.
Answer: Reliable transfer of data requires modules to check whether space is available on the target module and to block the sending module if space is not available. This check requires extra communication between the modules, but the overhead enables the system to provide a stronger abstraction than one which does not guarantee reliable transfer. The stronger abstraction typically reduces the complexity of the code in the modules. In the STREAMS abstraction, however, there is unreliability introduced by the driver end, which is allowed to drop messages if the corresponding device cannot handle the incoming data. Consequently, even if there is reliable transfer of data between the modules, messages could be dropped at the device if the corresponding buffer fills up. This requires retransmission of data and special code for handling such retransmissions, thereby somewhat limiting the advantages that are associated with reliable transfer between the modules.

Protection

The various processes in an operating system must be protected from one another's activities. For that purpose, various mechanisms exist that can be used to ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.

In this chapter, we examine the problem of protection in great detail and develop a unifying model for implementing protection.

It is important that the student learn the concepts of the access matrix, access lists, and capabilities. Capabilities have been used in several modern systems and can be tied in with abstract data types. The paper by Lampson [1971] is the classic reference on protection.

Exercises

14.1 Consider the ring protection scheme in MULTICS. If we were to implement the system calls of a typical operating system and store them in a segment associated with ring 0, what should be the values stored in the ring field of the segment descriptor? What happens during a system call when a process executing in a higher-numbered ring invokes a procedure in ring 0?

Answer: The ring should be associated with an access bracket (b1, b2), a limit value b3, and a set of designated entry points. The processes that are allowed to invoke any code stored in segment 0 in an unconstrained manner are those processes that are currently executing in ring i where $b1 \leq i \leq b2$. Any other process executing within ring $b2 < i \leq b3$ is allowed to invoke only those functions that are designated entry points. This implies that we should have $b1 = 0$ and set $b2$ to be the highest ring number that comprises of system code that is allowed to invoke the code in segment 0 in an unconstrained fashion. We should also store only the system call functions as designated entry points and we should set $b3$ to be the ring number associated with user code so that user code can invoke the system calls.

14.2 The access-control matrix could be used to determine whether a process can switch from, say, domain A to domain B and enjoy the access privileges of domain B. Is this approach equivalent to including the access privileges of domain B in those of domain A?
Answer: Yes, this approach is equivalent to including the access privileges of domain B in those of domain A as long as the switch privileges associated with domain B are also copied over to domain A.

14.3 Consider a system in which "computer games" can be played by students only between 10 P.M. and 6 A.M., by faculty members between 5 P.M. and 8 A.M., and by the computer center staff at all times. Suggest a scheme for implementing this policy efficiently.
Answer: Set up a dynamic protection structure that changes the set of resources available with respect to the time allotted to the three categories of users. As time changes, so does the domain of users eligible to play the computer games. When the time comes that a user's eligibility is over, a revocation process must occur. Revocation could be immediate, selective (since the computer staff may access it at any hour), total, and temporary (since rights to access will be given back later in the day).

14.4 What hardware features are needed for efficient capability manipulation? Can these be used for memory protection?
Answer: A hardware feature is needed allowing a capability object to be identified as either a capability of accessible object. Typically, several bits are necessary to distinguish between different types of capability objects. For example, 4 bits could be used to uniquely identify 2^4 or 16 different types of capability objects.
These could not be used for routine memory protection as they offer little else for protection apart from a binary value indicating whether they are a capability object or not. Memory protection requires full support from virtual memory features discussed in Chapter 9.

14.5 Discuss the strengths and weaknesses of implementing an access matrix using access lists that are associated with objects.
Answer: The strength of storing an access list with each object is the control that comes from storing the access privileges along with each object, thereby allowing the object to revoke or expand the access privileges in a localized manner. The weakness with associating access lists is the overhead of checking whether the requesting domain appears on the access list. This check would be expensive and needs to be performed every time the object is accessed.

14.6 Discuss the strengths and weaknesses of implementing an access matrix using capabilities that are associated with domains.
Answer: Capabilities associated with domains provide substantial flexibility and faster access to objects. When a domain presents a capability, the system just needs to check the authenticity of the capability and that could be performed efficiently. Capabilities could also be passed around from one domain to another domain with great ease, allowing a system with a great amount of flexibility. However, the flex-

ibility comes at the cost of a lack of control: revoking capabilities and restricting the flow of capabilities is a difficult task.

14.7 Explain why a capability-based system such as Hydra provides greater flexibility than the ring-protection scheme in enforcing protection policies.

Answer: The ring-based protection scheme requires the modules to be ordered in a strictly hierarchical fashion. It also enforces the restriction that system code in internal rings cannot invoke operations in the external rings. This restriction limits the flexibility in structuring the code and is unnecessarily restrictive. The capability system provided by Hydra not only allows unstructured interactions between different modules, but also enables the dynamic instantiation of new modules as the need arises.

14.8 Discuss the need for rights amplification in Hydra. How does this practice compare with the cross-ring calls in a ring protection scheme?

Answer: Rights amplification is required to deal with cross-domain calls where code in the calling domain does not have the access privileges to perform certain operations on an object but the called procedure has an expanded set of access privileges on the same object. Typically, when an object is created by a module, if the module wants to export the object to other modules without granting the other modules privileges to modify the object, it could export the object with those kinds of access privileges disabled. When the object is passed back to the module that created it in order to perform some mutations on it, the rights associated with the object need to be expanded. A more coarse-grained approach to rights amplification is employed in Multics. When a cross-ring call occurs, a set of checks are made to ensure that the calling code has sufficient rights to invoke the target code. Assuming that the checks are satisfied, the target code is invoked and the ring number associated with the process is modified to be ring number associated with the target code, thereby expanding the access rights associated with the process.

14.9 What is the need-to-know principle? Why is it important for a protection system to adhere to this principle?

Answer: A process may access at any time those resources that it has been authorized to access *and* are required currently to complete its task. It is important in that it limits the amount of damage a faulty process can cause in a system.

14.10 Discuss which of the following systems allow module designers to enforce the need-to-know principle.

a. The MULTICS ring protection scheme

b. Hydra's capabilities

c. JVM's stack-inspection scheme

Answer: The ring protection scheme in MULTICS does not necessarily enforce the need-to-know principle. If an object must be accessible in a domain at ring level j but not accessible in a domain at ring level i,

then we must have $j < i$. But this requirement means that every object accessible in level i must also be accessible in level j.
A similar problem arises in JVM's stack inspection scheme. When a sequence of calls is made within a doPrivileged code block, then all of the code fragments in the called procedure have the same access privileges as the original code block that performed the doPrivileged operation, thereby violating the need-to-know principle.
In Hydra, the rights-amplification mechanism ensures that only the privileged code has access privileges to protected objects, and if this code were to invoke code in other modules, the objects could be exported to the other modules after lowering the access privileges to the exported objects. This mechanism provides fine-grained control over access rights and helps to guarantee that the need-to-know principle is satisfied.

14.11 Describe how the Java protection model would be sacrificed if a Java program were allowed to directly alter the annotations of its stack frame.
Answer: When a Java thread issues an access request in a `doPrivileged()` block, the stack frame of the calling thread is *annotated* according to the calling thread's protection domain. A thread with an annotated stack frame can make subsequent method calls that require certain privileges. Thus, the annotation serves to mark a calling thread as being privileged. By allowing a Java program to directly alter the annotations of a stack frame, a program could potentially perform an operation for which it does not have the necessary permissions, thus violating the security model of Java.

14.12 How are the access-matrix facility and the role-based access-control facility similar? How do they differ?
Answer: The roles in a role-based access control facility are similar to the domain in the access-matrix facility. Just as a domain is granted access to certain resources, a role is also granted access to the appropriate resources. The two approaches differ in the amount of flexibility and the kind of access privileges that are granted to the entities. Certain access-control facilities allow modules to perform a *switch* operation that allows them to assume the privileges of a different module, and this operation can be performed in a transparent manner. Such switches are less transparent in role-based systems where the ability to switch roles is not a privilege that is granted through a mechanism that is part of the access-control system, but instead requires the explicit use of passwords.

14.13 How does the principle of least privilege aid in the creation of protection systems?
Answer: The principle of least privilege allows users to be given just enough privileges to perform their tasks. A system implemented within the framework of this principle has the property that a failure or compromise of a component does the minimum damage to the system since the failed or compromised component has the least set of privileges required to support its normal mode of operation.

14.14 How can systems that implement the principle of least privilege still have protection failures that lead to security violations?

Answer: The principle of least privileges only limits the damage but does not prevent the misuse of access privileges associated with a module if the module were to be compromised. For instance, if a system code is given the access privileges to deal with the task of managing tertiary storage, a security loophole in the code would not cause any damage to other parts of the system, but it could still cause protection failures in accessing the tertiary storage.

Security

The information stored in the system (both data and code), as well as the physical resources of the computer system, need to be protected from unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency. In this chapter, we examine the ways in which information may be misused or intentionally made inconsistent. We then present mechanisms to guard against this occurrence.

Exercises

15.1 Buffer-overflow attacks can be avoided by adopting a better programming methodology or by using special hardware support. Discuss these solutions.

Answer: One form of hardware support that guarantees that a buffer-overflow attack does not take place is to prevent the execution of code that is located in the stack segment of a process's address space. Recall that buffer-overflow attacks are performed by overflowing the buffer on a stack frame and overwriting the return address of the function, thereby jumping to another portion of the stack frame that contains malicious executable code, that had been placed there as a result of the buffer overflow. By preventing the execution of code from the stack segment, this problem is eliminated.

Approaches that use a better programming methodology are typically built around the use of bounds-checking to guard against buffer overflows. Buffer overflows do not not occur in languages like Java where every array access is guaranteed to be within bounds through a software check. Such approaches require no hardware support but result in run-time costs associated with performing bounds-checking.

15.2 A password may become known to other users in a variety of ways. Is there a simple method for detecting that such an event has occurred? Explain your answer.

Answer: Whenever a user logs in, the system prints the last time that user was logged on the system.

15.3 The list of all passwords is kept within the operating system. Thus, if a user manages to read this list, password protection is no longer provided. Suggest a scheme that will avoid this problem. (Hint: Use different internal and external representations.)

Answer: Encrypt the passwords internally so that they can only be accessed in coded form. The only person with access or knowledge of decoding should be the system operator.

15.4 What is the purpose of using a "salt" along with the user-provided password? Where should the "salt" be stored, and how should it be used?

Answer: When a user creates a password, the system generates a random number (which is the salt) and appends it to the user-provided password, encrypts the resulting string and stores the encrypted result and the salt in the password file. When a password check is to be made, the password presented by the user is first concatenated with the salt and then encrypted before checking for equality with the stored password. Since the salt is different for different users, a password cracker cannot check a single candidate password, encrypt it, and check it against all of the encrypted passwords simultaneously.

15.5 An experimental addition to UNIX allows a user to connect a **watchdog** program to a file. The watchdog is invoked whenever a program requests access to the file. The watchdog then either grants or denies access to the file. Discuss two pros and two cons of using watchdogs for security.

Answer: The watchdog program becomes the primary security mechanism for file access. Because of this we find its primary benefits and detractions. A benefit of this approach is that you have a centralized mechanism for controlling access to a file—the watchdog program. By ensuring the watchdog program has sufficient security techniques, you are assured of having secure access to the file. However, this is also the primary negative of this approach as well—the watchdog program becomes the bottleneck. If the watchdog program is not properly implemented (that is, it has a security hole), there are no other backup mechanisms for file protection.

15.6 The UNIX program COPS scans a given system for possible security holes and alerts the user to possible problems. What are two potential hazards of using such a system for security? How can these problems be limited or eliminated?

Answer: The COPS program itself could be modified by an intruder to disable some of its features or even to take advantage of its features to create new security flaws. Even if COPS is not cracked, it is possible for an intruder to gain a copy of COPS, study it, and locate security breaches which COPS does not detect. Then that intruder could prey on systems in which the management depends on COPS for security (thinking it is providing security), when all COPS is providing is management complacency. COPS could be stored on a read-only medium or file system to avoid its modification. It could be provided only to bona fide systems

managers to prevent it from falling into the wrong hands. Neither of these is a foolproof solution, however.

15.7 Discuss a means by which managers of systems connected to the Internet could have designed their systems to limit or eliminate the damage done by a worm. What are the drawbacks of making the change that you suggest?
Answer: "Firewalls" can be erected between systems and the Internet. These systems filter the packets moving from one side of them to the other, assuring that only valid packets owned by authorized users are allowed to access the protect systems. Such firewalls usually make use of the systems less convenient (and network connections less efficient).

15.8 Argue for or against the judicial sentence handed down against Robert Morris, Jr., for his creation and execution of the Internet worm discussed in this chapter.
Answer: An argument against the sentence is that it was simply excessive. Furthermore, many have now commented that this worm actually made people more aware of potential vulnerabilities in the public Internet. An argument for the sentence is that this worm cost Internet users significant time and money and—considering its apparent intent—the sentence was appropriate.

We encourage professors to use a case such as this—and the many similar contemporary cases—as a topic for a class debate.

15.9 Make a list of six security concerns for a bank's computer system. For each item on your list, state whether this concern relates to physical, human, or operating-system security.
Answer: In a protected location, well guarded: physical, human.
Network tamperproof: physical, human, operating system.
Modem access eliminated or limited: physical, human.
Unauthorized data transfers prevented or logged: human, operating system.
Backup media protected and guarded: physical, human.
Programmers, data entry personnel, trustworthy: human.

15.10 What are two advantages of encrypting data stored in the computer system?
Answer: Encrypted data are guarded by the operating system's protection facilities, as well as a password that is needed to decrypt them. Two keys are better than one when it comes to security.

15.11 What commonly used computer programs are prone to man-in-the-middle attacks? Discuss solutions for preventing this form of attack.
Answer: Any protocol that requires a sender and a receiver to agree on a session key before they start communicating is prone to the man-in-the-middle attack. For instance, if one were to implement on a secure shell protocol by having the two communicating machines to identify a common session key, and if the protocol messages for exchanging the session key is not protected by the appropriate authentication mechanism, then it is possible for an attacker to manufacture a separate session key and get access to the data being communicated between

the two parties. In particular, if the server is supposed to manufacture the session key, the attacker could obtain the session key from the server, communicate its locally manufactured session key to the client, and thereby convince the client to use the fake session key. When the attacker receives the data from the client, it can decrypt the data, reencrypt it with the original key from the server, and transmit the encrypted data to the server without alerting either the client or the server about the attacker's presence. Such attacks could be avoided by using digital signatures to authenticate messages from the server. If the server could communicate the session key and its identity in a message that is guarded by a digital signature granted by a certifying authority, then the attacker would not be able to forge a session key, and therefore the man-in-the-middle attack could be avoided.

15.12 Compare symmetric and asymmetric encryption schemes, and discuss under what circumstances a distributed system would use one or the other.

Answer: A symmetric encryption scheme allows the same key to be used for encrypting and decrypting messages. An asymmetric scheme requires the use of two different keys for performing the encryption and the corresponding decryption. Asymmetric key cryptographic schemes are based on mathematical foundations that provide guarantees on the intractability of reverse-engineering the encryption scheme, but they are typically much more expensive than symmetric schemes, which do not provide any such theoretical guarantees. Asymmetric schemes are also superior to symmetric schemes since they could be used for other purposes such as authentication, confidentiality, and key distribution.

15.13 Why doesn't $D(k_d, N)(E(k_e, N)(m))$ provide authentication of the sender? To what uses can such an encryption be put?

Answer: $D(k_d, N)(E(k_e, N)(m))$ means that the message is encrypted using the public key and then decrypted using the private key. This scheme is not sufficient to guarantee authentication since any entity can obtain the public keys and therefore could have fabricated the message. However, the only entity that can decrypt the message is the entity that owns the private key, which guarantees that the message is a secret message from the sender to the entity owning the private key; no other entity can decrypt the contents of the message.

15.14 Discuss how the asymmetric encryption algorithm can be used to achieve the following goals.

a. Authentication: receiver knows that only the sender could have generated the message.

b. Secrecy: only the receiver can decrypt the message.

c. Authentication and secrecy: only the receiver can decrypt the message, and the receiver knows that only the sender could have generated the message.

Answer: Let k_e^s be the public key of the sender, k_e^r be the public key of the receiver, k_d^s be the private key of the sender, and k_e^s be the private key

of the receiver. Authentication is performed by having the sender send a message that is encoded using k_d^s. Secrecy is ensured by having the sender encode the message using k_e^r. Both authentication and secrecy are guaranteed by performing double encryption using both k_d^s and k_e^r.

15.15 Consider a system that generates 10 million audit records per day. Also assume that there are on average 10 attacks per day on this system and that each such attack is reflected in 20 records. If the intrusion-detection system has a true-alarm rate of 0.6 and a false-alarm rate of 0.0005, what percentage of alarms generated by the system correspond to real intrusions?

Answer: The probability of occurrence of intrusive records is $10 * 20/10^6 = 0.0002$. Using Bayes' theorem, the probability that an alarm corresponds to a real intrusion is simply $0.0002 * 0.6/(0.0002 * 0.6 + 0.9998 * 0.0005) = 0.193$.

Distributed System Structures

A *distributed system* is a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication networks, such as high-speed buses or telephone lines. In this chapter, we discuss the general structure of distributed systems and the networks that interconnect them. We contrast the main differences in operating-system design between these types of systems and the centralized systems with which we were concerned previously. Detailed discussions are given in Chapters 17 and 18.

Exercises

16.1 What is the difference between computation migration and process migration? Which is easier to implement, and why?

Answer: Process migration is an extreme form of computation migration. In computation migration, an RPC might be sent to a remote processor in order to execute a computation that could be more efficiently executed on the remote node. In process migration, the entire process is transported to the remote node, where the process continues its execution. Since process migration is an extension of computation migration, more issues need to be considered for implementing process migration. In particular, it is always challenging to migrate all of the necessary state to execute the process, and it is sometimes difficult to transport state regarding open files and open devices. Such a high degree of transparency and completeness is not required for computation migration, where it is clear to the programmer that only a certain section of the code is to be executed remotely. programmer.

16.2 Contrast the various network topologies in terms of the following attributes:

a. Reliability

b. Available bandwidth for concurrent communications

c. Installation cost

d. Load balance in routing responsibilities

Answer: A fully connected network provides the most reliable topology since if any of the links go down, it is likely there exists another path to route the message. A partially connected network may suffer from the problem that if a specific link goes down, another path to route a message may not exist. Of the partially-connected topologies, various levels of reliability exist. In a tree-structured topology, if any of the links goes down, there is no guarantee that messages may be routed. A ring topology requires two links to fail for this situation to occur. If a link fails in a star network, the node connected to that link becomes disconnected from the remainder of the network. However, if the central node fails, the entire network becomes unusable.

Regarding available bandwidth for concurrent communications, the fully connected network provides the maximum utility followed by partially connected networks. Tree-structured networks, rings, and star networks have a linear number of network links and therefore have limited capability with regard to performing high-bandwidth concurrent communications. Installation costs follow a similar trend, with fully connected networks requiring a huge investment, and trees, rings, and stars requiring the least investment.

Fully connected networks and ring networks enjoy symmetry in the structure and do not suffer from hot spots. Given random communication patterns, the routing responsibilities are balanced across the different nodes. Trees and stars suffer from hotspots: the central node in the star and the nodes in the upper levels of the tree carry much more traffic than the other nodes in the system and therefore suffer from load imbalances in routing responsibilities.

16.3 Even though the ISO model of networking specifies seven layers of functionality, most computer systems use fewer layers to implement a network. Why do they use fewer layers? What problems could the use of fewer layers cause?

Answer: A certain network layered protocol may achieve the same functionality of the ISO in fewer layers by using one layer to implement functionality provided in two (or possibly more) layers in the ISO model. Other models may decide there is no need for certain layers in the ISO model. For example, the presentation and session layers are absent in the TCP/IP protocol. Another reason may be that certain layers specified in the ISO model do not apply to a certain implementation. Let's use TCP/IP again as an example where no data link or physical layer is specified by the model. The thinking behind TCP/IP is that the functionality behind the data link and physical layers is not pertinent to TCP/IP—it merely assumes some network connection is provided, whether it be Ethernet, wireless, token ring, etc.

A potential problem with implementing fewer layers is that certain functionality may not be provided by features specified in the omitted layers.

16.4 Explain why doubling the speed of the systems on an Ethernet segment may result in decreased network performance. What changes could help solve this problem?

Answer: Faster systems may be able to send more packets in a shorter time. The network would then have more packets traveling on it, resulting in more collisions, and therefore less throughput relative to the number of packets being sent. More networks can be used, with fewer systems per network, to reduce the number of collisions.

16.5 What are the advantages of using dedicated hardware devices for routers and gateways? What are the disadvantages of using these devices compared with using general-purpose computers?
Answer: The advantages are that dedicated hardware devices for routers and gateways are very fast as all their logic is provided in hardware (firmware.) Using a general-purpose computer for a router or gateway means that routing functionality is provided in software—which is not as fast as providing the functionality directly in hardware.

A disadvantage is that routers or gateways as dedicated devices may be more costly than using off-the-shelf components that comprise a modern personal computer.

16.6 In what ways is using a name server better than using static host tables? What problems or complications are associated with name servers? What methods could you use to decrease the amount of traffic name servers generate to satisfy translation requests?
Answer: Name servers require their own protocol, so they add complication to the system. Also, if a name server is down, host information may become unavailable. Backup name servers are required to avoid this problem. Caches can be used to store frequently requested host information to cut down on network traffic.

16.7 Name servers are organized in a hierarchical manner. What is the purpose of using a hierarchical organization?
Answer: Hierarchical structures are easier to maintain since any changes in the identity of name servers require an update only at the next-level name server in the hierarchy. Changes are therefore localized. The downside of this approach, however, is that the name servers at the top level of the hierarchy are likely to suffer from high loads. This problem can be alleviated by replicating the services of the top-level name servers.

16.8 Consider a network layer that senses collisions and retransmits immediately on detection of a collision. What problems could arise with this strategy? How could they be rectified?
Answer: Delegating the retransmission decisions to the network layer might be appropriate in many settings. In a congested system, immediate retransmissions might increase the congestion in the system, resulting in more collisions and lower throughput. Instead, the decision of when to retransmit could be left to the upper layers, which could delay the retransmission by a period of time that is proportional to the current congestion in the system. An exponential backoff strategy is the most commonly used strategy to avoid over-congesting a system.

16.9 The lower layers of the ISO network model provide datagram service, with no delivery guarantees for messages. A transport-layer protocol

such as TCP is used to provide reliability. Discuss the advantages and disadvantages of supporting reliable message delivery at the lowest possible layer.

Answer: Many applications might not require reliable message delivery. For instance, a coded video stream could recover from packet losses by performing interpolations to derive lost data. In fact, in such applications, retransmitted data are of little use since they would arrive much later than the optimal time and not conform to realtime guarantees. For such applications, reliable message delivery at the lowest level is an unnecessary feature and might result in increased message traffic, most of which is useless, thereby resulting in performance degradation. In general, the lowest levels of the networking stack needs to support the minimal amount of functionality required by all applications and leave extra functionality to be implemented at the upper layers.

16.10 What are the implications of using a dynamic routing strategy on application behavior? For what type of applications is it beneficial to use virtual routing instead of dynamic routing?

Answer: Dynamic routing might route different packets through different paths. Consecutive packets might therefore incur different latencies and there could be substantial jitter in the received packets. Also, many protocols, such as TCP, that assume that reordered packets imply dropped packets, would have to be modified to take into account that reordering is a natural phenomenon in the system and does not imply packet losses. Realtime applications such as audio and video transmissions might benefit more from virtual routing since it minimizes jitter and packet reorderings.

16.11 Run the program shown in Figure 16.5 and determine the IP addresses of the following host names:

- www.wiley.com
- www.cs.yale.edu
- www.javasoft.com
- www.westminstercollege.edu
- www.ietf.org

Answer: As of December 2004, the corresponding IP addresses are

- www.wiley.com—`208.215.179.146`
- www.cs.yale.edu—`128.36.229.30`
- www.javasoft.com—`192.18.97.39`
- www.westminstercollege.edu—`146.86.1.2`
- www.ietf.org—`132.151.6.21`

16.12 Consider a distributed system with two sites, A and B. Consider whether site A can distinguish among the following:

a. B goes down.

b. The link between A and B goes down.

c. B is extremely overloaded and its response time is 100 times longer than normal.

What implications does your answer have for recovery in distributed systems?

Answer: One technique would be for B to periodically send a *I-am-up* message to A indicating it is still alive. If A does not receive an *I-am-up* message, it can assume either B—or the network link—is down. Note that an *I-am-up* message does not allow A to distinguish between each type of failure. One technique that allows A to better determine if the network is down is to send an *Are-you-up* message to B using an alternate route. If it receives a reply, it can determine that indeed the network link is down and that B is up.

If we assume that A knows B is up and is reachable (via the *I-am-up* mechanism) and that A has some value *N* that indicates a normal response time, A could monitor the response time from B and compare values to *N*, allowing A to determine if B is overloaded or not.

The implications of both of these techniques are that A could choose another host—say C—in the system if B is either down, unreachable, or overloaded.

16.13 The original HTTP protocol used TCP/IP as the underlying network protocol. For each page, graphic, or applet, a separate TCP session was constructed, used, and torn down. Because of the overhead of building and destroying TCP/IP connections, performance problems resulted from this implementation method. Would using UDP rather than TCP be a good alternative? What other changes could you make to improve HTTP performance?

Answer: Despite the connectionless nature of UDP, it is not a serious alternative to TCP for the HTTP. The problem with UDP is that it is unreliable, documents delivered via the web must be delivered reliably. (This is easy to illustrate—a single packet missing from an image downloaded from the web makes the image unreadable.)

One possibility is to modify how TCP connections are used. Rather than setting up—and breaking down—a TCP connection for every web resource, allow *persistent* connections where a single TCP connection stays open and is used to deliver multiple web resources.

16.14 Of what use is an address-resolution protocol? Why is it better to use such a protocol than to make each host read each packet to determine that packet's destination? Does a token-passing network need such a protocol? Explain your answer.

Answer: An ARP translates general-purpose addresses into hardware interface numbers so the interface can know which packets are for it. Software need not get involved. It is more efficient than passing each packet to the higher layers. Yes, for the same reason.

16.15 What are the advantages and the disadvantages of making the computer network transparent to the user?
Answer: The advantage is that all files are accessed in the same manner. The disadvantage is that the operating system becomes more complex.

Distributed File Systems

Chapter 17 looks at the current major research and development in distributed-file systems (DFS). The purpose of a DFS is to support the same kind of sharing when the files are physically dispersed among the various sites of a distributed system.

We discuss the various ways a distributed file system can be designed and implemented. First, we discuss common concepts on which distributed file systems are based. Then, we illustrate our concepts by examining AFS—the Andrew distributed file system. By exploring this example system, we hope to provide a sense of the considerations involved in designing an operating system, and also to indicate current areas of operating-system research: network and distributed operating systems.

Exercises

17.1 What are the benefits of a DFS when compared to a file system in a centralized system?

Answer: A DFS allows the same type of sharing available on a centralized system, but the sharing may occur on physically and logically separate systems. Users around the world are able to share data as if they were in the same building, allowing a much more flexible computing environment than would otherwise be available.

17.2 Which of the example DFSs discussed in this chapter would handle a large, multiclient database application most efficiently? Explain your answer.

Answer: The Andrew file system can handle a large, multiclient database as scalability is one of its hallmark features. Andrew is designed to handle up to 5,000 client workstations as well. A database also needs to run in a secure environment and Andrew uses the Kerberos security mechanism for encryption.

17.3 Discuss whether AFS and NFS provide the following: (a) location transparency and (b) location independence.

Answer: NFS provides location transparence since one cannot determine the server hosting the file from its name. (One could however view the mount-tables to determine the file server from which the corresponding file system is mounted, but the file server is not hardcoded in the name of the file.) NFS does not provide location independence since files cannot be moved automatically between different file systems. AFS provides location transparence and location independence.

17.4 Under what circumstances would a client prefer a location-transparent DFS? Under which circumstances would she prefer a location-independent DFS? Discuss the reasons for these preferences.
Answer: Location-transparent DFS is good enough in systems in which files are not replicated. Location-independent DFS is necessary when any replication is done.

17.5 What aspects of a distributed system would you select for a system running on a totally reliable network?
Answer: Since the system is totally reliable, a stateful approach would make the most sense. Error recovery would seldom be needed, allowing the features of a stateful system to be used. If the network is very fast as well as reliable, caching can be done on the server side. On a slower network caching on both server and client will speed performance, as would file location-independence and migration. In addition, RPC-based service is not needed in the absence of failures, since a key part of its design is recovery during networking errors. Virtual-circuit systems are simpler and more appropriate for systems with no communications failures.

17.6 Consider AFS, which is a stateful distributed file system. What actions need to be performed to recover from a server crash in order to preserve the consistency guaranteed by the system?
Answer: A server needs to keep track of what clients are currently caching a file in order to issue a callback when the file is modified. When a server goes down, this state is lost. A server would then have to reconstruct this state typically by contacting all of the clients and having them report to the server what files are currently being cached by each client.

17.7 Compare and contrast the techniques of caching disk blocks locally, on a client system, and remotely, on a server.
Answer: Caching locally can reduce network traffic substantially as the local cache can possibly handle a significant number of the remote accesses. This can reduce the amount of network traffic and lessen the load on the server. However, to maintain consistency, local updates to disk blocks must be updated on the server using either a write-through or delayed-write policy. A strategy must also be provided that allows the client to determine if its cached data is stale and needs to be updated. Caching locally provides is obviously more complicated than having a client request all data from the server. But if access patterns indicate heavy writes to the data, the mechanisms for handling inconsistent data may increase network traffic and server load.

17.8 AFS is designed to support a large number of clients. Discuss three techniques used to make AFS a scalable system.
Answer: Three techniques that make AFS a scalable system are:

a. Caching: AFS performs caching of files and name translations, thereby limiting the number of operations sent to the server.

b. Whole-file caching: when a file is opened, the entire contents of the file are transported to the client and no further interactions with the server are required. (This approach is refined in later versions where large chunks of a file rather than the entire file are transported in a single operation.)

c. Callbacks: It is the server's responsibility to revoke outdated copies of files. Clients can cache files and reuse the cached data multiple times without making requests to the server.

17.9 Discuss the advantages and disadvantages of performing path-name translation by having the client ship the entire path to the server requesting a translation for the entire path name of the file.
Answer: The advantage is that a single network request is sufficient to perform the path-name translation. Schemes that perform translations one component at a time incur more network traffic. However, when translations are performed one component at a time, the translations of the parent directories are obtained and cached for future reuse, whereas if the translation of the entire path is performed, translations for none of the intermediate elements are available in the cache.

17.10 What are the benefits of mapping objects into virtual memory, as Apollo Domain does? What are the drawbacks?
Answer: Mapping objects into virtual memory greatly eases the sharing of data between processes. Rather than opening a file, locking access to it, and reading and writing sections via the I/O system calls, memory-mapped objects are accessible as "normal" memory, with reads and writes to locations independent of disk pointers. Locking is much easier also, since one shared memory location can be used as a locking variable for semaphore access. Unfortunately, memory mapping adds complexity to the operating system, especially in a distributed system.

17.11 Describe some of the fundamental differences between AFS and NFS (see Chapter 11).
Answer: Some of the distinctive differences include:

a. AFS has a rich set of features whereas NFS is organized around a much simpler design.

b. NFS allows a workstation to act as either a client, a server, or both. AFS distinguishes between clients and server and identifies dedicated servers.

c. NFS is stateless, meaning a server does not maintain state during client updates of a file. AFS is stateful between the period of when a client opens a file, updates it, and closes the file. (NFS does not even allow the opening and closing of files.)

d. Caching is a fundamental feature of AFS, allowing client-side caching with cache consistency. In fact, it is an architectural principle behind AFS to allow clients to cache entire files. Consistency is provided by servers when cached files are closed. The server then invalidates cached copies existing on other clients. Caching is allowed in NFS as well, but because of its stateless nature modified data must be committed to the server before results are received back by the client.

e. AFS provides session semantics whereas NFS supports UNIX file consistency semantics.

17.12 Discuss whether clients in the following systems can obtain inconsistent or stale data from the file server and, if so, under what scenarios this could occur.

a. AFS

b. Sprite

c. NFS

Answer: Sprite guarantees that clients never see stale data. When a file is opened for write-sharing by multiple clients, all caching for the file is disabled and all operations are sent directly to the server. This scheme guarantees consistency of data. In AFS, writes to files are performed on local disks, and when the client closes the files, the writes are propagated to the server, which then issues callbacks on the various cached copies. During the time the file is written but not closed, the other clients could be accessing stale data. Also, even after the file is closed, the other clients might access stale data if they had performed the open on the file before the updating client had closed it. It is only at the point of time when the next open is performed on a caching client that the server is contacted and the most recent data propagated from the server to the client. NFS uses a more ad-hoc consistency mechanism. Data is flushed from the clients to servers at periodic intervals and on file close operations. Also, a client caching the file data checks for inconsistencies also at periodic intervals. Any updates made and flushed to the server during these intervals are not seen immediately on a client caching the file data.

Distributed Coordination

Chapter 18 examines various mechanisms for process synchronization and communication, as well as methods for dealing with the deadlock problem, in a distributed environment. In addition, since a distributed system may suffer from a variety of failures that are not encountered in a centralized system, we also discuss here the issue of failure in a distributed system.

Exercises

18.1 Discuss the advantages and disadvantages of the two methods we presented for generating globally unique timestamps.

Answer: Globally unique timestamps can be generated using either a centralized or distributed approach. The centralized approach uses a single site for generating timestamps. A disadvantage of this approach is that if this site fails, timestamps can no longer be produced.

Generating timestamps using the distributed approach provides more of a fail-safe mechanism; however, care must be taken to ensure the logical clocks at each site are synchronized.

18.2 The logical clock timestamp scheme presented in this chapter provides the following guarantee: If event A happens before event B, then the timestamp of A is less than the timestamp of B. Note, however, that one cannot order two events based only on their timestamps. The fact that an event C has a timestamp that is less than the timestamp of event D does not necessarily mean that event C happened before event D; C and D could be concurrent events in the system. Discuss ways in which the logical clock timestamp scheme could be extended to distinguish concurrent events from events that can be ordered by the *happens-before* relationship.

Answer: Vector clocks can be used to distinguish concurrent events from events ordered by the happens-before relationship. A vector clock works as follows. Each process maintains a vector timestamp that comprises of a vector of scalar timestamp, where each element reflects the number of events that have occurred in each of the other processes in the system. More formally, a process i maintains a vector timestamp t_i

such that t_i^j is equal to the number of events in process j that have occurred before the current event in process i. When a local event occurs in process i, t_i^i is incremented by one to reflect that one more event has occurred in the process. In addition, any time a message is sent from a process to another process it communicates the timestamp vector of the source process to the destination process, which then updates its local timestamp vector to reflect the newly obtained information. More formally, when s sends a message to d, s communicates t_s along with the message and d updates t_d such that for all $i \neq d$, $t_d^i = max(t_s^i, t_d^i)$.

18.3 Your company is building a computer network, and you are asked to write an algorithm for achieving distributed mutual exclusion. Which scheme will you use? Explain your choice.

Answer: The options are a (1) centralized, (2) fully distributed, or (3) token-passing approach. We reject the centralized approach as the centralized coordinator becomes a bottleneck. We also reject the token-passing approach for its difficulty in re-establishing the ring in case of failure.

We choose the fully distributed approach for the following reasons:

- Mutual exclusion is obtained.
- Freedom from deadlock is ensured.
- Freedom from starvation is ensured, since entry to the critical section is scheduled according to the timestamp ordering.
- The number of messages per critical-section entry is $2 \times (n - 1)$. This number is the minimum number of required messages per critical-section entry when processes act independently and concurrently.

18.4 Why is deadlock detection much more expensive in a distributed environment than it is in a centralized environment?

Answer: The difficulty is that each site must maintain its own local wait-for graph. However, the lack of a cycle in a local graph does not ensure freedom from deadlock. Instead, we can ensure the system is not deadlocked only if the union of **all** local wait-for graphs is acyclic.

18.5 Your company is building a computer network, and you are asked to develop a scheme for dealing with the deadlock problem.

a. Would you use a deadlock-detection scheme or a deadlock-prevention scheme?

b. If you were to use a deadlock-prevention scheme, which one would you use? Explain your choice.

c. If you were to use a deadlock-detection scheme, which one would you use? Explain your choice.

Answer:

a. Would you use a deadlock-detection scheme, or a deadlock-prevention scheme?

We would choose deadlock prevention as it is systematically easier to prevent deadlocks than to detect them once they have occurred.

b. If you were to use a deadlock-prevention scheme, which one would you use? Explain your choice.
A simple resource-ordering scheme would be used: preventing deadlocks by requiring processes to acquire resources in order.

c. If you were to use a deadlock-detection scheme, which one would you use? Explain your choice.
If we were to use a deadlock detection algorithm, we would choose a fully distributed approach as the centralized approach provides for a single point of failure.

18.6 Under what circumstances does the wait–die scheme perform better than the wound–wait scheme for granting resources to concurrently executing transactions?
Answer: In the wound–wait scheme an older process never waits for a younger process; it instead rolls back the younger process and preempts its resources. When a younger process requests a resource held by an older process, it simply waits and there are no rollbacks. In the wait–die scheme, older processes wait for younger processes without any rollbacks but a younger process gets rolled back if it requests a resource held by an older process. This rollback might occur multiple times if the resource is being held by the older process for a long period of time. Repeated rollbacks do not occur in the wound–wait scheme. Therefore, the two schemes perform better under different circumstances depending upon whether older processes are more likely to wait for resources held by younger processes or not.

18.7 Consider the centralized and the fully distributed approaches to deadlock detection. Compare the two algorithms in terms of message complexity.
Answer: The centralized algorithm for deadlock detection requires individual processors or sites to report its local waits-for graph to a centralized manager. The edges in the waits-for graph are combined and a cycle detection algorithm is performed in the centralized manager. The cost of this algorithm is the cost of communicating the various waits-for graph to the centralized server. In the distributed approach, each site builds its own local waits-for graph and predicts whether there is a possibility of a cycle based on local observations. If indeed there is a possibility of a cycle, a message is sent along the various sites that might constitute a cyclic dependency. Multiple sites that are potentially involved in the cyclic dependency could initiate this operation simultaneously. Therefore, the distributed algorithm is sometimes better than the centralized algorithm and is sometimes worse than the centralized algorithm in terms of message complexity. It is better than the centralized algorithm since it does not communicate the entire local waits-for graph to the centralized server. (Also note that that there could be performance bottlenecks due to the use of the centralized server in the centralized algorithm.) However, the distributed algorithm could

incur more messages when multiple sites are simultaneously exploring the existence of a cyclic dependency.

18.8 Consider the following *hierarchical* deadlock-detection algorithm, in which the global wait-for graph is distributed over a number of different *controllers*, which are organized in a tree. Each non-leaf controller maintains a wait-for graph that contains relevant information from the graphs of the controllers in the subtree below it. In particular, let S_A, S_B, and S_C be controllers such that S_C is the lowest common ancestor of S_A and S_B (S_C must be unique, since we are dealing with a tree). Suppose that node T_i appears in the local wait-for graph of controllers S_A and S_B. Then T_i must also appear in the local wait-for graph of

- Controller S_C
- Every controller in the path from S_C to S_A
- Every controller in the path from S_C to S_B

In addition, if T_i and T_j appear in the wait-for graph of controller S_D and there exists a path from T_i to T_j in the wait-for graph of one of the children of S_D, then an edge $T_i \rightarrow T_j$ must be in the wait-for graph of S_D.

Show that, if a cycle exists in any of the wait-for graphs, then the system is deadlocked.

Answer: A proof of this can be found in the article *Distributed deadlock detection algorithm*, ACM Transactions on Database, June 1982.

18.9 Derive an election algorithm for bidirectional rings that is more efficient than the one presented in this chapter. How many messages are needed for n processes?

Answer: The following algorithm requires $O(n \log n)$ messages. Each node operates in phases and performs:

- If a node is still active, it sends its unique node identifier in both directions.
- In phase k, the tokens travel a distance of 2^k and return back to their points of origin.
- A token might not make it back to the originating node if it encounters a node with a lower unique node identifier.
- A node makes it to the next phase only if it receives its tokens back from the previous round.

The node with the lowest unique node identifier is the only one that will be active after $\log n$ phases.

18.10 Consider a setting where processors are not associated with unique identifiers but the total number of processors is known and the processors are organized along a bidirectional ring. Is it possible to derive an election algorithm for such a setting?

Answer: It is impossible to elect a leader in a deterministic manner when the processors do not contain unique identifiers. Since different processors have no distinguishing mark, they are assumed to start in

the same state and transmit and receive the same messages in every timestep. Hence, there is no way to distinguish the processors even after an arbitrarily large number of timesteps.

18.11 Consider a failure that occurs during 2PC for a transaction. For each possible failure, explain how 2PC ensures transaction atomicity despite the failure.
Answer: Possible failures include (1) failure of a participating site, (2) failure of the coordinator, and (3) failure of the network. We consider each approach in the following:

- **Failure of a Participating Site**—When a participating site recovers from a failure, it must examine its log to determine the fate of those transactions that were in the midst of execution when the failure occurred. The system will then act accordingly depending upon the type of log entry when the failure occurred.
- **Failure of the Coordinator**—If the coordinator fails in the midst of the execution of the commit protocol for transaction T, then the participating sites must decide on the fate of T. The participating sites will then determine to either commit or abort T or wait for the recovery of the failed coordinator.
- **Failure of the Network**—When a link fails, all the messages in the process of being routed through the link do not arrive at their destination intact. From the viewpoint of the sites connected throughout that link, the other sites appears to have failed. Thus, either of the approaches discussed above apply.

18.12 Consider the following failure model for faulty processors. Processors follow the protocol but might fail at unexpected points in time. When processors fail, they simply stop functioning and do not continue to participate in the distributed system. Given such a failure model, design an algorithm for reaching agreement among a set of processors. Discuss the conditions under which agreement could be reached.
Answer: Assume that each node has the following multicast functionality, which we refer to as *basic multicast* or *b-multicast*. *b-multicast(v)*: node simply iterates through all of the nodes in the system and sends an unicast message to each node in the system containing v.
Also assume that each node performs the following algorithm in a synchronous manner assuming that node i starts with the value v_i.

- At round 1, the node performs b-multicast(v_i).
- In each round, the node gathers the values received since the previous round, computes the newly received values, and performs a b-multicast of all the newly received values.
- After round $f + 1$, if the number of failures is less than f, then each node has received exactly the same set of values from all of the other nodes in the system. In particular, this set of values includes all values from nodes that managed to send its local value to any of the nodes that do not fail.

The above algorithm works only in a synchronous setting and when the message delays are bounded. It also works only when the messages are delivered reliably.

Real-time Systems

Chapter 19 introduces real-time computing systems. The requirements of real-time systems differ from those of many of the systems we have described in the previous chapters, largely because real-time systems must produce results within certain deadlines. In this chapter we provide an overview of real-time computer systems and describe how real-time operating systems must be constructed to meet the stringent timing requirements of these systems.

Exercises

19.1 Identify the following environments as either hard or soft real time.

a. Thermostat in a household.

b. Control system for a nuclear power plant.

c. Fuel economy system in an automobile.

d. Landing system in a jet airliner.

Answer: Hard real-time scheduling constraints are required for the nuclear power plant and for the jet airliner. In both settings, a delayed reaction could have disastrous consequences and therefore the scheduler would need to satisfy real-time scheduling requirements. On the other hand, the thermostat and the fuel economy system can be operated within the context of a scheduler that provides only soft real-time constraints. Delayed triggering of these devices would result only in suboptimal use of resources or a slight increase in discomfort.

19.2 Discuss ways in which the priority inversion problem could be addressed in a real-time system. Also discuss whether the solutions could be implemented within the context of a proportional share scheduler.
Answer: The priority inversion problem could be addressed by temporarily changing the priorities of the processes involved. Processes that are accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources in question. When they are finished, their priority reverts to its original value.

This solution can be easily implemented within a proportional share scheduler; the shares of the high-priority processes are simply transferred to the lower-priority process for the duration when it is accessing the resources.

19.3 The Linux 2.6 kernel can be built with no virtual memory system. Explain how this feature may appeal to designers of real-time systems.
Answer: By disabling the virtual memory system, processes are guaranteed to have portions of its address space resident in physical memory. This results in a system that does not suffer from page faults and therefore does not have to deal with unanticipated costs corresponding to paging the address space. The resulting system is appealing to designers of real-time systems who prefer to avoid variability in performance.

19.4 Under what circumstances is the rate-monotonic scheduling inferior to earliest-deadline-first scheduling in meeting the deadlines associated with processes?
Answer: Consider two processes P_1 and P_2 where $p_1 = 50$, $t_1 = 25$ and $p_2 = 75$, $t_2 = 30$. If P_1 were assigned a higher priority than P_2, then the following scheduling events happen under rate-monotonic scheduling. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. The earliest deadline schedule performs the following scheduling events: P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 55$, and so on. This schedule actually meets the deadlines and therefore earliest-deadline-first scheduling is more effective than the rate-monotonic scheduler.

19.5 Consider two processes P_1 and P_2 where $p_1 = 50$, $t_1 = 25$ and $p_2 = 75$, $t_2 = 30$.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart.

b. Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

Answer: Consider when P_1 is assigned a higher priority than P_2 with the rate monotonic scheduler. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. When P_1 is assigned a lower priority than P_2, then P_1 does not meet its deadline since it will not be scheduled in time.

19.6 What are the various components of interrupt and dispatch latency?
Answer: The interrupt latency comprises of the time required to save the state of the current process before invoking the specific interrupt service handler. It therefore comprises the following tasks: determine the interrupt time, perform the context switch, and jump to the appropriate ISR. The dispatch latency corresponds to the time required to stop one process and start another process. It might typically comprise the following tasks: preemption of any process running in the kernel,

release of resources required by the process to be scheduled, and the context-switch cost corresponding to scheduling the new process.

19.7 Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.

Answer: Interrupt latency is the period of time required to perform the following tasks: save the currently executing instruction, determine the type of interupt, save the current process state, and then invoke the appropriate interrupt service routine. Dispatch latency is the cost associated with stopping one process and starting another. Both interrupt and dispatch latency needs to be minimized in order to ensure that real-time tasks receive immediate attention. Furthermore, sometimes interrupts are disabled when kernel data structures are being modified, so the interrupt does not get serviced immediately. For hard real-time systems, the time-period for which interrupts are disabled must be bounded in order to guarantee the desired quality of service.

Multimedia Systems

In earlier chapters, we generally concerned ourselves with how operating systems handle conventional data, such as text files, programs, binaries, word-processing documents, and spreadsheets. However, operating systems may have to handle other kinds of data as well. A recent trend in technology is the incorporation of **multimedia data** into computer systems. Multimedia data consist of continuous-media (audio and video) data as well as conventional files. Continuous-media data differ from conventional data in that continuous-media data—such as frames of video—must be delivered (streamed) according to certain time restrictions (for example, 30 frames per second). In this chapter, we explore the demands of continuous-media data. We also discuss in more detail how such data differ from conventional data and how these differences affect the design of operating systems that support the requirements of multimedia systems.

Exercises

20.1 Provide examples of multimedia applications that are delivered over the Internet.

Answer: Examples include: streaming audio and video of current events as they are taking place, real-time video conferencing, and voice-over IP.

20.2 Distinguish between progressive download and real-time streaming.

Answer: Progressive download is the situation where a media file is downloaded on demand and stored on the local disk. The user is able to play the media file as it is being downloaded without having to wait for the entire file to be accessed. Real-time streaming differs from progressive downloads in that the media file is simply streamed to the client and not stored on the client disk. A limited amount of buffering might be used to tolerate variances in streaming bandwidth, but otherwise the media file is played and discarded without requiring storage.

20.3 Which of the following types of real-time streaming applications can tolerate delay? Which can tolerate jitter?

- Live real-time streaming
- On-demand real-time streaming

Answer: Both delay and jitter are important issues for live real-time streaming. The user is unlikely to tolerate a large delay or significant jitter. Delay is not an important issue for on-demand real-time streaming as the stream does not contain live clips. These applications can also tolerate jitter by buffering a certain amount of data before beginning the playback. In other words, jitter can be overcome by increasing delay, and since delay is not an important consideration for this application, the increase in delay is not very critical.

20.4 Discuss what techniques could be used to meet quality of service requirements for multimedia applications in the following components of a system:

- Process scheduler
- Disk scheduler
- Memory manager

Answer: The process scheduler can use the rate-monotonic scheduling algorithm to guarantee that the processor is utilized for meeting the quality of service requirements in a timely manner. In this scheme, processes are modeled to be periodic and require a fixed processing time every time they are scheduled.
The disk scheduler needs to schedule requests in a timely manner and also minimize the movement of the disk head in order to minimize disk seek overheads. One option is to use a technique that combines the disk SCAN technique with the earliest-deadline-first strategy. Tasks are processed in the order of deadlines. When a batch of requests have the same or related deadlines, then the SCAN technique is used to satisfy the batch of requests.
The memory management needs to ensure that unexpected page faults do not occur during playback of media files. This can be guaranteed if the required pages are swapped into physical memory before the multimedia application is scheduled.

20.5 Explain why the traditional Internet protocols for transmitting data are not sufficient to provide the quality of service guarantees required for a multimedia system. Discuss what changes are required to provide the QoS guarantees.
Answer: Internet protocols such as IP and TCP do not typically reserve resources during connection setup time at the intermediate routers. Consequently, during congestion, when the buffers in routers fill up, some of the packets might be delayed or lost, thereby violating any quality of service requirements. Congestion losses could be avoided if the Internet uses circuit switching and reservation of buffer resources to ensure that packets are not lost.

20.6 Assume that a digital video file is being displayed at a rate of 30 frames per second; the resolution of each frame is 640×480, and 24 bits are

being used to represent each color. Assuming that no compression is being used, what is the bandwidth necessary to deliver this file? Next, assuming that the file has been compressed at a ratio of 200 : 1, what is the bandwidth necessary to deliver the compressed file?
Answer: he bandwidth required for uncompressed data is $30 \times 640 \times 480 \times 3$ bytes per second assuming 640×480 frames at a rate of 30 frames per second. This works out to about 26 MB/s. If the file is compressed by a ratio of 200 : 1, then the bandwidth requirement drops to 135 KB/s.

20.7 A multimedia application consists of a set containing 100 images, 10 minutes of video, and 10 minutes of audio. The compressed sizes of the images, video, and audio are 500 MB, 550 MB, and 8 MB, respectively. The images were compressed at a ratio of 15 : 1, and the video and audio were compressed at 200 : 1 and 10 : 1, respectively. What were the sizes of the images, video, and audio before compression?
Answer: The sizes of the images, video, and audio before compression are 33.33 MB, 2.75 MB, and 0.8 MB respectively.

20.8 Assume that we wish to compress a digital video file using MPEG-1 technology. The target bit rate is 1.5 MB/s. If the video is displayed at a resolution of 352 × 240 at 30 frames per second using 24 bits to represent each color, what is the necessary compression ratio to achieve the desired bit rate?
Answer: The bit rate required to support uncompressed video is 352*240*30*3/(1024*1024) MB/s = 7.2509 MB/s. The required compression ratio is therefore 7.2509*8/1.5 = 38 : 1.

20.9 Consider two processes, P_1 and P_2, where $p_1 = 50$, $t_1 = 25$, $p_2 = 75$, and $t_2 = 30$.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart.

b. Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

Answer: If P_1 were assigned a higher priority than P_2, then the following scheduling events happen under rate-monotonic scheduling. P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 50$, and P_2 is scheduled at $t = 75$. P_2 is not scheduled early enough to meet its deadline. If P_1 were assigned a lower priority than P_2, then the following scheduling events happen under rate-monotonic scheduling. P_2 is scheduled at $t = 0$, P_1 is scheduled at $t = 30$, which is not scheduled early enough to meet its deadline. The earliest-deadline schedule performs the following scheduling events: P_1 is scheduled at $t = 0$, P_2 is scheduled at $t = 25$, P_1 is scheduled at $t = 55$, and so on.

20.10 The following table contains a number of requests with their associated deadlines and cylinders. Requests with deadlines occurring within 100 milliseconds of each other will be batched. The disk head is currently at cylinder 94 and is moving toward cylinder 95. If SCAN-EDF disk scheduling is used, how are the requests batched together, and what is the order of requests within each batch?

request	deadline	cylinder
R1	57	77
R2	300	95
R3	250	25
R4	88	28
R5	85	100
R6	110	90
R7	299	50
R8	300	77
R9	120	12
R10	212	2

Answer: Batch 1 (R1, R4, R5), batch 2 (R6, R9), and batch 3 (R2, R3, R7, R8, R10). Within batch 1, requests are scheduled as: (R5, R1, R4). Within batch 2, requests are scheduled as: (R9, R6). Within batch 3, requests are scheduled as: (R2, R8, R7, R3, R10).

20.11 Repeat the preceding question, but this time batch requests that have deadlines occurring within 75 milliseconds of each other.
Answer: The batches are as follows: Batch 1: (R1), Batch 2: (R4, R5, R6, R9), Batch 3: (R10), Batch 4: (R2, R3, R7, R8).

20.12 Contrast unicasting, multicasting, and broadcasting as techniques for delivering content across a computer network.
Answer: Unicasting is the situation where a server sends data to only one client. If the content is required by multiple clients and if unicast was the only mechanism available, then the server would have to send multiple unicast streams to reach the different clients. Broadcasting allows a server to deliver the content to all of the clients irrespective of whether they wish to receive the content or not. This technique could result in unnecessary work for those clients that do not need the content but still get the broadcast data. Multicasting is a reasonable compromise where the server can send data to some subset of the clients and requires support from the network router to intelligently duplicate the streams at those points in the network where the destination clients are attached to different sub networks.

20.13 Describe why HTTP is often insufficient for delivering streaming media.
Answer: HTTP is a stateless protocol where the server does not maintain any information regarding the clients. This is sufficient for accessing static data, but for streaming media where the clients might require the ability to pause the stream and resume it at a later time, a stateless protocol is insufficient. The server would not be able to keep track of the current position within the stream for a client and therefore would not be able to resume the streaming at a later point in time.

20.14 What operating principle is used by the CineBlitz system in performing admission control for requests for media files?

Answer: Cineblitz differentiates clients into two classes: those that require realtime service and those do not. The resources are allocated such that a fraction of it is reserved for realtime clients and the rest are allocated to non-realtime clients. Furthermore, when a client requires realtime service enters the system, it is admitted into the system only if there are sufficient resources to service the new client. In particular, when a client makes a request, the system estimates the service time for the request and the request is admitted only if the sum of the estimated service times for all admitted requests does not exceed the duration of service cycle T.

The Linux System

Linux is a UNIX-like system that has gained popularity in recent years. In this chapter, we look at the history and development of Linux, and cover the user and programmer interfaces that Linux presents, interfaces that owe a great deal to the UNIX tradition. We also discuss the internal methods by which Linux implements these interfaces. However, since Linux has been designed to run as many standard UNIX applications as possible, it has much in common with existing UNIX implementations. We do not duplicate the basic description of UNIX given in the previous chapter.

Linux is a rapidly evolving operating system. This chapter describes specifically the Linux 2.6 kernel, released in late 2003.

Exercises

21.1 What are the advantages and disadvantages of writing an operating system in a high-level language, such as C?

Answer: There are many advantages to writing an operating system in a high-level language such as C. First, by programming at a higher abstraction, the number of programming errors is reduced as the code becomes more compact. Second, many high-level languages provide advanced features such as bounds checking that further minimize programming errors and security loopholes. Also, high-level programming languages have powerful programming environments that include tools such as debuggers and performance profilers that could be handy for developing code. The disadvantage with using a high-level language is that the programmer is distanced from the underlying machine, which could cause a few problems. First, there could be a performance overhead introduced by the compiler and run-time system used for the high-level language. Second, certain operations and instructions that are available at the machine level might not be accessible from the language level, thereby limiting some of the functionality available to the programmer.

21.2 In what circumstances is the system-call sequence `fork()` `exec()` most appropriate? When is `vfork()` preferable?

Answer: vfork() is a special case of clone and is used to create new processes without copying the page tables of the parent process. vfork() differs from fork in that the parent is suspended until the child makes a call to exec() or exit(). The child shares all memory with its parent, including the stack, until the child makes the call. This implies constraints on the program that it should be able to make progress without requiring the parent process to execute and is not suitable for certain programs where the parent and child processes interact before the child performs an exec. For such programs, the system-call sequence `fork()` `exec()` more appropriate.

21.3 What socket type should be used to implement an intercomputer file-transfer program? What type should be used for a program that periodically tests to see whether another computer is up on the network? Explain your answer.
Answer: Sockets of type SOCK_STREAM use the TCP protocol for communicating data. The TCP protocol is appropriate for implementing an intercomputer file-transfer program since it provides a reliable, flow-controlled, and congestion-friendly communication channel. If data packets corresponding to a file transfer are lost, then they are retransmitted. Furthermore, the file transfer does not overrun buffer resources at the receiver and adapts to the available bandwidth along the channel. Sockets of type SOCK_DGRAM use the UDP protocol for communicating data. The UDP protocol is more appropriate for checking whether another computer is up on the network. Since a connection-oriented communication channel is not required and since there might not be any active entities on the other side to establish a communication channel with, the UDP protocol is more appropriate.

21.4 Linux runs on a variety of hardware platforms. What steps must the Linux developers take to ensure that the system is portable to different processors and memory-management architectures, and to minimize the amount of architecture-specific kernel code?
Answer: The organization of architecture-dependent and architecture-independent code in the Linux kernel is designed to satisfy two design goals: to keep as much code as possible common between architectures and to provide a clean way of defining architecture-specific properties and code. The solution must of course be consistent with the overriding aims of code maintainability and performance.
There are different levels of architecture dependence in the kernel, and different techniques are appropriate in each case to comply with the design requirements. These levels include:

a. **CPU word size and endianness**. These are issues that affect the portability of all software written in C, but especially so for an operating system, where the size and alignment of data must be carefully arranged.

b. **CPU process architecture**. Linux relies on many forms of hardware support for its process and memory management. Different processors have their own mechanisms for changing between protection domains (e.g., entering kernel mode from user mode),

rescheduling processes, managing virtual memory, and handling incoming interrupts.

The Linux kernel source code is organized so as to allow as much of the kernel as possible to be independent of the details of these architecture-specific features. To this end, the kernel keeps not one but two separate subdirectory hierarchies for each hardware architecture. One contains the code that is appropriate only for that architecture, including such functionality as the system call interface and low-level interrupt-management code.

The second architecture-specific directory tree contains C header files that are descriptive of the architecture. These header files contain type definitions and macros designed to hide the differences between architectures. They provide standard types for obtaining words of a given length, macro constants defining such things as the architecture word size or page size, and function macros to perform common tasks such as converting a word to a given byte order or doing standard manipulations to a page-table entry.

Given these two architecture-specific subdirectory trees, a large portion of the Linux kernel can be made portable between architectures. An attention to detail is required: when a 32-bit integer is required, the programmer must use the explicit __int32 type rather than assume than an int is a given size, for example. However, as long as the architecture-specific header files are used, then most process and page-table manipulation can be performed using common code between the architectures. Code that definitely cannot be shared is kept safely detached from the main common kernel code.

21.5 What are the advantages and disadvantages of making only some of the symbols defined inside a kernel accessible to a loadable kernel module?
Answer: The advantage of making only some of the symbols defined inside a kernel accessible to a loadable kernel module is that there is a fixed set of entry points made available to the kernel module. This ensures that loadable modules cannot invoke arbitrary code within the kernel and thereby interfere with the kernel's execution. By restricting the set of entry points, the kernel is guaranteed that the interactions with the module take place at controlled points where certain invariants hold. The disadvantage of making only a small set of the symbols defined accessible to the kernel module is the loss in flexibility and might sometimes lead to a performance issue as some of the details of the kernel are hidden from the module.

21.6 What are the primary goals of the conflict resolution mechanism used by the Linux kernel for loading kernel modules?
Answer: Conflict resolution prevents different modules from having conflicting access to hardware resources. In particular, when multiple drivers are trying to access the same hardware, it resolves the resulting conflict.

21.7 Discuss how the clone() operation supported by Linux is used to support both processes and threads.

Answer: In Linux, threads are implemented within the kernel by a clone mechanism that creates a new process within the same virtual address space as the parent process. Unlike some kernel-based thread packages, the Linux kernel does not make any distinction between threads and processes: a thread is simply a process that did not create a new virtual address space when it was initialized.
The main advantage of implementing threads in the kernel rather than in a user-mode library are that:

- kernel-threaded systems can take advantage of multiple processors if they are available; and
- if one thread blocks in a kernel service routine (for example, a system call or page fault), other threads are still able to run.

21.8 Would one classify Linux threads as user-level threads or as kernel-level threads? Support your answer with the appropriate arguments.
Answer: Linux threads are kernel-level threads. The threads are visible to the kernel and are independently schedule-able. User-level threads, on the other hand, are not visible to the kernel and are instead manipulated by user-level schedulers. In addition, the threads used in the Linux kernel are used to support both the thread abstraction and the process abstraction. A new process is created by simply associating a newly created kernel thread with a distinct address space, whereas a new thread is created by simply creating a new kernel thread with the same address space. This further indicates that the thread abstaction is intimately tied into the kernel.

21.9 What are the extra costs incurred by the creation and scheduling of a process, as compared to the cost of a cloned thread?
Answer: In Linux, creation of a thread involves only the creation of some very simple data structures to describe the new thread. Space must be reserved for the new thread's execution context, its saved registers, its kernel stack page and dynamic information such as its security profile and signal state, but no new virtual address space is created.

Creating this new virtual address space is the most expensive part of the creation of a new process. The entire page table of the parent process must be copied, with each page being examined so that copy-on-write semantics can be achieved and so that reference counts to physical pages can be updated. The parent process's virtual memory is also affected by the process creation: any private read/write pages owned by the parent must be marked read-only so that copy-on-write can happen (copy-on-write relies on a page fault being generated when a write to the page occurs).

Scheduling of threads and processes also differs in this respect. The decision algorithm performed when deciding what process to run next is the same regardless of whether the process is a fully independent process or just a thread, but the action of context switching to a separate process is much more costly than switching to a thread. A process requires that the CPU's virtual memory control registers be updated to point to the new virtual address space's page tables.

In both cases—creation of a process or context switching between processes—the extra virtual memory operations have a significant cost. On many CPUs, changing page tables or swapping between page tables is not cheap: all or part of the virtual address translation look-aside buffers in the CPU must be purged when the page tables are changed. These costs are not incurred when creating or scheduling between threads.

21.10 The Linux scheduler implements *soft* real-time scheduling. What features are missing that are necessary for some real-time programming tasks? How might they be added to the kernel?

Answer: Linux's "soft" real-time scheduling provides ordering guarantees concerning the priorities of runnable processes: real-time processes will always be given a higher priority by the scheduler than normal time-sharing processes, and a real-time process will never be interrupted by another process with a lower real-time priority.

However, the Linux kernel does not support "hard" real-time functionality. That is, when a process is executing a kernel service routine, that routine will always execute to completion unless it yields control back to the scheduler either explicitly or implicitly (by waiting for some asynchronous event). There is no support for preemptive scheduling of kernel-mode processes. As a result, any kernel system call that runs for a significant amount of time without rescheduling will block execution of any real-time processes.

Many real-time applications require such hard real-time scheduling. In particular, they often require guaranteed worst-case response times to external events. To achieve these guarantees, and to give user-mode real-time processes a true higher priority than kernel-mode lower-priority processes, it is necessary to find a way to avoid having to wait for low-priority kernel calls to complete before scheduling a real-time process. For example, if a device driver generates an interrupt that wakes up a high-priority real-time process, then the kernel needs to be able to schedule that process as soon as possible, even if some other process is already executing in kernel mode.

Such preemptive rescheduling of kernel-mode routines comes at a cost. If the kernel cannot rely on non-preemption to ensure atomic updates of shared data structures, then reads of or updates to those structures must be protected by some other, finer-granularity locking mechanism. This fine-grained locking of kernel resources is the main requirement for provision of tight scheduling guarantees.

Many other kernel features could be added to support real-time programming. Deadline-based scheduling could be achieved by making modifications to the scheduler. Prioritization of IO operations could be implemented in the block-device IO request layer.

21.11 Under what circumstances would an user process request an operation that results in the allocation of a demand-zero memory region?

Answer: Uninitialized data can be backed by demand-zero memory regions in a process's virtual address space. In addition, newly malloced space can also be backed by a demand-zero memory region.

21.12 What scenarios would cause a page of memory to be mapped into an user program's address space with the copy-on-write attribute enabled?
Answer: When a process performs a fork operation, a new process is created based on the original binary but with a new address space that is a clone of the original address space. One possibility is to not to create a new address space but instead to share the address space between the old process and the newly created process. The pages of the address space are mapped with the copy-on-write attribute enabled. Then, when one of the processes performs an update on the shared address space, a new copy is made and the processes no longer share the same page of the address space.

21.13 In Linux, shared libraries perform many operations central to the operating system. What is the advantage of keeping this functionality out of the kernel? Are there any drawbacks? Explain your answer.
Answer: There are a number of reasons for keeping functionality in shared libraries rather than in the kernel itself. These include:

a. **Reliability**. Kernel-mode programming is inherently higher risk than user-mode programming. If the kernel is coded correctly so that protection between processes is enforced, then an occurrence of a bug in a user-mode library is likely to affect only the currently executing process, whereas a similar bug in the kernel could conceivably bring down the entire operating system.

b. **Performance**. Keeping as much functionality as possible in user-mode shared libraries helps performance in two ways. First of all, it reduces physical memory consumption: kernel memory is non-pageable, so every kernel function is permanently resident in physical memory, but a library function can be paged in from disk on demand and does not need to be physically present all of the time. Although the library function may be resident in many processes at once, page sharing by the virtual memory system means that it is loaded at most once into physical memory.

   Second, calling a function in a loaded library is a very fast operation, but calling a kernel function through a kernel system service call is much more expensive. Entering the kernel involves changing the CPU protection domain, and once in the kernel, all of the arguments supplied by the process must be very carefully checked for correctness: the kernel cannot afford to make any assumptions about the validity of the arguments passed in, whereas a library function might reasonably do so. Both of these factors make calling a kernel function much slower than calling the same function in a library.

c. **Manageability**. Many different shared libraries can be loaded by an application. If new functionality is required in a running system, shared libraries to provide that functionality can be installed without interrupting any already running processes. Similarly, existing shared libraries can generally be upgraded without requiring any system down time. Unprivileged users can create shared libraries to be run by their own programs. All of these

attributes make shared libraries generally easier to manage than kernel code.

There are, however, a few disadvantages to having code in a shared library. There are obvious examples of code that is completely unsuitable for implementation in a library, including low-level functionality such as device drivers or file systems. In general, services shared around the entire system are better implemented in the kernel if they are performance-critical, since the alternative—running the shared service in a separate process and communicating with it through interprocess communication—requires two context switches for every service requested by a process. In some cases, it may be appropriate to prototype a service in user mode but implement the final version as a kernel routine.

Security is also an issue. A shared library runs with the privileges of the process calling the library. It cannot directly access any resources inaccessible to the calling process, and the calling process has full access to all of the data structures maintained by the shared library. If the service being provided requires any privileges outside of a normal process's, or if the data managed by the library needs to be protected from normal user processes, then libraries are inappropriate and a separate server process (if performance permits) or a kernel implementation is required.

21.14 The directory structure of a Linux operating system could comprise files corresponding to different file systems, including the Linux /proc file system. What are the implications that arise from having to support different file system types on the structure of the Linux kernel?

Answer: There are many implications to having to support different file system types within the Linux kernel. For one thing, the file system interface should be independent of the data layouts and data structures used within the file system to store file data. For another thing, it might have to provide interfaces to file systems where the file data is not static data and is not even stored on the disk; instead, the filedata could be computed every time an operation is invoked to access it, as is the case with the /proc file system. These call for a fairly general virtual interface to sit on top of the different file systems.

21.15 In what ways does the Linux setuid feature differ from the setuid feature in standard UNIX?

Answer: Linux augments the standard setuid feature in two ways. First, it allows a program to drop and reacquire its effective uid repeatedly. In order to minimize the amount of time that a program executes with all of its privileges, a program might drop to a lower privilege level and thereby prevent the exploitation of security loopholes at the lower-level. However, when it needs to perform privileged operations, it can switch to its effective uid. Second, Linux allows a process to take on only a subset of the rights of the effective uid. For instance, an user can use a process that serves files without having control over the process in terms of being able to kill or suspend the process.

21.16 The Linux source code is freely and widely available over the Internet or from CD-ROM vendors. What three implications does this availability have on the security of the Linux system?

Answer: The open availability of an operating system's source code has both positive and negative impacts on security, and it is probably a mistake to say that it is definitely a good thing or a bad thing.

Linux's source code is open to scrutiny by both the good guys and the bad guys. In its favor, this has resulted in the code being inspected by a large number of people who are concerned about security and who have eliminated any vulnerabilities they have found.

On the other hand is the "security through obscurity" argument, which states that attackers' jobs are made easier if they have access to the source code of the system they are trying to penetrate. By denying attackers information about a system, the hope is that it will be harder for those attackers to find and exploit any security weaknesses that may be present.

In other words, open source code implies both that security weaknesses can be found and fixed faster by the Linux community, increasing the security of the system; and that attackers can more easily find any weaknesses that do remain in Linux.

There are other implications for source code availability, however. One is that if a weakness in Linux is found and exploited, then a fix for that problem can be created and distributed very quickly. (Typically, security problems in Linux tend to have fixes available to the public within 24 hours of their discovery.) Another is that if security is a major concern to particular users, then it is possible for those users to review the source code to satisfy themselves of its level of security or to make any changes that they wish to add new security measures.

Windows XP

The Microsoft Windows XP operating system is a 32/64-bit preemptive multitasking operating system for AMD K6/K7, Intel IA32/IA64 and later microprocessors. The successor to Windows NT/2000, Windows XP, is also intended to replace the MS-DOS operating system. Key goals for the system are security, reliability, ease of use, Windows and POSIX application compatibility, high performance, extensibility, portability and international support. This chapter discusses the key goals for Windows XP, the layered architecture of the system that makes it so easy to use, the file system, networks, and the programming interface. Windows XP serves as an excellent case study as an example operating system.

Exercises

22.1 Under what circumstances would one use the deferred procedure calls facility in Windows XP?
Answer: Deferred procedure calls are used to postpone interrupt processing in situations where the processing of device interrupts can be broken into a critical portion that is used to unblock the device and a non-critical portion that can be scheduled later at a lower priority. The non-critical section of code is scheduled for later execution by queuing a deferred procedure call.

22.2 What is a handle, and how does a process obtain a handle?
Answer: User-mode code can access kernel-mode objects by using a reference value called a handle. An object handle is thus an identifier (unique to a process) that allows access and manipulation of a system resource. When a user-mode process wants to use an object it calls the object manager's open method. A reference to the object is inserted in the process's object table and a handle is returned. Processes can obtain handles by creating an object, opening an existing object, receiving a duplicated handle from another process, or inheriting a handle from a parent process.

22.3 Describe the management scheme of the virtual memory manager. How does the VM manager improve performance?

Answer: The VM manager uses a page-based management scheme. Pages of data allocated to a process that are not in physical memory are either stored in paging files on disk or mapped to a regular file on a local or remote file system. To improve performance of this scheme, a privileged process is allowed to lock selected pages in physical memory preventing those pages from being paged out. Furthermore, since when a page is used, adjacent pages will likely be used in the near future, adjacent pages are prefetched to reduce the total number of page faults.

22.4 Describe an useful application of the no-access page facility provided in Windows XP.
Answer: When a process accesses a no-access page, an exception is raised. This feature is used to check whether a faulty program accesses beyond the end of an array. The array needs to be allocated in a manner such that it appears at the end of a page, so that buffer overruns would cause exceptions.

22.5 The IA64 processors contain registers that can be used to address a 64-bit address space. However, Windows XP limits the address space of user programs to 8-TB, which corresponds to 43 bits' worth. Why was this decision made?
Answer: Each page table entry is 64 bits wide and each page is 8 KB on the IA64. Consequently, each page can contain 1024 page table entries. The virtual memory system therefore requires three levels of page tables to translate a virtual address to a physical address in order to address a 8-TBvirtual address space. (The first-level page table is indexed using the first 10 bits of the virtual address, the second-level page table using the next 10 bits, and the third-level page table is indexed using the next 10 bits, with the remaining 13 bits used to index into the page.) If the virtual address space is bigger, more levels would be required in the page table organization, and therefore more memory references would be required to translate a virtual address to the corresponding physical address during a TLB fault. The decision regarding the 43-bit address space represents a trade off between the size of the virtual address space and the cost of performing an address translation.

22.6 Describe the three techniques used for communicating data in a local procedure call. What different settings are most conducive for the application of the different message-passing techniques?
Answer: Data is communicated using one of the following three facilities: 1) messages are simply copied from one process to the other, 2) a shared memory segment is created and messages simply contain a pointer into the shared memory segment, thereby avoiding copies between processes, 3) a process directly writes into the other process's virtual space.

22.7 What manages cache in Windows XP? How is cache managed?
Answer: In contrast to other operating systems where caching is done by the file system, Windows XP provides a centralized cache manager which works closely with the VM manager to provide caching services for all components under control of the I/O manager. The size of the cache changes dynamically depending upon the free memory available

in the system. The cache manager maps files into the upper half of the system cache address space. This cache is divided into blocks that can each hold a memory-mapped region of a file.

22.8 What is the purpose of the Windows16 execution environment? What limitations are imposed on the programs executing inside this environment? What are the protection guarantees provided between different applications executing inside the Windows16 environment? What are the protection guarantees provided between an application executing inside the Windows16 environment and a 32-bit application?
Answer: Windows16 execution environment provides a virtual environment for executing 16-bit applications that use the Windows 3.1 kernel interface. The interface is supported in software using stub routines that call the appropriate Win32 API subroutines by converting 16-bit addresses into 32-bit addresses. This allows the system to run legacy applications. The environment can multitask with other processes on Windows XP. It can contain multiple Windows16 applications, but all applications share the same address space and the same input queue. Also, one can execute only one Windows16 application at a given point in time. A Windows16 application can therefore crash other Windows16 applications by corrupting the address space, but it cannot corrupt the address spaces of Win32 applications. Multiple Windows16 execution environments could also coexist.

22.9 Describe two user-mode processes that Windows XP provides to enable it to run programs developed for other operating systems.
Answer: Environmental subsystems are user-mode processes layered over the native executable services to enable Windows XP to run programs developed for other operating systems. (1) A Win32 application called the virtual DOS machine (VDM) is provided as a user-mode process to run MS-DOS applications. The VDM can execute or emulate Intel 486 instructions and also provides routines to emulate MSDOS BIOS services and provides virtual drivers for screen, keyboard, and communication ports. (2) Windows-on-Windows (WOW32) provides kernel and stub routines for Windows 3.1 functions. The stub routines call the appropriate Win32 subroutines, converting the 16-bit addresses into 32-bit addresses.

22.10 How does the NTFS directory structure differ from the directory structure used in UNIX operating systems?
Answer: The NTFS namespace is organized as a hierarchy of directories where each directory uses a B+ tree data structure to store an index of the filenames in that directory. The index root of a directory contains the top level of the B+ tree. Each entry in the directory contains the name and file reference of the file as well as the update timestamp and file size. The UNIX operating system simply stores a table of entries mapping names to i-node numbers in a directory file. Lookups and updates require a linear scan of the directory structure in UNIX systems.

22.11 What is a process, and how is it managed in Windows XP?
Answer: A process is an executing instance of an application containing one or more threads. Threads are the units of code that are scheduled

by the operating system. A process is started when some other process calls the CreateProcess routine, which loads any dynamic link libraries used by the process, resulting in a primary thread. Additional threads can also be created. Each thread is created with its own stack with a wrapper function providing thread synchronization.

22.12 What is the fiber abstraction provided by Windows XP? How does it differ from the threads abstraction?

Answer: A fiber is a sequential stream of execution within a process. A process can have multiple fibers in it, but unlike threads, only one fiber at a time is permitted to execute. The fiber mechanism is used to support legacy applications written for a fiber-execution model.

Influential Operating Systems

Now that you understand the fundamental concepts of operating systems (CPU scheduling, memory management, processes, and so on), we are in a position to examine how these concepts have been applied in several older and highly influential operating systems. Some of them (such as the XDS-940 and the THE system) were one-of-a-kind systems; others (such as OS/360) are widely used. The order of presentation highlights the similarities and differences of the systems; it is not strictly chronological or ordered by importance. The serious student of operating systems should be familiar with all these systems.

Exercises

23.1 Discuss what considerations were taken into account by the computer operator in deciding in what sequences programs would be run on early computer systems that were manually operated.
Answer: Jobs with similar needs are batched together and run together to reduce set-up time. For instance, jobs that require the same compiler because they were written in the same language are scheduled together so that the compiler is loaded only once and used on both programs.

23.2 What were the various optimizations used to minimize the discrepancy between CPU and I/O speeds on early computer systems?
Answer: An optimization used to minimize the discrepancy between CPU and I/O speeds is spooling. Spooling overlaps the I/O of one job with the computation of other jobs. The spooler for instance could be reading the input of one job while printing the output of a different job or while executing another job.

23.3 Consider the page replacement algorithm used by Atlas. In what ways is it different from the clock algorithm discussed in an earlier chapter?
Answer: The page replacement algorithm used in Atlas is very different from the clock algorithm discussed in earlier chapters. The Atlas system keeps track of whether a page was accessed in each period of 1024 instructions for the last 32 periods. Let t1 be the time since the most recent reference to a page, while t2 is the interval between the last two

references of a page. The paging system then discards any page that has t1 > t2 + 1. If it cannot find any such page, it discards the page with the largest t2 - t1. This algorithm assumes that programs access memory in loops and the idea is to retain pages even if it has not been accessed for a long time if there has been a history of accessing the page regularly albeit at long intervals. The clock algorithm, on the other hand, is an approximate version of the least recently used algorithm and therefore discards the least recently used page without taking into account that some of the pages might be infrequently but repeatedly accessed.

23.4 Consider the multilevel feedback queue used by CTSS and MULTICS. Consider a program that consistently uses 7 time units everytime it is scheduled before it performs an I/O operation and blocks. How many time units are allocated to this program when it is scheduled for execution at different points of time?

Answer: Assume that the process is initially scheduled for one time unit. Since the process does not finish by the end of the time quantum, it is moved to a lower level queue and its time quantum is raised to two time units. This process continues till it is moved to a level 4 queue with a time quantum of 8 time units. In certain multilevel systems, when the process executes next and does not use its full time quantum, the process might be moved back to a level 3 queue.

23.5 What are the implications of supporting BSD functionality in user mode servers within the Mach operating system?

Answer: Mach operating system supports the BSD functionality in user mode servers. When the user process makes a BSD call, it traps into kernel mode and the kernel copies the arguments to the user level server. A context switch is then made and the user level performs the requested operation and computes the results which are then copied back to the kernel space. Another context switch takes place to the original process which is in kernel mode and the process eventually transitions from kernel mode to user mode along with the results of the BSD call. Therefore, in order to perform the BSD call, there are two kernel crossings and two process switches thereby resulting in a large overhead. This is significantly higher than the cost if the BSD functionality is supported within the kernel.