computer system. Security measures are responsible for defending a computer system from external or internal attacks.

Distributed systems allow users to share resources on geographically dispersed hosts connected via a computer network. Services may be provided through either the client–server model or the peer-to-peer model. In a clustered system, multiple machines can perform computations on data residing on shared storage, and computing can continue even when some subset of cluster members fails.

LANs and WANs are the two basic types of networks. LANs enable processors distributed over a small geographical area to communicate, whereas WANs allow processors distributed over a larger area to communicate. LANs typically are faster than WANs.

There are several computer systems that serve specific purposes. These include real-time operating systems designed for embedded environments such as consumer devices, automobiles, and robotics. Real-time operating systems have well-defined, fixed-time constraints. Processing must be done within the defined constraints, or the system will fail. Multimedia systems involve the delivery of multimedia data and often have special requirements of displaying or playing audio, video, or synchronized audio and video streams.

Recently, the influence of the Internet and the World Wide Web has encouraged the development of operating systems that include Web browsers and networking and communication software as integral features.

The free software movement has created thousands of open-source projects, including operating systems. Because of these projects, students are able to use source code as a learning tool. They can modify programs and test them, help find and fix bugs, and otherwise explore mature, full-featured operating systems, compilers, tools, user interfaces, and other types of programs.

GNU/Linux, BSD UNIX, and Solaris are all open-source operating systems. The advantages of free software and open sourcing are likely to increase the number and quality of open-source projects, leading to an increase in the number of individuals and companies that use these projects.

Practice Exercises

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

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

1.3 List the four steps that are necessary to run a program on a completely dedicated machine—a computer that is running only that program.

1.4 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?

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

1.6 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 answers.

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

1.8 Which of the following instructions should be privileged?

a. Set value of timer.
b. Read the clock.
c. Clear memory.
d. Issue a trap instruction.
e. Turn off interrupts.
f. Modify entries in device-status table.
g. Switch from user to kernel mode.
h. Access I/O device.

1.9 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.

1.10 Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?

1.11 Timers could be used to compute the current time. Provide a short description of how this could be accomplished.

1.12 Is the Internet a LAN or a WAN?

Exercises

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

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

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

a. Mainframe or minicomputer systems
b. Workstations connected to servers
c. Handheld computers
1.15 Under what circumstances would a user be better off using a timesharing system rather than a PC or a single-user workstation?

1.16 Identify which of the functionalities listed below need to be supported by the operating system for (a) handheld devices and (b) real-time systems.
   a. Batch programming
   b. Virtual memory
   c. Time sharing

1.17 Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?

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

1.19 Distinguish between the client-server and peer-to-peer models of distributed systems.

1.20 Consider a computing cluster consisting of two nodes running a database. Describe two ways in which the cluster software can manage access to the data on the disk. Discuss the benefits and disadvantages of each.

1.21 How are network computers different from traditional personal computers? Describe some usage scenarios in which it is advantageous to use network computers.

1.22 What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

1.23 Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.
   a. How does the CPU interface with the device to coordinate the transfer?
   b. How does the CPU know when the memory operations are complete?
   c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of the user programs? If so, describe what forms of interference are caused.

1.24 Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems? Give arguments both that it is and that it is not possible.

1.25 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?

1.26 Consider an SMP system similar to what is shown in Figure 1.6. Illustrate with an example how data residing in memory could in fact have two different values in each of the local caches.

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

   a. Single-processor systems
   b. Multiprocessor systems
   c. Distributed systems

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

1.29 What network configuration would best suit the following environments?

   a. A dormitory floor
   b. A university campus
   c. A state
   d. A nation

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

   a. Batch
   b. Interactive
   c. Time sharing
   d. Real time
   e. Network
   f. Parallel
   g. Distributed
   h. Clustered
   i. Handheld

1.31 What are the tradeoffs inherent in handheld computers?

1.32 Identify several advantages and several disadvantages of open-source operating systems. Include the types of people who would find each aspect to be an advantage or a disadvantage.
when in an interactive or time-shared mode. System programs are provided to satisfy many common user requests.

The types of requests vary according to level. The system-call level must provide the basic functions, such as process control and file and device manipulation. Higher-level requests, satisfied by the command interpreter or system programs, are translated into a sequence of system calls. System services can be classified into several categories: program control, status requests, and I/O requests. Program errors can be considered implicit requests for service.

Once the system services are defined, the structure of the operating system can be developed. Various tables are needed to record the information that defines the state of the computer system and the status of the system’s jobs.

The design of a new operating system is a major task. It is important that the goals of the system be well defined before the design begins. The type of system desired is the foundation for choices among various algorithms and strategies that will be needed.

Since an operating system is large, modularity is important. Designing a system as a sequence of layers or using a microkernel is considered a good technique. The virtual-machine concept takes the layered approach and treats both the kernel of the operating system and the hardware as though they were hardware. Even other operating systems may be loaded on top of this virtual machine.

Throughout the entire operating-system design cycle, we must be careful to separate policy decisions from implementation details (mechanisms). This separation allows maximum flexibility if policy decisions are to be changed later.

Operating systems are now almost always written in a systems-implementation language or in a higher-level language. This feature improves their implementation, maintenance, and portability. To create an operating system for a particular machine configuration, we must perform system generation.

Debugging process and kernel failures can be accomplished through the use of debuggers and other tools that analyze core dumps. Tools such as DTrace analyze production systems to find bottlenecks and understand other system behavior.

For a computer system to begin running, the CPU must initialize and start executing the bootstrap program in firmware. The bootstrap can execute the operating system directly if the operating system is also in the firmware, or it can complete a sequence in which it loads progressively smarter programs from firmware and disk until the operating system itself is loaded into memory and executed.

Practice Exercises

1. What is the purpose of system calls?
2. What are the five major activities of an operating system with regard to process management?
3. What are the three major activities of an operating system with regard to memory management?
2.4 What are the three major activities of an operating system with regard to secondary-storage management?

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

2.6 What system calls have to be executed by a command interpreter or shell in order to start a new process?

2.7 What is the purpose of system programs?

2.8 What is the main advantage of the layered approach to system design? What are the disadvantages of using the layered approach?

2.9 List five services provided by an operating system, and explain how each creates convenience for users. In which cases would it be impossible for user-level programs to provide these services? Explain your answer.

2.10 Why do some systems store the operating system in firmware, while others store it on disk?

2.11 How could a system be designed to allow a choice of operating systems from which to boot? What would the bootstrap program need to do?

### Exercises

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

2.13 Describe three general methods for passing parameters to the operating system.

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

2.15 What are the five major activities of an operating system with regard to file management?

2.16 What are the advantages and disadvantages of using the same system-call interface for manipulating both files and devices?

2.17 Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

2.18 What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

2.19 Why is the separation of mechanism and policy desirable?

2.20 It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.
2.21 What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

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

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

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

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

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

**Programming Problems**

2.27 In Section 2.3, we described a program that copies the contents of one file to a destination file. This program works by first prompting the user for the name of the source and destination files. Write this program using either the Win32 or POSIX API. Be sure to include all necessary error checking, including ensuring that the source file exists.

Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. Linux systems provide the ptrace utility, and Solaris systems use the truss or dtrace command. On Mac OS X, the ktrace facility provides similar functionality. As Windows systems do not provide such features, you will have to trace through the Win32 version of this program using a debugger.

**Programming Projects**

2.28 Adding a system call to the Linux Kernel.

In this project, you will study the system-call interface provided by the Linux operating system and learn how user programs communicate with the operating system kernel via this interface. Your task is to incorporate a new system call into the kernel, thereby expanding the functionality of the operating system.
Practice Exercises

The processes executing in the operating system may be either independent processes or cooperating processes. Cooperating processes require an interprocess communication mechanism to communicate with each other. Principally, communication is achieved through two schemes: shared memory and message passing. The shared-memory method requires communicating processes to share some variables. The processes are expected to exchange information through the use of these shared variables. In a shared-memory system, the responsibility for providing communication rests with the application programmers; the operating system needs to provide only the shared memory. The message-passing method allows the processes to exchange messages. The responsibility for providing communication may rest with the operating system itself. These two schemes are not mutually exclusive and can be used simultaneously within a single operating system.

Communication in client-server systems may use (1) sockets, (2) remote procedure calls (RPCs), or (3) pipes. A socket is defined as an endpoint for communication. A connection between a pair of applications consists of a pair of sockets, one at each end of the communication channel. RPCs are another form of distributed communication. An RPC occurs when a process (or thread) calls a procedure on a remote application. Ordinary pipes allow communication between parent and child processes, while named pipes permit unrelated processes to communicate with one another.

Practice Exercises

3.1 Palm OS provides no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

3.2 The Sun UltraSPARC processor has multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

3.3 When a process creates a new process using the for() operation, which of the following states is shared between the parent process and the child process?
   a. Stack
   b. Heap
   c. Shared memory segments

3.4 With respect to the RPC mechanism, consider the "exactly once" semantic. Does the algorithm for implementing this semantic execute correctly even if the ACK message back to the client is lost due to a network problem? Describe the sequence of messages and discuss whether "exactly once" is still preserved.

3.5 Assume that a distributed system is susceptible to server failure. What mechanisms would be required to guarantee the "exactly once" semantic for execution of RPCs?
Exercises

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

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

3.8 Construct a process tree similar to Figure 3.9. To obtain process information for the UNIX or Linux system, use the command `ps -ael`. Use the command `man ps` to get more information about the `ps` command. On Windows systems, you will have to use the task manager.

3.9 Including the initial parent process, how many processes are created by the program shown in Figure 3.28?

3.10 Using the program in Figure 3.29, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

3.11 Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

3.12 Consider the RPC mechanism. Describe the undesirable consequences that could arise from not enforcing either the “at most once” or “exactly once” semantic. Describe possible uses for a mechanism that has neither of these guarantees.

3.13 Using the program shown in Figure 3.30, explain what the output will be at Line A.

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

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

Figure 3.28 How many processes are created?
```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d", pid); /* A */
        printf("child: pid1 = %d", pid1); /* B */
    }
    else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d", pid); /* C */
        printf("parent: pid1 = %d", pid1); /* D */
        wait(NULL);
    }

    return 0;
}
```

**Figure 3.29** What are the pid values?

**3.14** What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

a. Synchronous and asynchronous communication

b. Automatic and explicit buffering

c. Send by copy and send by reference

d. Fixed-sized and variable-sized messages

**Programming Problems**

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

\[ fib_0 = 0 \]
\[ fib_1 = 1 \]
\[ fib_n = fib_{n-1} + fib_{n-2} \]
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main()
{
    pid_t pid;

    pid = fork();

    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d",value); /* LINE A */
        return 0;
    }
}

Figure 3.30 What output will be at Line A?

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

3.16 Repeat the preceding exercise, this time using the CreateProcess() function in the Win32 API. In this instance, you will need to specify a separate program to be invoked from CreateProcess(). It is this separate program that will run as a child process outputting the Fibonacci sequence. Perform necessary error checking to ensure that a non-negative number is passed on the command line.

3.17 Modify the date server shown in Figure 3.19 so that it delivers random jokes rather than the current date. Allow the jokes to contain multiple lines. The date client shown in Figure 3.20 can be used to read the multi-line jokes returned by the joke server.

3.18 An echo server echoes back whatever it receives from a client. For example, if a client sends the server the string Hello there! the server will respond with the exact data it received from the client—that is, Hello there!
task points to the data structures of the parent task, depending on the set of flags passed to clone().

Several distributions of the Linux kernel now include the NPTL thread library. NPTL (which stands for Native POSIX Thread Library) provides a POSIX-compliant thread model for Linux systems along with several other features, such as better support for SMP systems, as well as taking advantage of NUMA support. In addition, the start-up cost for creating a thread is lower with NPTL than with traditional Linux threads. Finally, with NPTL, the system has the potential to support hundreds of thousands of threads. Such support becomes more important with the growth of multicore and other SMP systems.

4.6 Summary

A thread is a flow of control within a process. A multithreaded process contains several different flows of control within the same address space. The benefits of multithreading include increased responsiveness to the user, resource sharing within the process, economy, and scalability issues such as more efficient use of multiple cores.

User-level threads are threads that are visible to the programmer and are unknown to the kernel. The operating-system kernel supports and manages kernel-level threads. In general, user-level threads are faster to create and manage than are kernel threads, as no intervention from the kernel is required. Three different types of models relate user and kernel threads: The many-to-one model maps many user threads to a single kernel thread. The one-to-one model maps each user thread to a corresponding kernel thread. The many-to-many model multiplexes many user threads to a smaller or equal number of kernel threads.

Most modern operating systems provide kernel support for threads; among these are Windows 98, NT, 2000, and XP, as well as Solaris and Linux.

Thread libraries provide the application programmer with an API for creating and managing threads. Three primary thread libraries are in common use: POSIX Pthreads, Win32 threads for Windows systems, and Java threads.

Multithreaded programs introduce many challenges for the programmer, including the semantics of the fork() and exec() system calls. Other issues include thread cancellation, signal handling, and thread-specific data.

Practice Exercises

4.1 Provide two programming examples in which multithreading provides better performance than a single-threaded solution.

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

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

4.4 What resources are used when a thread is created? How do they differ from those used when a process is created?
4.5 Assume that an operating system maps user-level threads to the kernel using the many-to-many model and that the mapping is done through LWPs. Furthermore, the system allows developers to create real-time threads for use in real-time systems. Is it necessary to bind a real-time thread to an LWP? Explain.

4.6 A Pthread program that performs the summation function was provided in Section 4.3.1. Rewrite this program in Java.

Exercises

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

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

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

4.10 Which of the following components of program state are shared across threads in a multithreaded process?
   a. Register values
   b. Heap memory
   c. Global variables
   d. Stack memory

4.11 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? Explain.

4.12 As described in Section 4.5.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, many operating systems—such as Windows XP and Solaris—treat processes and threads differently. Typically, such systems use a notation wherein the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

4.13 The program shown in Figure 4.14 uses the Pthreads API. What would be the output from the program at LINE c and LINE p?

4.14 Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level
```c
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[]) {
    int pid;
    pthread_t tid;
    pthread_attr_t attr;

    pid = fork();
    if (pid == 0) { /* child process */
        pthread_attr_init(&attr);
        pthread_create(&tid,&attr,runner,NULL);
        pthread_join(tid,NULL);
        printf("CHILD: value = %d",value); /* LINE C */
    } else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d",value); /* LINE P */
    }
}

void *runner(void *param) {
    value = 5;
    pthread_exit(0);
}
```

Figure 4.14  C program for Exercise 4.13.

threads in the program be more than the number of processors in the system. Discuss the performance implications of the following scenarios:

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

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

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

4.15 Write a multithreaded Java, Pthreads, or Win32 program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.
4.16 Modify the socket-based date server (Figure 3.19) in Chapter 3 so that the server services each client request in a separate thread.

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

\[
\begin{align*}
    f_{ib_0} &= 0 \\
    f_{ib_1} &= 1 \\
    f_{ib_n} &= f_{ib_{n-1}} + f_{ib_{n-2}}
\end{align*}
\]

Write a multithreaded program that generates the Fibonacci sequence using either the Java, Pthreads, or Win32 thread library. This program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish, using the techniques described in Section 4.3.

4.18 Exercise 3.18 in Chapter 3 involves designing an echo server using the Java threading API. However, this server is single-threaded, meaning that the server cannot respond to concurrent echo clients until the current client exits. Modify the solution to Exercise 3.18 so that the echo server services each client in a separate request.

Projects

The set of projects below deal with two distinct topics—naming service and matrix multiplication.

Project 1: Naming Service Project

A naming service such as DNS (for domain name system) can be used to resolve IP names to IP addresses. For example, when someone accesses the host www.westminstercollege.edu, a naming service is used to determine the IP address that is mapped to the IP name www.westminstercollege.edu. This assignment consists of writing a multithreaded naming service in Java using sockets (see Section 3.6.1).

The java.net API provides the following mechanism for resolving IP names:

```java
InetAddress hostAddress = InetAddress.getByName("www.westminstercollege.edu");
String IPAddress = hostAddress.getHostAddress();
```

where getByName() throws an UnknownHostException if it is unable to resolve the host name.