INSTRUCTOR'S MANUAL
TO ACCOMPANY

APPLIED OPERATING-SYSTEM CONCEPTS

ABRAHAM SILBERSCHATZ
Bell Laboratories

PETER BAER GALVIN
Corporate Technologies

GREG GAGNE
Westminster College

PREFACE

This volume is an instructor's manual for the First Edition of *Applied Operating-System Concepts* by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. It consists of answers to the exercises in the parent text. In cases where the answer to a question involves a long program, algorithm development, or an essay, no answer is given, but simply the keywords "No Answer" are added.

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, both instructors and students, to help us in improving this manual. If you have better solutions to the exercises or other items which would be of use with *Applied 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 avi@bell-labs.com. Physical mail may be sent to Avi Silberschatz, Information Sciences Research Center, MH 2T-310, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, USA.

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

CONTENTS

Chapter 1

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.

■ Answers to Exercises

1.1 What are the three main purposes of an operating system?
Answer:

- To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.
- To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.
- As a control program it serves two major functions: (1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) management of the operation and control of I/O devices.

1.2 List the four steps that are necessary to run a program on a completely dedicated machine.
Answer:

a. Reserve machine time.

b. Manually load program into memory.

c. Load starting address and begin execution.

d. Monitor and control execution of program from console.

1.3 What is the main advantage of multiprogramming?

Answer: Multiprogramming makes efficient use of the CPU by overlapping the demands for the CPU and its I/O devices from various users. It attempts to increase CPU utilization by always having something for the CPU to execute.

1.4 What are the main differences between operating systems for mainframe computers and personal computers?

Answer: No answer.

1.5 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 we have 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.

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

a. Batch

b. Interactive

c. Time sharing

d. Real time

e. Network

f. Distributed

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; it 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 systems 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**.

f. **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 telephone line.

1.7 We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to "waste" resources? Why is such a system not really wasteful?

Answer: Single-user systems should maximize use of the system for the user. A GUI might "waste" CPU cycles, but it optimizes the user's interaction with the system.

1.8 Under what circumstances would a user be better off using a time-sharing system, rather than a personal computer 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.

1.9 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.

1.10 What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment?

Answer: The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system it is running. Therefore when writing an operating system for a real-time system, the writer must be sure that his scheduling schemes don't allow response time to exceed the time constraint.

1.11 Consider the various definitions of *operating system*. Consider whether the operating system should include applications such as Web browsers and mail programs. Argue both that it should and that it should not, and support your answer.

Answer: No answer.

Chapter 2

COMPUTER-SYSTEM STRUCTURES

Chapter 2 discusses the general structure of computer systems. It may be a good idea to review the basic concepts of machine organization and assembly language programming. The students should be comfortable with the concepts of memory, CPU, registers, I/O, interrupts, instructions, and the instruction execution cycle. Since the operating system is the interface between the hardware and user programs, a good understanding of operating systems requires an understanding of both hardware and programs.

■ Answers to Exercises

2.1 *Prefetching* is a method of overlapping the I/O of a job with that job's own computation. The idea is simple. After a read operation completes and the job is about to start operating on the data, the input device is instructed to begin the next read immediately. The CPU and input device are then both busy. With luck, by the time the job is ready for the next data item, the input device will have finished reading that data item. The CPU can then begin processing the newly read data, while the input device starts to read the following data. A similar idea can be used for output. In this case, the job creates data that are put into a buffer until an output device can accept them.

Compare the prefetching scheme with the spooling scheme, where the CPU overlaps the input of one job with the computation and output of other jobs.

Answer: Prefetching is a user-based activity, while spooling is a system-based activity. Spooling is a much more effective way of overlapping I/O and CPU operations.

2.2 How does the distinction between monitor mode and user mode function as a rudimentary form of protection (security) system?

Answer: By establishing a set of privileged instructions that can be executed only when in the monitor mode, the operating system is assured of controlling the entire system at all times.

2.3 What are the differences between a trap and an interrupt? What is the use of each function?

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.

2.4 For what types of operations is DMA useful? Explain your answer.

Answer: DMA is useful for transferring large quantities of data between memory and devices. It eliminates the need for the CPU to be involved in the transfer, allowing the transfer to complete more quickly and the CPU to perform other tasks concurrently.

2.5 Which of the following instructions should be privileged?

a. Set value of timer.

b. Read the clock.

c. Clear memory.

d. Turn off interrupts.

e. Switch from user to monitor mode.

Answer: The following instructions should be privileged:

a. Set value of timer.

b. Clear memory.

c. Turn off interrupts.

d. Switch from user to monitor mode.

2.6 Some computer systems do not provide a privileged mode of operation in hardware. Consider whether it is possible to construct a secure operating system for these computers. 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, APL, and LISP systems, for example). The software interpreter would provide, in software, what the hardware does not provide.

b. Require meant 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.

2.7 Some early computers protected the operating system by placing it in a memory partition that could not be modified by either the user job or the operating system itself. Describe two difficulties that you think could arise with such a scheme.

Answer: The data required by the operating system (passwords, access controls, accounting information, and so on) would have to be stored in or passed through unprotected memory and thus be accessible to unauthorized users.

2.8 Protecting the operating system is crucial to ensuring that the computer system operates correctly. Provision of this protection is the reason behind dual-mode operation, memory protection, and the timer. To allow maximum flexibility, however, we would also like to place minimal constraints on the user.

The following is a list of operations that are normally protected. What is the *minimal* set of instructions that must be protected?

a. Change to user mode.
b. Change to monitor mode.
c. Read from monitor memory.
d. Write into monitor memory.
e. Fetch an instruction from monitor memory.
f. Turn on timer interrupt.
g. Turn off timer interrupt.

Answer: The minimal set of instructions that must be protected are:

a. Change to monitor mode.
b. Read from monitor memory.
c. Write into monitor memory.
d. Turn off timer interrupt.

2.9 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. Cahces 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.

2.10 Writing an operating system that can operate without interference from malicious or undebugged user programs requires some hardware assistance. Name three hardware aids for writing an operating system, and describe how they could be used together to protect the operating system.
Answer:

a. Monitor / user mode
b. Privileged instructions
c. Timer
d. Memory protection

2.11 Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?
Answer: No answer.

Chapter 3

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.

■ Answers to Exercises

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

- The creation and deletion of both user and system processes
- The suspension and resumption of processes
- The provision of mechanisms for process synchronization
- The provision of mechanisms for process communication
- The provision of mechanisms for deadlock handling

3.2 What are the three major activities of an operating system in regard to memory management?
Answer:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes are to be loaded into memory when memory space becomes available.
- Allocate and deallocate memory space as needed.

3.3 What are the three major activities of an operating system in regard to secondary-storage management?

Answer:

- Free-space management.
- Storage allocation.
- Disk scheduling.

3.4 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

3.5 What is the purpose of the command interpreter? Why is it usually separate from the kernel?

Answer: It 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.

3.6 List five services provided by an operating system. Explain how each provides convenience to the users. Explain also in which cases it would be impossible for user-level programs to provide these services.

Answer:

- **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.
- **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 only access devices they should have access to and to only access them when they are otherwise unused.
- **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.
- **Communications**. Message passing between systems requires 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.

- **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 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.

3.7 What is the purpose of system calls?

Answer: System calls allow user-level processes to request services of the operating system.

3.8 Using system calls, write a program in either C or C++ that reads data from one file and copies it to another file. Such a program was described in Section 3.3.

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

3.9 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 where 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.

3.10 What is the purpose of system programs?

Answer: System programs can be thought of as bundles of useful system calls. They provide basic functionality to users and so users do not need to write their own programs to solve common problems.

3.11 What is the main advantage of the layered approach to system design?

Answer: As in all cases of modular design, designing an operating system in a modular way has several advantages. The system is easier to debug and modify because changes affect only limited sections of the system rather than touching all sections of the operating system. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer.

3.12 What are the main advantages of the microkernel approach to system design?

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.

3.13 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 may run on one physical system.

3.14 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 may 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.

3.15 Why is the separation of mechanism and policy a desirable property?

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.

3.16 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 so that building the operating system is made 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).

Chapter 4

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 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 5.

■ Answers to Exercises

4.1 MS-DOS provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

Answer:

- A method of time sharing must be implemented to allow each of several processes to have access to the system. This method involves the preemption of processes that do not voluntarily give up the CPU (by using a system call, for instance) and the kernel being reentrant (so more than one process may be executing kernel code concurrently).
- Processes and system resources must have protections and must be protected from each other. Any given process must be limited in the amount of memory it can use and the operations it can perform on devices like disks.
- Care must be taken in the kernel to prevent deadlocks between processes, so processes aren't waiting for each other's allocated resources.

4.2 Describe the differences among short-term, medium-term, and long-term scheduling.

Answer:

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

4.3 A DECSYSTEM-20 computer has multiple register sets. Describe the actions of a context switch if the new context is already loaded into one of the register sets. What else must happen if the new context is in memory rather than in a register set and all the register sets are in use?

Answer: The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and be moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

4.4 Describe the actions a kernel takes 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.

4.5 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: No answer.

4.6 Consider the interprocess-communication scheme where mailboxes are used.

a. Suppose a process P wants to wait for two messages, one from mailbox A and one from mailbox B. What sequence of **send** and **receive** should it execute?

b. What sequence of `send` and `receive` should P execute if P wants to wait for one message either from mailbox A or from mailbox B (or from both)?

c. A `receive` operation makes a process wait until the mailbox is nonempty. Either devise a scheme that allows a process to wait until a mailbox is empty, or explain why such a scheme cannot exist.

Answer: No answer.

Chapter 5

THREADS

The process model introduced in Chapter 4 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.

Answers to Exercises

5.1 Provide two programming examples of multithreading giving improved performance over a single-threaded solution.

Answer: (1) A Web server that services each request in a separate thread. (2) A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel. (3) An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

5.2 Provide two programming examples of multithreading that would *not* improve performance over 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.

5.3 What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

Answer: (1) User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads. (2) User threads are scheduled by the thread library and the kernel schedules kernel threads. (3) Kernel threads need not be associated with a process whereas every user thread belongs to a process.

5.4 Describe the actions taken by a kernel to context switch between kernel-level threads.

Answer: Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

5.5 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.

5.6 What resources are used when a thread is created? How do they differ from those used when a process is created?

Answer: Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

5.7 Modify the `MessageQueue` class such that the `receive()` method blocks until there is a message available in the queue.

Answer: Please refer to the supporting Web site for source code solution. Also note that this solution is not thread-safe. A mechanism for developing thread-safe Java programs will be shown in Chapter 7.

5.8 Implement the `send()` and `receive()` methods for a message-passing system that has zero capacity. Implement the `send()` method such that the sender blocks until the receiver accepts the message. Implement the `receive()` method such that the receiver blocks until there is a message available in the queue.

Answer: Please refer to the supporting Web site for source code solution. Also note that this solution is not thread-safe. A mechanism for developing thread-safe Java programs will be shown in Chapter 7.

5.9 Implement the `send()` and `receive()` methods for a message-passing system with bounded capacity. Implement the `send()` method such that the sender will block until there is space available in the queue. Implement the `receive()` method such that the receiver will block until there is a message available in the queue.

Answer: Please refer to the supporting Web site for source code solution. Also note that this solution is not thread-safe. A mechanism for developing thread-safe Java programs will be shown in Chapter 7.

5.10 Write a multithreaded Java program that generates the Fibonacci series. The operation of this program should be as follows. The user will run the program and 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.

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

5.11 Write a multithreaded Java 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 that the user entered.

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

Chapter 6

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 the trace tape up 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.

■ Answers to Exercises

6.1 A CPU scheduling algorithm determines an order for the execution of its scheduled processes. Given *n* processes to be scheduled on one processor, how many possible different schedules are there? Give a formula in terms of *n*.

Answer: $n!$ (n factorial $= n \times n - 1 \times n - 2 \times ... \times 2 \times 1$)

6.2 Define the difference between preemptive and nonpreemptive scheduling. State why strict nonpreemptive scheduling is unlikely to be used in a computer center.

Answer: Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process. Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

6.3 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.

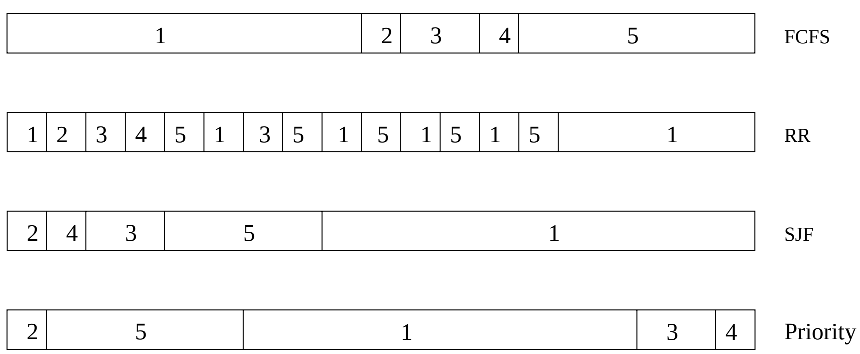
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

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

6.4 Suppose that the following processes arrive for execution at the times indicated. Each process will run the listed amount of time. In answering the questions, use nonpreemptive scheduling and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P_1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P_1 and P_2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

Answer:

a. 10.53

b. 9.53

c. 6.86

Remember that turnaround time is finishing time minus arrival time, so you have to subtract the arrival times to compute the turnaround times. FCFS is 11 if you forget to subtract arrival time.

6.5 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.

6.6 What advantage is there in having different time-quantum sizes on different levels of a multilevel queueing system?

Answer: Processes that need more frequent servicing, for instance, interactive processes such as editors, can be in a queue with a small time quantum. Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, making more efficient use of the computer.

6.7 Consider the following 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

6.8 Many CPU scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on.

These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of sets of algorithms?

a. Priority and SJF

b. Multilevel feedback queues and FCFS

c. Priority and FCFS

d. RR and SJF

Answer:

a. The shortest job has the highest priority.

b. The lowest level of MLFQ is FCFS.

c. FCFS gives the highest priority to the job having been in existence the longest.

d. None

6.9 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

Answer: It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

6.10 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.

6.11 The following questions refer to the Java-based Round Robin Scheduler described in Section 6.7.3.

a. If there is currently a thread T_1 in the queue for the `Scheduler` running at priority 4 and another thread T_2 with priority 2, is it possible for thread T_2 to run during the time quantum for T_1? Explain.

b. Explain what happens if the currently running thread has its `stop()` method invoked during its time quantum.

c. Explain what happens if the scheduler selects a thread to run that has had its `suspend()` method invoked.

Answer:

a. Yes, if thread T_1 blocks for some reason (such as I/O), the JVM will schedule another thread. In this instance it would be thread T_2.

b. The remainder of its time quantum will be shared by all other threads in the queue using the default scheduling algorithm of the JVM.

c. The remainder of its time quantum will be shared by all other threads in the queue using the default scheduling algorithm of the JVM.

6.12 Modify the `CircularList` class by adding a method `isEmpty()` that returns information about whether the queue is empty. Have the scheduler check whether the queue is empty. If the queue is empty, have the scheduler go to sleep for some duration and then awake to check the queue again.

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

6.13 Modify the Java-based scheduler such that it has multiple queues representing different priorities. For example, have three separate queues, one each for priority 2, 3, and 4. Have the scheduler select a thread from the highest-priority queue, set that thread's priority to 5, and allow the thread to run for a time quantum. When the time quantum expires, select the next thread from the highest queue and repeat the process. You should also modify the `Scheduler` class such that, when a thread is given to the scheduler, an initial thread priority is specified.

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

Chapter 7

PROCESS SYNCHRONIZATION

Chapter 7 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.

■ Answers to Exercises

7.1 The first known correct software solution to the critical-section problem for two threads was developed by Dekker; it is shown in Figure 7.42. The two threads, T_0 and T_1, coordinate activity sharing an object of class `Dekker`. Show that the algorithm satisfies all three requirements for the critical-section problem.

Answer: No answer.

7.2 In Chapter 5, we gave a multithreaded solution to the bounded-buffer problem that used message passing. The `MessageQueue` class is not considered thread safe, meaning that a race condition is possible when multiple threads attempt to concurrently access the queue. Modify the `MessageQueue` class using Java synchronization such that it is thread-safe.

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

7.3 Create a `BinarySemaphore` class that implements a binary semaphore.

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

7.4 The `wait()` statement in all Java program examples was part of a `while` loop. Explain why you would always need to use a `while` statement when using the `wait()`, and why you would never use an `if` statement.

Answer: This is an important issue to emphasize! Java only provides anonymous notification—you cannot notify a certain thread that a certain condition is true. When a thread is notified, it is its responsibility to re-check the condition that it is waiting for. If a thread did not re-check the condition, it may have received the notification without the condition having been met. As an example, consider the `doWork()` method in Figure 7.38. If it wasn't the turn of the thread receiving the notification, without a `while` statement, the thread would proceed upon returning from the call to `wait()`. The `while` statement causes the thread to re-check the condition that it was waiting for.

7.5 The solution to the readers and writers problem does not prevent waiting writers from starving: If the database is currently being read and there is a waiting writer, subsequent readers will be allowed to read the database before the writer can write. Modify the solution such that it does not starve waiting writers.

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

7.6 A monitor-based solution to the dining-philosophers problem, written in Java-like pseudocode and using condition variables, was given in Section 7.7. Develop a solution to the dining-philosophers problem using Java synchronization.

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

7.7 The solution that we gave 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 such that philosopher_2 could never eat. Using Java synchronization, develop a solution to the dining-philosophers problem that prevents a philosopher from starving.

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

7.8 In Section 7.4, we mentioned that disabling interrupts frequently could affect the system's clock. Explain why it could, and how such effects could be minimized.

Answer: No Answer

7.9 In this chapter, we used the `synchronized` statement with *instance* methods. Calling an instance method requires associating the method with an object. Entering a `synchronized` method requires owning the object's lock. *Static* methods are unlike instance methods in that they do not require association with an object when they are called. Explain how it is possible for static methods also to be declared as `synchronized`.

Answer: This will most likely require some outside research on the student part. A good reference is Oaks and Wong [1999]. Just as each object has an associated lock, each class also has an associated lock. Calling a static synchronized method requires owning the lock for the class. However, the `wait()` and `notify()` methods cannot be called from within a static method as they are non-static methods of the `Object` class. To use `wait()` and `notify()` in a static method requires instantiating another object that will be used to call `wait()` and `notify()`.

7.10 *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 using Java synchronization.

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

7.11 *The Cigarette-Smokers Problem*. Consider a system with three *smoker* processes and one *agent* process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper, and matches. One of the smoker processes has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats. Write a program to synchronize the agent and the smokers using Java synchronization.

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

7.12 Give the reasons why Solaris 2 and Windows NT implement multiple locking mechanisms. Describe the circumstances under which they use spinlocks, mutexes, semaphores, adaptive mutexes, and condition variables. In each case, explain why the mechanism is needed.

Answer: These operating systems provide different locking mechanisms depending on the application developers needs. Spinlocks are useful for multiprocessor systems where a thread can run in a busy-loop (for a short period of time) rather than incurring the overhead of being put in a sleep queue. Mutexes are useful for locking resources. Solaris 2 uses adaptive mutexes, meaning that the mutex is implemented with a spin lock on multiprocessor machines. Semaphores and condition variables are more appropriate tools for synchronization when a resource must be held for a long period of time for spinning is inefficient for a long duration.

Chapter 8

DEADLOCKS

Deadlock is a problem that can only arise 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.

■ Answers to Exercises

8.1 List three examples of deadlocks that are not related to a computer-system environment.
Answer:

- Two cars crossing a single-lane bridge from opposite directions.
- A person going down a ladder while another person is climbing up the ladder.
- Two trains traveling toward each other on the same track.

8.2 Is it possible to have a deadlock involving only one single process? Explain your answer.
Answer: No. This follows directly from the hold-and-wait condition.

8.3 Consider the traffic deadlock depicted in Figure 8.11.

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

b. State a simple rule that will avoid deadlocks in this system.

8.4 Suppose that a system is in an unsafe state. Show that it is possible for the processes to complete their execution without entering a deadlock state.

Answer: No answer.

8.5 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 locks for five Java objects *A*..*E*, deadlock is possible. We can prevent the deadlock by adding a sixth object *F*. Whenever a thread wants to acquire the lock for any object *A*..*E*, it must first acquire the lock for object *F*. This solution is known as **containment**: The locks for objects *A*..*E* are contained within the lock for object *F*. Compare this scheme with the circular-wait scheme of Section 8.4.4.

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.

8.6 Write a Java program that illustrates deadlock by having `synchronized` methods calling other `synchronized` methods.

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

8.7 Write a Java program that illustrates deadlock by having separate threads attempting to perform operations on different semaphores.

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

8.8 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.

8.9 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.

8.10 Can a system detect that some of its processes are starving? If you answer "yes," explain how it can. If you answer "no," explain how the system can deal with the starvation problem.

Answer: No answer.

8.11 Consider the following resource-allocation policy. Requests and releases for resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check any processes that are blocked, waiting for resources. If they have the desired resources, then these resources are taken away from them and are given to the requesting process. The vector of resources for which the process is waiting is increased to include the resources that were taken away.

For example, consider a system with three resource types and the vector *Available* initialized to (4,2,2). If process P_0 asks for (2,2,1), it gets them. If P_1 asks for (1,0,1), it gets them. Then, if P_0 asks for (0,0,1), it is blocked (resource not available). If P_2 now asks for (2,0,0), it gets the available one (1,0,0) and one that was allocated to P_0 (since P_0 is blocked). P_0's *Allocation* vector goes down to (1,2,1), and its *Need* vector goes up to (1,0,1).

a. Can deadlock occur? If you answer "yes", give an example. If you answer "no," specify which necessary condition cannot occur.

b. Can indefinite blocking occur? Explain your answer.

Answer:

a. Deadlock cannot occur because preemption exists.

b. Yes. A process may never acquire all the resources it needs if they are continuously preempted by a series of requests such as those of process *C*.

8.12 A railroad tunnel with a single set of tracks conects two Vermont villages. The railroad can become deadlocked if both a northbound and southbound train enter the tunnel at the same time (the trains are unable to back up). Write a Java program that prevents deadlock using either semaphores or Java synchronization. Initially, do not be concerned about northbound trains starving southbound trains from using the tunnel (or vice versa) (and do not be concerned about trains failing to stop and crashing into each other). Once your solution prevents deadlock, modify it so that it is starvation-free.

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

Chapter 9

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.

Answers to Exercises

9.1 Name two differences between logical and physical addresses.
Answer: No answer.

9.2 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.

9.3 Describe the following allocation algorithms:

a. First fit

b. Best fit

c. Worst fit

Answer:

a. First-fit: search the list of available memory and allocate the first block that is big enough.

b. Best-fit: search the entire list of available memory and allocate the smallest block that is big enough.

c. Worst-fit: search the entire list of available memory and allocate the largest block. (The justification for this scheme is that the leftover block produced would be larger and potentially more useful than that produced by the best-fit approach.)

9.4 When a process is rolled out of memory, it loses its ability to use the CPU (at least for a while). Describe another situation where a process loses its ability to use the CPU, but where the process does not get rolled out.

Answer: When an interrupt occurs.

9.5 Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (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.

9.6 Consider a system where a program can be separated into two parts: code and data. The CPU knows whether it wants an instruction (instruction fetch) or data (data fetch or store). Therefore, two base–limit register pairs are provided: one for instructions and one for data. The instruction base–limit register pair is automatically read-only, so programs can be shared among different users. Discuss the advantages and disadvantages of this scheme.

Answer: The major advantage of this scheme is that it is an effective mechanism for code and data sharing. For example, only one copy of an editor or a compiler needs to be kept in memory, and this code can be shared by all processes needing access to the editor or compiler code. Another advantage is protection of code against erroneous modification. The only disadvantage is that the code and data must be separated, which is usually adhered to in a compiler-generated code.

9.7 Why are page sizes always powers of 2?

Answer: Recall that paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

9.8 Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.

a. How many bits are there in the logical address?

b. How many bits are there in the physical address?

Answer:

a. Logical address: 13 bits

b. Physical address: 15 bits

9.9 Why is it that, on a system with paging, a process cannot access memory it does not own? 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.

9.10 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.

9.11 What is the effect of allowing two entries in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory from one place to another. What effect would updating some byte on the one page have on the other page?

Answer: By allowing two entries in a page table to point to the same page frame in memory, users can share code and data. If the code is reentrant, much memory space can be saved through the shared use of large programs such as text editors, compilers,and database systems. "Copying" large amounts of memory could be effected by having different page tables point to the same memory location.

However, sharing of nonreentrant code or data means that any user having access to the code can modify it and these modifications would be reflected in the other user's "copy."

9.12 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.

9.13 Describe a mechanism by which one segment could belong to the address space of two different processes.

Answer: Since segment tables are a collection of base–limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. The two segment tables must have identical base pointers, and the shared segment number must be the same in the two processes.

9.14 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.

9.15 Sharing segments among processes without requiring the same segment number is possible in a dynamically linked segmentation system.

a. Define a system that allows static linking and sharing of segments without requiring that the segment numbers be the same.

b. Describe a paging scheme that allows pages to be shared without requiring that the page numbers be the same.

Answer: Both of these problems reduce to a program being able to reference both its own code and its data without knowing the segment or page number associated with the address. MULTICS solved this problem by associating four registers with each process. One register had the address of the current program segment, another had a base address for the stack, another had a base address for the global data, and so on. The idea is that all references have to be indirect through a register that maps to the current segment or page number. By changing these registers, the same code can execute for different processes without the same page or segment numbers.

9.16 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

9.17 Consider the Intel address translation scheme shown in Figure 9.20.

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?

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.

9.18 In the IBM/370, memory protection is provided through the use of *keys*. A key is a 4-bit quantity. Each 2K block of memory has a key (the storage key) associated with it. The CPU also has a key (the protection key) associated with it. A store operation is allowed only if both keys are equal, or if either is zero. Which of the following memory-management schemes could be used successfully with this hardware?

a. Bare machine

b. Single-user system

c. Multiprogramming with a fixed number of processes

d. Multiprogramming with a variable number of processes

e. Paging

f. Segmentation

Answer:

a. Protection not necessary, set system key to 0.
b. Set system key to 0 when in supervisor mode.
c. Region sizes must be fixed in increments of 2k bytes, allocate key with memory blocks.
d. Same as above.
e. Frame sizes must be in increments of 2k bytes, allocate key with pages.
f. Segment sizes must be in increments of 2k bytes, allocate key with segments.

Chapter 10

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.

■ Answers to Exercises

10.1 Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.

Answer: A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/Ois requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

10.2 Assume a page reference string for a process with m frames (initially all empty). The page reference string has length p with n distinct page numbers occurring in it. For any page-replacement algorithms,

a. What is a lower bound on the number of page faults?

b. What is an upper bound on the number of page faults?

Answer:

a. n

b. p

10.3 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.

10.4 Which of the following programming techniques and structures are "good" for a demand-paged environment ? Which are "not good"? Explain your answers.

a. Stack
b. Hashed symbol table
c. Sequential search
d. Binary search
e. Pure code
f. Vector operations
g. Indirection

Answer:

a. Stack—good.
b. Hashed symbol table—not good.
c. Sequential search—good.
d. Binary search—not good.
e. Pure code—good.
f. Vector operations—good.
g. Indirection—not good.

10.5 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:

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

10.6 Consider the following page-replacement algorithms. Rank these algorithms on a five-point scale from "bad" to "perfect" according to their page-fault rate. Separate those algorithms that suffer from Belady's anomaly from those that do not.

a. LRU replacement

b. FIFO replacement

c. Optimal replacement

d. Second-chance replacement

Answer:

Rank	Algorithm	Suffer from Belady's anomaly
1	Optimal	no
2	LRU	no
3	Second-chance	yes
4	FIFO	yes

10.7 When virtual memory is implemented in a computing system, there are certain costs associated with the technique and certain benefits. List the costs and the benefits. Is it possible for the costs to exceed the benefits? If it is, what measures can be taken to ensure that this does not happen?
Answer: The costs are additional hardware and slower access time. The benefits are good utilization of memory and larger logical address space than physical address space.

10.8 An operating system supports a paged virtual memory, using a central processor with a cycle time of 1 microsecond. It costs an additional 1 microsecond to access a page other than the current one. Pages have 1000 words, and the paging device is a drum that rotates at 3000 revolutions per minute and transfers 1 million words per second. The following statistical measurements were obtained from the system:

- 1 percent of all instructions executed accessed a page other than the current page.
- Of the instructions that accessed another page, 80 percent accessed a page already in memory.
- When a new page was required, the replaced page was modified 50 percent of the time.

Calculate the effective instruction time on this system, assuming that the system is running one process only, and that the processor is idle during drum transfers.
Answer:

$$
\begin{aligned}
\text{effective access time} &= 0.99 \times (1\ \mu\text{sec} + 0.008 \times (2\ \mu\text{sec}) \\
&\quad + 0.002 \times (10{,}000\ \mu\text{sec} + 1{,}000\ \mu\text{sec}) \\
&\quad + 0.001 \times (10{,}000\ \mu\text{sec} + 1{,}000\ \mu\text{sec}) \\
&= (0.99 + 0.016 + 22.0 + 11.0)\ \mu\text{sec} \\
&= 34.0\ \mu\text{sec}
\end{aligned}
$$

10.9 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. Get a faster CPU—No.

b. Get a bigger paging drum—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.

10.10 Consider the two-dimensional array `A`:

```
int A[][] = new int[100][100];
```

where `A[0][0]` is at location 200, in a paged system with pages of size 200. A small process is in page 0 (locations 0 to 199) for manipulating the matrix; thus, every instruction fetch will be from page 0.

For three page frames, how many page faults are generated by the following array-initialization loops, using LRU replacement, and assuming page frame 1 has the process in it, and the other two are initially empty:

a.
```
for (int j = 0; j < 100; j++)
for (int i = 0; i < 100; i++)
A[i][j] = 0;
```

b.
```
for (int i = 0; i < 100; i++)
for (int j = 0; j < 100; j++)
A[i][j] = 0;
```

Answer: o answer.

10.11 Consider the following page reference string:

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

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

Answer:

Number of frames	LRU	FIFO	Optimal
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7

10.12 Suppose that you want to use a paging algorithm that requires a reference bit (such as second-chance replacement or working-set model), but the hardware does not provide one. Sketch how you could simulate a reference bit even if one were not provided by the hardware, or explain why it is not possible to do so. If it is possible, calculate what the cost would be.

Answer: You can use the valid / invalid bit supported in hardware to simulate the reference bit. Initially set the bit to invalid. On first reference a trap to the operating system is generated. The operating system will set a software bit to 1 and reset the valid / invalid bit to valid.

10.13 You have devised a new page-replacement algorithm that you think may be optimal. In some contorted test cases, Belady's anomaly occurs. Is the new algorithm optimal? Explain your answer.

Answer: No answer.

10.14 Suppose that your replacement policy (in a paged system) is to examine each page regularly and to discarding 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: No answer

10.15 Segmentation is similar to paging but uses variable-sized "pages." Define two segment-replacement algorithms based on FIFO and LRU page-replacement schemes. Remember that since segments are not the same size, the segment that is chosen to be replaced may not be big enough to leave enough consecutive locations for the needed segment. Consider strategies for systems where segments cannot be relocated, and those for systems where they can.

Answer:

- **FIFO**. Find the first segment large enough to accommodate the incoming segment. If relocation is not possible and no one segment is large enough, select a combination of segments whose memories are contiguous, which are "closest to the first of the list" and which can accommodate the new segment. If relocation is possible, rearrange the memory so that the first N segments large enough for the incoming segment are contiguous in memory. Add any leftover space to the free-space list in both cases.
- **LRU**. Select the segment that has not been used for the longest period of time and that is large enough, adding any leftover space to the free space list. If no one segment is large enough, select a combination of the "oldest" segments that are contiguous in memory (if relocation is not available) and that are large enough. If relocation is available, rearrange the oldest N segments to be contiguous in memory and replace those with the new segment.

10.16 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:

   i. Initial value of the counters—0.

   ii. Counters are increased—whenever a new page is associated with that frame.

   iii. Counters are decreased—whenever one of the pages associated with that frame is no longer required.

   iv. 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

10.17 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 the remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Answer:

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

10.18 Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of CPU and the paging disk. The results are one of the following alternatives. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization? Is the paging helping?

a. CPU utilization 13 percent; disk utilization 97 percent

b. CPU utilization 87 percent; disk utilization 3 percent

c. CPU utilization 13 percent; disk utilization 3 percent

Answer:

a. Thrashing is occurring.

b. CPU utilization is sufficiently high to leave things alone, an increase degree of multiprogramming.

c. Increase the degree of multiprogramming.

10.19 We have an operating system for a machine that uses base and limit registers, but we have modified the machine to provide a page table. Can the page tables be set up to simulate base and limit registers? How can they be, or why can they not be?

Answer: The page table can be set up to simulate base and limit registers provided that the memory is allocated in fixed-size segments. In this way, the base of a segment can be entered into the page table and the valid / invalid bit used to indicate that portion of the segment as resident in the memory. There will be some problem with internal fragmentation.

10.20 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.

10.21 Write a Java 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..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 such that the number of page frames can vary from 1..7. Assume that demand paging is used.

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

Chapter 11

FILE SYSTEMS

Files are the most obvious object that operating systems manipulate. Everything is typically stored in files: programs, data, output, and so on. 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 with 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.

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.

■ Answers to Exercises

11.1 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 ensuring 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.

11.2 Some systems automatically delete all user files when a user logs off or a job terminates, unless the user explicitly requests that they be kept; other systems keep all files unless the user explicitly deletes them. Discuss the relative merits of each approach.

Answer: Deleting all files not specifically saved by the user has the advantage of minimizing the file space needed for each user by not saving unwanted or unnecessary files. Saving all files unless specifically deleted is more secure for the user in that it is not possible to inadvertently lose files by forgetting to save them.

11.3 Why do some systems keep track of the type of a file, while others leave it to the user or simply do not implement multiple file types? Which system is "better?"

Answer: Some systems allow different file operations based on the type of the file (for instance, an ascii file can be read as a stream while a database file can be read via an index to a block). Other systems leave such interpretation of a file's data to the process and provide no help in accessing the data. The method which is "better" depends on the needs of the processes on the system, and the demands the users place on the operating system. If a system runs mostly database applications, it may be more efficient for the operating system to implement a database-type file and provide operations rather than making each program implement the same thing (possibly in different ways). For general-purpose systems it may be better to only implement basic file types to keep the operating system size smaller and allow maximum freedom to the processes on the system.

11.4 Similarly, some systems support many types of structures for a file's data, while others simply support a stream of bytes. What are the advantages and disadvantages?

Answer: (See Section 11.3.)

11.5 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.

11.6 Could you simulate a multilevel directory structure with a single-level directory structure in which arbitrarily long names could be used? If your answer is yes, explain how you can do so, and contrast this scheme with the multilevel directory scheme. If your answer is no, explain what prevents your simulation's success. How would your answer change if file names were limited to seven characters?

Answer: If arbitrarily long names can be used, then it is possible to simulate a multilevel directory structure. This can be done, for example, by using the character "." to indicate the end of a subdirectory. Thus, for example, the name *jim.pascal.F1* specifies that *F1* is a file in subdirectory *pascal,* which in turn is in the root directory *jim*.

If file names were limited to seven characters, then the above scheme could not be utilized and thus, in general, the answer is *no*. The next best approach in this situation would be to use a specific file as a symbol table (directory) to map arbitrarily long names (such as *jim.pascal.F1*) into shorter arbitrary names (such as *XX00743*), which are then used for actual file access.

11.7 Explain the purpose of the `open` and `close` operations.

Answer:

- The `open` operation informs the system that the named file is about to become active.

- The `close` operation informs the system that the named file is no longer in active use by the user who issued the close operation.

11.8 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 when explicit opening and closing is required.

11.9 Give an example of an application in which data in a file should be accessed in the following order:

a. Sequentially

b. Randomly

Answer:

a. Print the content of the file.

b. Print the content of record *i*. This record can be found using hashing or index techniques.

11.10 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 the 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.

11.11 In some systems, a subdirectory can be read and written by an authorized user, just as ordinary files can be.

a. Describe the protection problems that could arise.

b. Suggest a scheme for dealing with each of the protection problems you named in part a.

Answer:

a. One piece of information kept in a directory entry is file location. If a user could modify this location, then he could access other files defeating the access-protection scheme.

b. Do not allow the user to directly write onto the subdirectory. Rather, provide system operations to do so.

11.12 Consider a system that supports 5000 users. Suppose that you want to allow 4990 of these users to be able to access one file.

a. How would you specify this protection scheme in UNIX?

b. Could you suggest another protection scheme that could be used more effectively for this purpose than the scheme provided by UNIX?

Answer:

a. There are two methods for achieving this:
   i. Create an access control list with the names of all 4990 users.
   ii. Put these 4990 users in one group and set the group access accordingly. This scheme cannot always be implemented since user groups are restricted by the system.

b. The universe access information applies to all users unless their name appears in the access-control list with different access permission. With this scheme you simply put the names of the remaining ten users in the access control list but with no access privileges allowed.

11.13 Researchers have suggested that, instead of having an access list associated with each file (specifying which users can access the file, and how), we should have a *user control list* associated with each user (specifying which files a user can access, and how). Discuss the relative merits of these two schemes.

Answer:

- *File control list*. Since the access control information is concentrated in one single place, it is easier to change access control information and this requires less space.
- *User control list*. This requires less overhead when opening a file.

11.14 Consider a file currently consisting of 100 blocks. Assume that the file control block (and the index block, in the case of indexed allocation) is already in memory. Calculate how many disk I/O operations are required for contiguous, linked, and indexed (single-level) allocation strategies, if, for one block, the following conditions hold. In the contiguous-allocation case, assume that there is no room to grow in the beginning, but there is room to grow in the end. Assume that the block information to be added is stored in memory.

a. The block is added at the beginning.
b. The block is added in the middle.
c. The block is added at the end.
d. The block is removed from the beginning.
e. The block is removed from the middle.
f. The block is removed from the end.

Answer:

	Contiguous	Linked	Indexed
a.	201	1	1
b.	101	52	1
c.	1	3	1
d.	198	1	0
e.	98	52	0
f.	0	100	0

11.15 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. 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 were 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.

11.16 What problems could occur if a system allowed a file system to be mounted simultaneously at more than one location?

Answer: There would be multiple paths to the same file, which could confuse users or encourage mistakes. (Deleting a file with one path deletes the file in all the other paths.)

11.17 Why must the bit map for file allocation be kept on mass storage rather than in main memory?

Answer: In case of system crash (memory failure), the free-space list would not be lost as it would be if the bit map had been stored in main memory.

11.18 Consider a system that supports the strategies of contiguous, linked, and indexed allocation. What criteria should be used in deciding which strategy is best utilized for a particular file?

Answer:

- **Contiguous**—if file is usually accessed sequentially, if file is relatively small.
- **Linked**—if file is large and usually accessed sequentially.
- **Indexed**—if file is large and usually accessed randomly.

11.19 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.

   i. Add X to Z to obtain the physical block number. Y is the displacement into that block.

   ii. 1

b. Linked. Divide the logical physical address by 511 with X and Y the resulting quotient and remainder respectively.

   i. Chase down the linked list (getting $X + 1$ blocks). $Y + 1$ is the displacement into the last physical block.

   ii. 4

c. Indexed. Divide the logical address by 512, with X and Y the resulting quotient and remainder, respectively.
   i. 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.
   ii. 2

11.20 One problem with contiguous allocation is that the user must preallocate enough space for each file. If the file grows to be larger than the space allocated for it, special actions must be taken. One solution to this problem is to define a file structure consisting of an initial contiguous area (of a specified size). If this area is filled, the operating system automatically defines an overflow area that is linked to the initial contiguous area. If the overflow area is filled, another overflow area is allocated. Compare this implementation of a file with the standard contiguous and linked implementations.

Answer: This method requires more overhead then the standard contiguous allocation. It requires less overhead than the standard linked allocation.

11.21 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 would 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.

11.22 How do caches help improve performance? Why do systems not use more or larger caches if they are so useful?

Answer: Caches allow components of differing speeds to communicate more efficiently by storing data from the slower device, temporarily, in a faster device (the cache). Caches are, almost by definition, more expensive than the device they are caching for, so increasing the number or size of caches would increase system cost.

11.23 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.

11.24 Why is it advantageous for the user of an operating system to dynamically allocate its internal tables? What are the penalties to the operating system for doing so?

Answer: Dynamic tables allow more flexibility in system use growth—tables are never exceeded, avoiding artificial use limits. Unfortunately, kernel structures and code are more complicated, so there is more potential for bugs. The use of one resource can take away more system resources (by growing to accommodate the requests) than with static tables.

11.25 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 11.10.2 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 11.10.2? 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.

Chapter 12

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.

■ Answers to Exercises

12.1 State three advantages of placing functionality in a device controller rather than in the kernel. State three disadvantages.

Answer: Three advantages: Bugs are less likely to cause an operating system crash.

Performance can be improved by utilizing dedicated hardware and hard-coded algorithms.

The kernel is simplified by moving algorithms out of it.

Three disadvantages: Bugs are harder to fix—a new firmware version or new hardware is needed

Improving algorithms likewise requires a hardware update rather than just a kernel or device driver update.

Embedded algorithms could conflict with the application's use of the device, causing decreased performance.

12.2 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 differences 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 vice 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 only useful for input and for I/O completion detection, neither of which is needed for a memory-mapped device.

12.3 The example of handshaking in Section 12.1 used 2 bits: a `busy` bit and a `command-ready` bit. Is it possible to implement this handshaking with only 1 bit? If it is, describe the protocol. If it is not, explain why 1 bit is insufficient.

Answer: No answer.

12.4 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 only be waiting 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.

12.5 Why might a system use interrupt-driven I/O to manage a single serial port but polling I/O to manage a front-end processor, such as a terminal concentrator?

Answer: Polling can be more efficient than interrupt-driven I/O. This is the case when the I/O is frequent and of short duration. Even though a single serial port will perform I/O relatively infrequently and should thus use interrupts, a collection of serial ports such as those in a terminal concentrator can produce a lot of short I/O operations, and interrupting for each one could create a heavy load on the system. A well-timed polling loop could alleviate that load without wasting many resources through looping with no I/O needed.

12.6 Polling for an I/O completion can waste a large number of CPU cycles if the processor iterates a busy-waiting loop many times before the I/O completes. But if the I/O device is ready for service, polling can be much more efficient than is catching and dispatching an interrupt. Describe a hybrid strategy that combines polling, sleeping, and interrupts for I/O device service. For each of these three strategies (pure polling, pure interrupts, hybrid), describe a computing environment in which that strategy is more efficient than is either of the others.

Answer: No answer.

12.7 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

12.8 How does DMA increase system concurrency? How does it complicate hardware design?

Answer:

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

12.9 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: No answer.

12.10 Why is it important to scale up system bus and device speeds as the CPU speed increases?

Answer:

Consider a system that performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%. Doubling both system aspects would increase performance by 100%. Generally, it is important to remove the current system bottleneck and to increase overall system performance rather than blindly increase the performance of individual system components.

Chapter 13

MASS STORAGE STRUCTURE

In this chapter we describe the internal data structures and algorithms used by the operating system to implement this interface. 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 13.7 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.

■ Answers to Exercises

13.1 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 *un*fair 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.

13.2 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.

13.3 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 13.2 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 13.2. 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.

13.4 Suppose that the disk in Exercise 13.3 rotates at 7200 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. 7200 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.

13.5 The accelerating seek described in Exercise 13.3 is typical of hard-disk drives. By contrast, floppy disks (and many hard disks manufactured before the mid-1980s) typically seek at a fixed rate. Suppose that the disk in Exercise 13.3 has a constant-rate seek rather than a constant-acceleration seek, so the seek time is of the form $t = x + yL$, where t is the time in milliseconds and L is the seek distance. Suppose that the time to seek to an adjacent cylinder is 1 millisecond, as before, and is 0.5 milliseconds for each additional cylinder.

a. Write an equation for this seek time as a function of the seek distance.

b. Using the seek-time function from part a, calculate the total seek time for each of the schedules in Exercise 13.2. Is your answer the same as it was for Exercise 13.3(c)?

c. What is the percentage speedup of the fastest schedule over FCFS in this case?

Answer:

a. $t = 0.95 + 0.05L$

b. FCFS 362.60; SSTF 95.80; SCAN 497.95; LOOK 174.50; C-SCAN 500.15; (and C-LOOK 176.70). SSTF is still the winner, and LOOK is the runner-up.

c. $(362.60 - 95.80)/362.60 = 0.74$ The percentage speedup of SSTF over FCFS is 74%, with respect to the seek time. If we include the overhead of rotational latency and data transfer, the percentage speedup will be less.

13.6 Write a Java program for disk scheduling using the SCAN and C-SCAN disk-scheduling algorithms.

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

13.7 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 only causes 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.

13.8 Is disk scheduling, other than FCFS scheduling, useful in a single-user environment? Explain your answer.

Answer: In a single-user environment, the I/O queue usually is empty. Requests generally arrive from a single process for one block or for a sequence of consecutive blocks. In these cases, FCFS is an economical method of disk scheduling. But LOOK is nearly as easy to program and will give much better performance when multiple processes are performing concurrent I/O, such as when a Web browser retrieves data in the background while the operating system is paging and another application is active in the foreground.

13.9 Explain why SSTF scheduling tends to favor middle cylinders over the innermost and outermost cylinders.

Answer: The center of the disk is the location having the smallest average distance to all other tracks. Thus the disk head tends to move away from the edges of the disk. Here is another way to think of it. The current location of the head divides the cylinders into two groups. If the head is not in the center of the disk and a new request arrives, the new request is more likely to be in the group that includes the center of the disk;, thus, the head is more likely to move in that direction.

13.10 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 only contains 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*.)

13.11 Why is rotational latency usually not considered in disk scheduling? How would you modify SSTF, SCAN, and C-SCAN to include latency optimization?

Answer: Most disks do not export their rotational position information to the host. Even if they did, the time for this information to reach the scheduler would be subject to imprecision and the time consumed by the scheduler is variable, so the rotational position information would become incorrect. Further, the disk requests are usually given in terms of logical block numbers, and the mapping between logical blocks and physical locations is very complex.

13.12 How would use of a RAM disk affect your selection of a disk-scheduling algorithm? What factors would you need to consider? Do the same considerations apply to hard-disk scheduling, given that the file system stores recently used blocks in a buffer cache in main memory?

Answer: Disk scheduling attempts to reduce the overhead time of disk head positioning. Since a RAM disk has uniform access times, scheduling is largely unnecessary. The comparison between RAM disk and the main memory disk-cache has no implications for hard-disk scheduling because we only schedule the buffer cache misses, not the requests that find their data in main memory.

13.13 Why is it important to balance file system I/O among the disks and controllers on a system in a multitasking environment?

Answer: A system can only perform at the speed of its slowest bottleneck. Disks or disk controllers are frequently the bottleneck in modern systems as their individual performance cannot keep up with that of the CPU and system bus. By balancing I/O among disks and controllers, neither an individual disk nor a controller is overwhelmed, so that bottleneck is avoided.

13.14 What are the tradeoffs involved in rereading code pages from the file system versus using swap space to store them?

Answer: If code pages are stored in swap space, they can be transferred more quickly to main memory (because swap space allocation is tuned for faster performance than general file system allocation). Using swap space can require startup time if the pages are copied there at process invocation rather than just being paged out to swap space on demand. Also, more swap space must be allocated if it is used for both code and data pages.

13.15 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.

13.16 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 5 years. If such a drive truly has a million-hour MTBF, it is very unlikely that the drive will fail during its expected lifetime.

13.17 The term “fast wide SCSI-II” denotes a SCSI bus that operates at a data rate of 20 megabytes per second when it moves a packet of bytes between the host and a device. Suppose that a fast wide SCSI-II disk drive spins at 7200 RPM, has a sector size of 512 bytes, and holds 160 sectors per track.

a. Estimate the sustained transfer rate of this drive in megabytes per second.

b. Suppose that the drive has 7000 cylinders, 20 tracks per cylinder, a head switch time (from one platter to another) of 0.5 millisecond, and an adjacent cylinder seek time of 2 milliseconds. Use this additional information to give an accurate estimate of the sustained transfer rate for a huge transfer.

c. Suppose that the average seek time for the drive is 8 milliseconds. Estimate the I/Os per second and the effective transfer rate for a random-access workload that reads individual sectors that are scattered across the disk.

d. Calculate the random-access I/Os per second and transfer rate for I/O sizes of 4 kilobytes, 8 kilobytes, and 64 kilobytes.

e. If multiple requests are in the queue, a scheduling algorithm such as SCAN should be able to reduce the average seek distance. Suppose that a random-access workload is reading 8-kilobyte pages, the average queue length is 10, and the scheduling algorithm reduces the average seek time to 3 milliseconds. Now calculate the I/Os per second and the effective transfer rate of the drive.

Answer:

a. The disk spins 120 times per second, and each spin transfers a track of 80 KB. Thus, the sustained transfer rate can be approximated as 9600 KB/s.

b. Suppose that 100 cylinders is a huge transfer. The transfer rate is total bytes divided by total time. Bytes: 100 cyl * 20 trk/cyl * 80 KB/trk, i.e., 160,000 KB. Time: rotation time + track switch time + cylinder switch time. Rotation time is 2000 trks / 120 trks_per_sec, i.e., 16.667 s. Track switch time is 19 switch_per_cyl * 100 cyl * 0.5 ms, i.e., 950 ms. Cylinder switch time is 99 * 2 ms, i.e., 198 ms. Thus, the total time is 16.667 + 0.950 + 0.198, i.e., 17.815 s. (We are ignoring any initial seek and rotational latency, which might add about 12 ms to the schedule, i.e. 0.1%.) Thus the transfer rate is 8981.2 KB/s. The overhead of track and cylinder switching is about 6.5%.

c. The time per transfer is 8 ms to seek + 4.167 ms average rotational latency + 0.052 ms (calculated from 1 / (120 trk_per_second * 160 sector_per_trk)) to rotate one sector past the disk head during reading. We calculate the transfers per second as 1/(0.012219), i.e., 81.8. Since each transfer is 0.5 KB, the transfer rate is 40.9 KB/s.

d. We ignore track and cylinder crossings for simplicity. For reads of size 4 KB, 8 KB, and 64 KB, the corresponding I/Os per second are calculated from the seek, rotational latency, and rotational transfer time as in the previous item, giving (respectively) 1/(0.0126), 1/(0.013), and 1/(0.019). Thus we get 79.4, 76.9, and 52.6 transfers per second, respectively. Transfer rates are obtained from 4, 8, and 64 times these I/O rates, giving 318 KB/s, 615 KB/s, and 3366 KB/s, respectively.

e. From 1/(3+4.167+0.83) we obtain 125 I/Os per second. From 8 KB per I/O we obtain 1000 KB/s.

13.18 More than one disk drive can be attached to a SCSI bus. In particular, a fast wide SCSI-II bus (see Exercise 13.17) can be connected to at most 15 disk drives. Recall that this bus has a bandwidth of 20 megabytes per second. At any time, only one packet can be transferred on the bus between some disk's internal cache and the host. However, a disk can be moving its disk arm while some other disk is transferring a packet on the bus. Also, a disk can be transferring data between its magnetic platters and its internal cache while some other disk is transferring a packet on the bus. Considering the transfer rates that you calculated for the various workloads in Exercise 13.17, discuss how many disks can be used effectively by one fast wide SCSI-II bus.

Answer:

For 8 KB random I/Os on a lightly loaded disk, where the random access time is calculated to be about 13 ms (see Exercise 13.17), the effective transfer rate is about 615 MB/s. In this case, 15 disks would have an aggregate transfer rate of less than 10 MB/s, which should not saturate the bus. For 64 KB random reads to a lightly loaded disk, the transfer rate is about 3.4 MB/s, so 5 or fewer disk drives would saturate the bus. For 8 KB reads with a large enough queue to reduce the average seek to 3 ms, the transfer rate is about 1 MB/s, so the bus bandwidth may be adequate to accommodate 15 disks.

13.19 Remapping of bad blocks by sector sparing or sector slipping could influence performance. Suppose that the drive in Exercise 13.17 has a total of 100 bad sectors at random locations and that each bad sector is mapped to a spare that is located on a different track, but within the same cylinder. Estimate the number of I/Os per second and the effective transfer rate for a random-access workload consisting of 8-kilobyte reads, with a queue length of 1 (that is, the choice of scheduling algorithm is not a factor). What is the effect of a bad sector on performance?

Answer:

Since the disk holds 22,400,000 sectors, the probability of requesting one of the 100 remapped sectors is very small. An example of a worst-case event is that we attempt to read, say, an 8 KB page, but one sector from the middle is defective and has been remapped to the worst possible location on another track in that cylinder. In this case, the time for the retrieval could be 8 ms to seek, plus two track switches and two full rotational latencies. It is likely that a modern controller would read all the requested good sectors from the original track before switching to the spare track to retrieve the remapped sector and thus would incur only one track switch and rotational latency. So the time would be 8 ms seek + 4.17 ms average rotational latency + 0.05 ms track switch + 8.3 ms rotational latency + 0.83 ms read time (8 KB is 16 sectors, 1/10 of a track rotation). Thus, the time to service this request would be 21.8 ms, giving an I/O rate of 45.9 requests per second and an effective bandwidth of 367 KB/s. For a severely time-constrained application this might matter, but the overall impact in the weighted average of 100 remapped sectors and 22.4 million good sectors is nil.

13.20 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.

13.21 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.

13.22 In a disk jukebox, what would be the effect of having more open files than the number of drives in the jukebox?

Answer: Two bad outcomes could result. One possibility is starvation of the applications that issue blocking I/Os to tapes that are not mounted in drives. Another possibility is thrashing, as the jukebox is commanded to switch tapes after every I/O operation.

13.23 What would be the effect on cost and performance if tape storage were to achieve the same areal density as disk storage?

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.

13.24 If magnetic hard disks eventually have the same cost per gigabyte as do tapes, will tapes become obsolete, or will they still be needed? Explain your answer.

Answer: Tapes are easily removable, so they are useful for off-site backups and for bulk transfer of data (by sending cartridges). As a rule, a magnetic hard disk is not a removable medium.

13.25 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, i.e., 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%, i.e., $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 1 drive. Then the aggregate data rate is 6×10 MB/s, i.e., 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 $1/20^{th}$ 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.

13.26 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: No answer.

13.27 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, i.e., $1,250.

13.28 Suppose that we agree that 1 kilobyte is 1,024 bytes, 1 megabyte is $1,024^2$ bytes, and 1 gigabyte is $1,024^3$ bytes. This progression continues through terabytes, petabytes, and exabytes ($1,024^6$). There are currently several new proposed scientific projects that plan to record and store a few exabytes of data during the next decade. To answer the following questions, you will need to make a few reasonable assumptions; state the assumptions that you make.

a. How many disk drives would be required to hold 4 exabytes of data?

b. How many magnetic tapes would be required to hold 4 exabytes of data?

c. How many optical tapes would be required to hold 4 exabytes of data (see Exercise 13.27)?

d. How many holographic storage cartridges would be required to hold 4 exabytes of data (see Exercise 13.26)?

e. How many cubic feet of storage space would each option require?

Answer:

a. Assume that a disk drive holds 10 GB. Then 100 disks hold 1 TB, 100,000 disks hold 1 PB, and 100,000,000 disks hold 1 EB. To store 4 EB would require about 400 million disks. If a magnetic tape holds 40 GB, only 100 million tapes would be required. If an optical tape holds 50 times more data than a magnetic tape, 2 million optical tapes would suffice. If a holographic cartridge can store 180 GB, about 22.2 million cartridges would be required.

b. A 3.5” disk drive is about 1” high, 4” wide, and 6” deep. In feet, this is 1/12 by 1/3 by 1/2, or 1/72 cubic feet. Packed densely, the 400 million disks would occupy 5.6 million cubic feet. If we allow a factor of 2 for air space and space for power supplies, the required capacity is about 11 million cubic feet.

c. A 1/2” tape cartridge is about 1” high and 4.5” square. The volume is about 1/85 cubic feet. For 100 million magnetic tapes packed densely, the volume is about 1.2 million cubic feet. For 2 million optical tapes, the volume is 23,400 cubic feet.

d. A CD-ROM is 4.75” in diameter and about 1/16” thick. If we assume that a holostore disk is stored in a library slot that is 5” square and 1/8” wide, we calculate the volume of 22.2 million disks to be about 40,000 cubic feet.

13.29 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 13.7.2.1.

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.

Chapter 14

NETWORK STRUCTURES

There are basically two schemes for building distributed systems. In a *multiprocessor* (tightly coupled) system, the processors share memory and a clock, and communication usually takes place through the shared memory. In a *distributed* (loosely coupled) system, the processors 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. Detailed discussions are given in Chapters 16 to 18.

■ Answers to Exercises

14.1 What are two main differences between a WAN and a LAN?

Answer: (1) Distance covered. A WAN can cover hundreds or even thousands of miles, a LAN typically covers an area less than a kilometer (although it is often much less.) (2) Speed. WANs typically run at about 56 Kbps, and modern LANs can run up to 1 gigabit.

14.2 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: No Answer

14.3 Explain why a doubling of the speed of the systems on an Ethernet segment may result in decreased network performance. What changes could be made to ameliorate the problem?

Answer: Faster systems may be able to send more packets in a shorter amount of 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.

14.4 Under what circumstances is a token-ring network more effective than an Ethernet network?

Answer: A token ring is very effective under high sustained load, as no collisions can occur and each slot may be used to carry a message, providing high throughput. A token ring is less effective when the load is light (token processing takes longer than bus access, so any one packet can take longer to reach its destination), or sporadic.

14.5 Why would it be a bad idea for gateways to pass broadcast packets between networks? What would be the advantages of doing so?

Answer: All broadcasts would be propagated to all networks, causing a *lot* of network traffic. If broadcast traffic were limited to important data (and very little of it), then broadcast propagation would save gateways from having to run special software to watch for this data (such as network routing information) and rebroadcast it.

14.6 What are the advantages of using dedicated hardware devices for routers and gateways? What are the disadvantages over using general-purpose computers?

Answer: No answer.

14.7 In what ways is using a name server better than using static host tables? What are the problems and complications associated with name servers? What methods could be used 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.

14.8 The original *HTTP* protocol used TCP/IP as the underlying network protocol. For each page, graphic, or applet, a separate TCP session was contructed, used, and torn down. Because of the overhead of building and destroying TCP/IP connections, there were performance problems with this implementation method. Would using UDP rather than TCP have been a good alternative? What other changes could be made to improve HTTP performance?

Answer: No answer.

14.9 Of what use is an address resolution protocol? Why is the use of such a protocol better than making each host read each packet to determine to whom it is destined? Does a token-ring network need such a protocol?

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.

14.10 What are the advantages and disadvantages of making the computer network transparent to the user?

Answer: No answer.

14.11 What are two formidable problems that designers must solve to implement a network-transparent system?

Answer: No answer.

14.12 Process migration within a heterogeneous network is usually impossible, given the differences in architectures and operating systems. Describe a method for process migration across different architectures running:

a. The same operating system

b. Different operating systems

Answer: No answer.

14.13 To build a robust distributed system, you must know what kinds of failures can occur.

a. List three possible types of failure in a distributed system.

b. Specify which of the entries in your list also are applicable to a centralized system.

Answer: No answer.

14.14 Is it always crucial to know that the message you have sent has arrived at its destination safely? If your answer is “yes,” explain why. If your answer is “no,” give appropriate examples.
Answer: No answer.

14.15 Present an algorithm for reconstructing a logical ring after a process in the ring fails.
Answer: No answer.

14.16 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: No answer.

Chapter 15

DISTRIBUTED COMMUNICATION

In this chapter, we discuss several different strategies that allow distributed applications to communicate. As examples, we present Java solutions using two different communication strategies. The first is sockets, a TCP/IP scheme that allows distributed processes to communicate over a network. The second is a Java-based communication protocol that allows a thread in one Java program to invoke a remote method in another Java program. We also discuss CORBA an emerging technology.

We encourage you to have students implement a distributed solution to one of the classical synchronization problems. The programming is not difficult and often simply requires a cookbook approach. Furthermore, RMI is an evolving technology. Changes to security in different releases of Java may require running a program differently from how it is described in the text. Please read all the documentation to run the programs that are available at the supporting Web site.

■ Answers to Exercises

15.1 Write a socket-based *Fortune Teller* server. Your program should create a server that listens to a specified port. When a connection is received by a client, the server should respond with a random fortune chosen from its database of fortunes.

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

15.2 Two problems may arise when we multithread a server. The first is that we may pay a high overhead for thread creation if a server creates a separate thread whenever it receives a request. The second problem is that too many threads may be created if there are several concurrent requests to the server. (This problem is especially severe for thread architectures that use the one-to-one model.) One way of addressing these problems is with a *thread pool:* a collection (pool) of threads waiting to be assigned a task. Rather than creating a thread when a request is made, the server creates several threads up front and adds them to the pool. When the server receives a request, it removes a thread from this pool and assigns to that thread the task of handling the request. When the request has been serviced, the server puts the thread back in the pool.

Implement the time-of-day server using a thread pool.

Answer: Please refer to the supporting Web site for source code solution. Note that this is a difficult problem and will most likely require further research by the student. Good references include Oaks and Wong [1999] and Java Report [1998].

15.3 Implement Algorithm 3 from Chapter 7 using RMI.

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

15.4 This chapter provided a distributed solution to the bounded-buffer problem that used message passing. Another solution to the bounded-buffer problem used Java synchronization; it was shown in Chapter 7. Modify this second solution using RMI, such that the solution works in a distributed environment.

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

15.5 We solved the readers-and-writers problem by using Java synchronization in Chapter 7. Modify that solution such that it works in a distributed environment using RMI.

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

15.6 Implement a distributed solution to the dining-philosophers problem using RMI.

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

15.7 Explain the differences and similarities between RMI and CORBA.

Answer: They are similar in that both are object-based and that you program to an interface, essentially ignoring the implementation of the interface. The main distinction is that RMI is a Java-based technology and CORBA allows applications written in various languages (including Java) to communicate.

Chapter 16

DISTRIBUTED COORDINATION

Chapter 16 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.

■ Answers to Exercises

16.1 Your company is building a computer network, and you have been asked to write an algorithm for achieving distributed mutual exclusion. Which scheme will you use? Explain your choice.

Answer: No answer.

16.2 Why is deadlock detection much more expensive in a distributed environment than it is in a centralized environment?

Answer: No answer.

16.3 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: No answer.

16.4 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 nonleaf 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: No answer.

16.5 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: No answer.

Chapter 17

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 the UNIX United, NFS, Andrew, Sprite, and Locus distributed file systems. By exploring these example systems, 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.

■ Answers to 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 across 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 would handle a large, multiclient database application most efficiently? Explain your answer.

Answer: No answer.

17.3 Under which 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.

17.4 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.

17.5 Compare and contrast the techniques of caching disk blocks locally, on a client system, and remotely, on a server.

Answer: No answer.

17.6 What are the benefits of mapping objects into virtual memory, as Apollo Domain does? What are the detriments?

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.

Chapter 18

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.

■ Answers to Exercises

18.1 What are the main differences between capability lists and access lists?

Answer: An access list is a list for each object consisting of the domains with a nonempty set of access rights for that object. A capability list is a list of objects and the operations allowed on those objects for each domain.

18.2 A Burroughs B7000/B6000 MCP file can be tagged as sensitive data. When such a file is deleted, its storage area is overwritten by some random bits. For what purpose would such a scheme be useful?

Answer: This would be useful as an extra security measure so that the old content of memory cannot be accessed, either intentionally or by accident, by another program. This is especially useful for any highly classified information.

18.3 In a ring-protection system, level 0 has the greatest access to objects, and level n (greater than zero) has fewer access rights. The access rights of a program at a particular level in the ring structure are considered as a set of capabilities. What is the relationship between the capabilities of a domain at level j and a domain at level i to an object (for $j > i$)?

Answer: D_j is a subset of D_i.

18.4 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).

18.5 The RC 4000 system (and other systems) have defined a tree of processes (called a process tree) such that all the descendants of a process are given resources (objects) and access rights by their ancestors only. Thus, a descendant can never have the ability to do anything that its ancestors cannot do. The root of the tree is the operating system, which has the ability to do anything. Assume the set of access rights was represented by an access matrix, A. $A(x,y)$ defines the access rights of process x to object y. If x is a descendant of z, what is the relationship between $A(x,y)$ and $A(z,y)$ for an arbitrary object y?

Answer: $A(x,y)$ is a subset of $A(z,y)$.

18.6 What hardware features are needed for efficient capability manipulation? Can these be used for memory protection?

Answer: No answer.

18.7 Consider a computing environment where a unique number is associated with each process and each object in the system. Suppose that we allow a process with number n to access an object with number m only if $n > m$. What type of protection structure do we have?

Answer: Hierarchical structure.

18.8 What protection problems may arise if a shared stack is used for parameter passing?

Answer: No answer.

18.9 Consider a computing environment where a process is given the privilege of accessing an object only n times. Suggest a scheme for implementing this policy.

Answer: Add an integer counter with the capability.

18.10 If all the access rights to an object are deleted, the object can no longer be accessed. At this point, the object should also be deleted, and the space it occupies should be returned to the system. Suggest an efficient implementation of this scheme.

Answer: Reference counts.

18.11 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.

18.12 Why is it difficult to protect a system in which users are allowed to do their own I/O?

Answer: No answer.

18.13 Capability lists are usually kept within the address space of the user. How does the system ensure that the user cannot modify the contents of the list?

Answer: No answer.

Chapter 19

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.

■ Answers to Exercises

19.1 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.

19.2 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.

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

Answer: No answer.

19.4 The UNIX program, COPS, scans a given system for possible security holes and alerts the user to possible problems. What are the 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 media or file system to avoid its modification. It could only be provided to bona fide systems managers to prevent it from falling into the wrong hands. Neither of these is a foolproof solution, however.

19.5 Discuss ways by which managers of systems connected to the Internet could have limited or eliminated the damage done by the worm. What are the drawbacks of making such changes to the way in which the system operates?

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).

19.6 Argue for or against the sentence handed down against Robert Morris, Jr., for his creation and execution of the Internet worm.

Answer: No answer.

19.7 Make a list of security concerns for a computer system for a bank. For each item on your list, state whether this concern relates to physical security, human security, 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.

19.8 What are the 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.

Chapter 20

THE UNIX SYSTEM

Although operating-system concepts can be considered in purely theoretical terms, it is often useful to see how they are implemented in practice. This chapter presents an in-depth examination of the 4.3BSD operating system, a version of UNIX, as an example of the various concepts presented in this book. By examining a complete, real system, we can see how the various concepts discussed in this book relate both to one another and to practice.

This UNIX operating system was chosen in part because at one time it was almost small enough to understand and yet is not a "toy" operating system. Most of its internal algorithms were selected for *simplicity*, not for speed or sophistication. UNIX is readily available to departments of computer science, so many students may have access to it.

It might be best to have the students read the papers by Ritchie and Thompson [1974] and Thompson [1978] before reading this chapter.

■ Answers to Exercises

20.1 What are the major differences between 4.3BSD UNIX and SYSVR3? Is one system "better" than the other? Explain your answer.

Answer: 4.3BSD includes several features not found in SYSVR3, including long file names (up to 254 characters), the Berkeley File System (faster file access and more robust), symbolic links (soft pointers to files), processes having multiple access groups, and job control (easy per-job multiprocessing). SYSVR3 has Streams (a multilayered communications facility). Neither is "better" per se, but BSD does have more features.

20.2 How were the design goals of UNIX different from those of other operating systems during the early stages of UNIX development?

Answer: Rather than being a market-oriented operating system, like MULTICS, with definite goals and features, UNIX grew as a tool to allow Thompson and Ritchie to get their research done at Bell Labs. They found a spare PDP-11 system and wrote UNIX to help them with text-processing requirements. It therefore exactly suited their personal needs, not those of a company.

20.3 Why are many different versions of UNIX currently available? In what ways is this diversity an advantage to UNIX? In what ways is it a disadvantage?

Answer: AT&T made the source code of UNIX available to universities and other sites, where experimentation and expansion took place. This allowed many people to have an influence on UNIX and to try out their own ideas. These ideas were circulated, and the best ones were culled for inclusion in the standard varieties of UNIX. The disadvantage this causes is there is no "standard" version of UNIX. Programs written for UNIX may only run on one, or some, versions of UNIX but rarely all.

20.4 What are the advantages and disadvantages of writing an operating system in a high-level language, such as C?

Answer: C makes UNIX highly portable, as evidenced by the many systems it runs on. It is also (arguably) faster to write and debug code in a high-level language, allowing UNIX to be modified more quickly than assembly-language-based operating systems. Of course, it runs less efficiently than if it had been written in assembly language, like most other operating systems. It is generally larger than assembly-language operating systems too.

20.5 In what circumstances is the system-call sequence **fork execve** most appropriate? When is **vfork** preferable?

Answer: Since **vfork** is a fairly dangerous system call, it should only be used when a large process needs to be started. For small child processes, the **fork execve** call sequence is almost as efficient and does not allow its address space to be affected.

20.6 Does 4.3BSD UNIX give scheduling priority to I/O or CPU-bound processes? For what reason does it differentiate between these categories, and why is one given priority over the other? How does it know which of these categories fits a given process?

Answer: I/O-bound processes have priority. Since I/O-bound processes (like text editors) are more closely associated with a user, a better performance for I/O-bound processes give the users quicker response and makes the system seem "faster." UNIX tracks the number of input and output characters for each process, and the devices they are associated with. The more characters UNIX sees coming from tty devices, the more I/O-bound a process is.

20.7 Early UNIX systems used swapping for memory management, whereas 4.3BSD used paging and swapping. Discuss the advantages and disadvantages of the two memory methods.

Answer: When a CPU is slow, compared to its backing store, swapping makes sense. The CPU can issue one transfer command, and the I/O system can move an entire process into or out of main memory. As CPUs get faster, paging makes more sense. The CPU has more time to decide which pages are not being used and to issue transfer requests. Paging generally requires "smarter" hardware, with access bits for each page of memory, or at least invalid page bits. Swapping wastes memory due to external fragmentation. Even on paging systems, swapping is useful when thrashing is occurring due to too many active processes touching too many pages.

20.8 Describe the modifications to a file system that the 4.3BSD kernel makes when a process requests the creation of a new file */tmp/foo* and writes to that file sequentially until the file size reaches 20K.

Answer: Let's assume that the block size is 4K. The kernel receives a *creat* or *open* system call (with the "create" flag set). It locates the directory in which the file is requested to be created and verifies that the process has write permission in that directory, and that no file exists with that same name without write permission. It locates the cylinder group that contains the directory, and it finds a free inode in that cylinder group if there is room; if not, it does a "long seek" to a nearby group that has room.

It allocates the inode by removing it from the free inode list. It then modifies the free inode to show that it is used and updates all the appropriate fields (write date, size = 0, owner and group, protection, etc.). The system then creates a new directory entry in the parent directory's data area that has the name of the new file and a pointer to its newly allocated inode. The inode is then placed in the per-process table of open files, and its file pointer is set to 0. The kernel's file-structure table and the in-core inode list are updated too. The directory entry is then written to disk to assure that directories are always consistent.

The system then receives "write" system calls until 20K of data is received. If the caller is efficient, the writes will occur in 4K chunks (the size of a complete block). If this is the case, the system locates a free block in the cylinder group and changes the free block bit map to show the block is in use. It changes the inode such that the next free direct block is changed to have the value of the disk block. So the first write of 4K would allocate the first direct block, the second write the second block, and so on. These writes are buffered in the block buffer cache until the system deems it necessary to write them to disk.

If writes are done in other than 4K increments, the system must allocate fragments of 1K to handle any writes that do not end at a 4K increment. Each following write would require the system to copy the data in any fragments left by last write into a new block and would start appending the new data there. Obviously this is very efficient (2 reads and a write per write). Fortunately, the disk buffer cache alleviates some of this overhead by not writing data immediately to disk.

20.9 Directory blocks in 4.3BSD are written synchronously when they are changed. Consider what would happen if they were written asynchronously. Describe the state of the file system if a crash occurred after all the files in a directory were deleted but before the directory entry was updated on disk.

Answer: The contents of the file system and the description of that file system (the directory structure) would not correspond. In such a case points to invalid blocks, or blocks of another file, might result. The file system would be in a state of chaos and unusable.

20.10 Describe the process that is needed to recreate the free list after a crash in 4.1BSD.

Answer: No answer.

20.11 What effects on system performance would the following changes to 4.3BSD have? Explain your answers.

a. The merging of the block buffer cache and the process paging space

b. Clustering disk I/O into larger chunks

c. Implementing and using shared memory to pass data between processes rather than using RPC or sockets

d. Using the ISO seven-layer networking model rather than the ARM network model

Answer:

a. Such a merge was done in SunOS 4.1. The result is a more general model of memory use. If lots of file transfers are occurring, more memory is used to hold data blocks. If more processes are executing, more storage is devoted to paging.

b. Another change to SunOS. This change resulted in more efficient use of the disks in the system—larger chunks of data are transferred with fewer seeks.

c. More efficient data transfer between communicating processes.

d. Less efficient network use, as a packet spends more time traversing the network protocol stack before and after being transmitted on the network.

20.12 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: Reliable delivered message would be best, because transfers are sure to occur but open connections are not needed between the systems. Datagrams are the next best, because they are unreliable. Perhaps streams are another choice if open connections are desired. Sun NFS uses datagrams because reliable delivered messages are not implemented. A datagram is about the only choice for testing the availability of other systems, since they may or may not be able to receive a packet (disallowing reliable delivered messages).

Chapter 21

THE LINUX SYSTEM

Chapter 20 discussed the internals of the 4.3BSD operating system in detail. BSD is just one of the UNIX-like systems. Linux is another 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 2.0 kernel, released in June 1996.

Answers to Exercises

21.1 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:

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.

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.

21.2 Dynamically loadable kernel modules give flexibility when drivers are added to a system, but do they have disadvantages too? Under what circumstances would a kernel be compiled into a single binary file, and when would it be better to keep it split into modules? Explain your answer.

Answer: There are two principal drawbacks with the use of modules. The first is size: module management consumes unpageable kernel memory, and a basic kernel with a number of modules loaded will consume more memory than an equivalent kernel with the drivers compiled into the kernel image itself. This can be a very significant issue on machines with limited physical memory.

The second drawback is that modules can increase the complexity of the kernel bootstrap process. It is hard to load up a set of modules from disk if the driver needed to access that disk as itself a module that needs to be loaded. As a result, managing the kernel bootstrap with modules can require extra work on the part of the administrator: the modules required to bootstrap need to be placed into a ramdisk image that is loaded alongside the initial kernel image when the system is initialized.

In certain cases it is better to use a modular kernel, and in other cases it is better to use a kernel with its device drivers prelinked. Where minimizing the size of the kernel is important, the choice will depend on how often the various device drivers are used. If they are in constant use, then modules are unsuitable. This is especially true where drivers are needed for the boot process itself. On the other hand, if some drivers are not always needed, then the module mechanism allows those drivers to be loaded and unloaded on demand, potentially offering a net saving in physical memory.

Where a kernel is to be built which must be usable on a large variety of very different machines, then building it with modules is clearly preferable to using a single kernel with dozens of unnecessary drivers consuming memory. This is particularly the case for commercially distributed kernels, where supporting the widest variety of hardware in the simplest manner possible is a priority.

However, if a kernel is being built for a single machine whose configuration is known in advance, then compiling and using modules may simply be an unnecessary complexity. In cases like this, the use of modules may well be a matter of taste.

21.3 Multithreading is a commonly used programming technique. Describe three different ways that threads could be implemented. Explain how these ways compare to the Linux **clone** mechanism. When might each alternative mechanism be better or worse than using clones?

Answer: Thread implementations can be broadly classified into two groups: kernel-based threads and user-mode threads. User-mode thread packages rely on some kernel support-they may require timer interrupt facilities, for example-but the scheduling between threads is not performed by the kernel but by some library of user-mode code. Multiple threads in such an implementation appear to the operating system as a single execution context. When the multithreaded process is running, it decides for itself which of its threads to execute, using non-local jumps to switch between threads according to its own preemptive or non-preemptive scheduling rules.

Alternatively, the operating system kernel may provide support for threads itself. In this case, the threads may be implemented as separate processes that happen to share a complete or partial common address space, or they may be implemented as separate execution contexts within a single process. Whichever way the threads are organized, they appear as fully independent execution contexts to the application.

Hybrid implementations are also possible, where a large number of threads are made available to the application using a smaller number of kernel threads. Runnable user threads are run by the first available kernel thread.

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.

A lesser advantage is the ability to assign different security attributes to each thread.

User-mode implementations do not have these advantages. Because such implementations run entirely within a single kernel execution context, only one thread can ever be running at once, even if multiple CPUs are available. For the same reason, if one thread enters a system call, no other threads can run until that system call completes. As a result, one thread doing a blocking disk read will hold up every thread in the application. However, user-mode implementations do have their own advantages. The most obvious is performance: invoking the kernel's own scheduler to switch between threads involves entering a new protection domain as the CPU switches to kernel mode, whereas switching between threads in user-mode can be achieved simply by saving and restoring the main CPU registers. User-mode threads may also consume less system memory: most UNIX systems will reserve at least a full page for a kernel stack for each kernel thread, and this stack may not be pageable.

The hybrid approach, implementing multiple user threads over a smaller number of kernel threads, allows a balance between these tradeoffs to be achieved. The kernel threads will allow multiple threads to be in blocking kernel calls at once and will permit running on multiple CPUs, and user-mode thread switching can occur within each kernel thread to perform lightweight threading without the overheads of having too many kernel threads. The downside of this approach is complexity: giving control over the tradeoff complicates the thread library's user interface.

21.4 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.

21.5 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.

21.6 The Linux kernel does not allow paging out of kernel memory. What effect does this restriction have on the kernel's design? What are two advantages and two disadvantages of this design decision?

Answer: The primary impact of disallowing paging of kernel memory in Linux is that the non-preemptability of the kernel is preserved. Any process taking a page fault, whether in kernel or in user mode, risks being rescheduled while the required data is paged in from disk. Because the kernel can rely on not being rescheduled during access to its primary data structures, locking requirements to protect the integrity of those data structures are very greatly simplified. Although design simplicity is a benefit in itself, it also provides an important performance advantage on uni-processor machines due to the fact that it is not necessary to do additional locking on most internal data structures.

There are a number of disadvantages to the lack of pageable kernel memory, however. First of all, it imposes constraints on the amount of memory that the kernel can use. It is unreasonable to keep very large data structures in non-pageable memory, since that represents physical memory that absolutely cannot be used for anything else. This has two impacts: first of all, the kernel must prune back many of its internal data structures manually, instead of being able to rely on a single virtual memory mechanism to keep physical memory usage under control. Second, it makes it infeasible to implement certain features that require large amounts of virtual memory in the kernel, such as the /tmp-filesystem (a fast virtual memory based file-system found on some UNIX systems).

Note that the complexity of managing page faults while running kernel code is not an issue here. The Linux kernel code is already able to deal with page faults: it needs to be able to deal with system calls whose arguments reference user memory which may be paged out to disk.

21.7 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:

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.

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 at most once it is only loaded 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.

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 which 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.

21.8 What are three advantages of dynamic (shared) linkage of libraries compared to static linkage? What are two cases where static linkage is preferable.

Answer: The primary advantages of shared libraries are that they reduce the memory and disk space used by a system, and they enhance maintainability.

When shared libraries are being used by all running programs, there is only one instance of each system library routine on disk, and at most one instance in physical memory. When the library in question is one used by many applications and programs, then the disk and memory savings can be quite substantial. In addition, the startup time for running new programs can be reduced, since many of the common functions needed by that program are likely to be already loaded into physical memory.

Maintainability is also a major advantage of dynamic linkage over static. If all running programs use a shared library to access their system library routines, then upgrading those routines, either to add new functionality or to fix bugs, can be done simply by replacing that shared library. There is no need to recompile or relink any applications;

any programs loaded after the upgrade is complete will automatically pick up the new versions of the libraries.

There are other advantages too. A program that uses shared libraries can often be adapted for specific purposes simply by replacing one or more of its libraries, or even (if the system allows it, and most UNIXs including Linux do) adding a new one at run time. For example, a debugging library can be substituted for a normal one to trace a problem in an application. Shared libraries also allow program binaries to be linked against commercial, proprietary library code without actually including any of that code in the program's final executable file. This is important because on most UNIX systems, many of the standard shared libraries are proprietary, and licensing issues may prevent including that code in executable files to be distributed to third parties.

In some places, however, static linkage is appropriate. One example is in rescue environments for system administrators. If a system administrator makes a mistake while installing any new libraries, or if hardware develops problems, it is quite possible for the existing shared libraries to become corrupt. As a result, often a basic set of rescue utilities are linked statically, so that there is an opportunity to correct the fault without having to rely on the shared libraries functioning correctly.

There are also performance advantages that sometimes make static linkage preferable in special cases. For a start, dynamic linkage does increase the startup time for a program, as the linking must now be done at run-time rather than at compile-time. Dynamic linkage can also sometimes increase the maximum working set size of a program (the total number of physical pages of memory required to run the program). In a shared library, the user has no control over where in the library binary file the various functions reside. Since most functions do not precisely fill a full page or pages of the library, loading a function will usually result in loading in parts of the surrounding functions, too. With static linkage, absolutely no functions that are not referenced (directly or indirectly) by the application need to be loaded into memory.

Other issues surrounding static linkage include ease of distribution: it is easier to distribute an executable file with static linkage than with dynamic linkage if the distributor is not certain whether the recipient will have the correct libraries installed in advance. There may also be commercial restrictions against redistributing some binaries as shared libraries. For example, the license for the UNIX "Motif" graphical environment allows binaries using Motif to be distributed freely as long as they are statically linked, but the shared libraries may not be used without a license.

21.9 Compare the use of networking sockets with the use of shared memory as a mechanism for communicating data between processes on a single computer. What are the advantages of each method? When might each be preferred?

Answer: Using network sockets rather than shared memory for local communication has a number of advantages. The main advantage is that the socket programming interface features a rich set of synchronization features. A process can easily determine when new data has arrived on a socket connection, how much data is present, and who sent it. Processes can block until new data arrives on a socket, or they can request that a signal be delivered when data arrives. A socket also manages separate connections. A process with a socket open for receive can accept multiple connections to that socket and will be told when new processes try to connect or when old processes drop their connections.

Shared memory offers none of these features. There is no way for a process to determine whether another process has delivered or changed data in shared memory other than by going to look at the contents of that memory. It is impossible for a process to block and request a wakeup when shared memory is delivered, and there is no standard mechanism for other processes to establish a shared memory link to an existing process.

However, shared memory has the advantage that it is very much faster than socket communications in many cases. When data is sent over a socket, it is typically copied from memory to memory multiple times. Shared memory updates require no data copies: if one process updates a data structure in shared memory, that update is immediately visible to all other processes sharing that memory. Sending or receiving data over a socket requires that a kernel system service call be made to initiate the transfer, but shared memory communication can be performed entirely in user mode with no transfer of control required.

Socket communication is typically preferred when connection management is important or when there is a requirement to synchronize the sender and receiver. For example, server processes will usually establish a listening socket to which clients can connect when they want to use that service. Once the socket is established, individual requests are also sent using the socket, so that the server can easily determine when a new request arrives and who it arrived from.

In some cases, however, shared memory is preferred. Shared memory is often a better solution when either large amounts of data are to be transferred or when two processes need random access to a large common data set. In this case, however, the communicating process may still need an extra mechanism in addition to shared memory to achieve synchronization between themselves. The X Window System, a graphical display environment for UNIX, is a good example of this: most graphic requests are sent over sockets, but shared memory is offered as an additional transport in special cases where large bitmaps are to be displayed on the screen. In this case, a request to display the bitmap will still be sent over the socket, but the bulk data of the bitmap itself will be sent via shared memory.

21.10 UNIX systems used to use disk-layout optimizations based on the rotation position of disk data, but modern implementations, including Linux, simply optimize for sequential data access. Why do they do so? Of what hardware characteristics does sequential access take advantage? Why is rotational optimization no longer so useful?

Answer: The performance characteristics of disk hardware has changed substantially in recent years. In particular, many enhancements have been introduced to increase the maximum bandwidth that can be achieved on a disk. In a modern system, there can be a long pipeline between the operating system and the disk's read-write head. A disk I/O request has to pass through the computer's local disk controller, over bus logic to the disk drive itself, and then internally to the disk where there is likely to be a complex controller that can cache data accesses and potentially optimize the order of I/O requests.

Because of this complexity, the time taken for one I/O request to be acknowledged and for the next request to be generated and received by the disk can far exceed the amount of time between one disk sector passing under the read-write head and the next sector header arriving. In order to be able to efficiently read multiple sectors at once, disks will employ a readahead cache. While one sector is being passed back to the host computer, the disk will be busy reading the next sectors in anticipation of a request to read them. If read requests start arriving in an order that breaks this readahead pipeline, performance will drop. As a result, performance benefits substantially if the operating system tries to keep I/O requests in strict sequential order.

A second feature of modern disks is that their geometry can be very complex. The number of sectors per cylinder can vary according to the position of the cylinder: more data can be squeezed into the longer tracks nearer the edge of the disk than at the center of the disk. For an operating system to optimize the rotational position of data on such disks, it would have to have complete understanding of this geometry, as well as the timing characteristics of the disk and its controller. In general, only the disk's internal logic can determine the optimal scheduling of I/Os, and the disk's geometry is likely to defeat any attempt by the operating system to perform rotational optimizations.

21.11 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.

Chapter 22

WINDOWS / NT

The Windows NT operating system is designed to take advantage of the many advances in processor technology. Although primarily run on the Intel architecture, NT was designed to be portable in order to take advantage of whatever promising technologies happened to come along. Key goals for the system included portability, security, POSIX compliance, multiprocessor support, extensibility, international support, and compatibility with MS-DOS and MS-Windows applications. Windows NT is similar to Mach in that it is a micro-kernel based operating system that results in a stable base operating system and allows enhancements to be made to one part of the operating system without changing any of the other parts.

■ Answers to Exercises

22.1 Discuss why moving the graphics code in NT from user mode to kernel mode would decrease the reliability of the system. How does this violate the original design goals for NT?

Answer: The code was moved to eliminate the overhead of interprocess communication. The advantage of the previous method of having the code in the Win32 subsystem is that the kernel / executive as well as other subsystems are protected from an error in the Win32 subsystem. The new method, while offering a performance increase to meet marketplace concerns, has the drawback that bad graphics code can bring down the entire system. Indeed, examples of this were seen posted on the Internet. The design goal that was violated was that of independent subsystems that would not be able to affect other subsystems or the kernel. This was brought about by complaints from users of the older 16-bit windows versions who felt that applications ran slower on NT.

22.2 The NT VM manager uses a two-stage process to allocate memory. Why is this approach beneficial?

Answer: A process in NT is limited to 2 GB address space for data. The two-stage process allows the access of much larger datasets, by reserving space in the processes address space first and then committing the storage to a memory mapped file. An application could thus window through a large database (by changing the committed section) without exceeding process quotas or utilizing a huge amount of physical memory.

22.3 Discuss the advantages and disadvantages of the particular page-table structure in NT.

Answer: Each process has its own page directory that requires about 4 MB of storage. Since it is a three level design, this means that there could be up to three page faults just accessing a virtual address. Shared memory adds one more level. The page faults can occur because NT does not commit the required memory (the 4 MB) until necessary. Since each process has its own page directory, there is no way for processes to share virtual addresses. The prototype page-table entry adds a level of indirection but eliminates the update of multiple page-table entries for shared pages.

22.4 What is the maximum number of page faults that could occur in the access of a virtual address and of a shared virtual address? What hardware mechanism is provided by most processors to decrease these numbers?

Answer: 4 for shared addresses. 3 for others. Translation Lookaside Buffers.

22.5 What is the purpose of a prototype page-table entry in NT?

Answer: The prototype page-table entry is used to point to shared pages instead of having multiple page-table entries point to the same page. It adds another layer of indirection but saves having to update N page-table entries.

22.6 What steps must the cache manager take to copy data into and out of the cache?

Answer: Please see Section 22.4.6 for details.

22.7 What are the problems involved in running 16-bit Windows applications in a VDM? What are the solutions chosen by NT? What are the drawbacks of these solutions?

Answer: No answer.

22.8 What changes would be needed for NT to run a process that uses a 64-bit address space?

Answer: Primarily, the VM Manager would have to be extensively modified. This might entail changing the page size, adding another level to the page-table structure, and so on. It may be impractical to support the full 64-bit address range. Indeed, the “64-bit” version of NT, NT Server/E 5.0, will support a maximum of 32 megabytes of RAM. For another approach, see the August 1997 Oracle White Paper entitled ”Oracle Very Large Memory (VLM) for Digital Alpha NT.”

22.9 NT has a centralized cache manager. What are the advantages and disadvantages of this cache manager?

Answer: One of the major advantages is that each file system doesn’t have to provide its own caching. Also, the cache manager is tightly coupled to the VM manager. The drawback is that some devices want to do DMA transfers. Also, different caching schemes might be able to save the data copying that is present with the NT scheme.

22.10 NT uses a packet-driven I/O system. Discuss the pros and cons of the packet-driven approach to I/O.

Answer: The standard form of the packet makes it easier to write drivers since they can follow a standard interface and processing hierarchy. A major disadvantage is that all the packet copying leads to inefficiencies, although that many TCP/IP stacks apparently have the same problem.

22.11 Consider a main-memory database of 1 TB. What mechanisms in NT could be used to access it?

Answer: See Question 22.2.