T2.1 ITSD BSS Exercises

Table of Contents

Chapter 1 Introduction

Exercise 1.1 Multiprogramming

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

Answer 1.1

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

Exercise 1.2 Symmetric Multiprocessing

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

Answer 1.2

Symmetric multiprocessing treats all processors as equals, and I/ O can be processed on any CPU. A symmetric 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 m aster 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.

Exercise 1.3 Clustered Systems

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

Answer1.3

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

Exercise 1.4 Interrupts

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

Answer 1.4

An interrupt is a hard ware-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.

Exercise 1.5 Caches

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 1.5

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

Exercise 1.6 Memory Protection

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

Answer 1.6

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

Exercise 1.7 Protection and Security

Explain the difference between protection and security.

Answer 1.7 (p 29)

- Protection is any mechanism for controlling the access of processes or users to the resources defined by a computer system. This mechanism must provide a means to specify the controls to be imposed and a means to enforce the controls. Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.
- Security to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial-of-service attacks (which use all of a system’s resources and so keep legitimate users out of the system), identity theft, and theft of service (unauthorized use of a system). Prevention of some of these attacks is considered an operating-system function on some systems, while other systems leave the prevention to policy or additional software.

Chapter 2 System Structures

Exercise 2.1 Operating System

Name three points to view an operating system

Answer 2.1 (p 49)

- The services that the system provides
- The interface that it makes available to users and programmers
- Its components and their interconnections

Exercise 2.2 OS Services

Name six operating-system services that provide functions that are helpful to the user.

Answer 2.2 (p 50)

1. User interface, for example a command-line interface (CLI), a batch interface or a graphical user interface (GUI)
2. Program execution; to load and run programs and to end its execution (normally or abnormally)
3. I/O operations; for accessing files and devices
4. File-system manipulation; also permissions management
5. Communications; to exchange information with another process
6. Error detection; The operating system needs to be constantly aware of possible errors and Debugging facilities

Exercise 2.3 Command Interpreter

What is the main function of the command interpreter?

Answer 2.3 (p 52)

The main function of the command interpreter, or shell, is to get and execute the next user-specified command.

Exercise 2.4 OS Functions

Name three operating system functions for ensuring the efficient operation of the system itself

Answer 2.4 (p 51)

1. Resource allocation; for resources such as CPU cycles, main memory, and file storage, I/O devices, printers, modems, USB storage drives, and other peripheral devices.
2. Accounting; keep track of user activities for billing or statistics
3. Protection and security; to ensuring that all access to system resource is controlled

Exercise 2.5 System Calls

How do application programmers typically access system calls?

Answer 2.5 (p 54)

Typically, application developers design programs according to an application programming interface (API).The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.

Exercise 2.6 Function Calls

What do functions calls provide?

Answer 2.6 (p 55)

System calls provide an interface to the services made available by an operating system.
Each operating system has its own name for each system call. Behind the scenes, the functions that make up an API typically invoke the actual system calls on behalf of the application programmer.

Exercise 2.7 API vs System Call

Why would an application programmer prefer programming according to an API rather than invoking actual system calls?

Answer 2.7 (p 56)

There are several reasons for doing so.

- Portability; the program can be compiled and run on any system that supports the same API (although in reality, architectural differences often make this more difficult than it may appear).
- The actual system calls can often be more detailed and difficult to work with than the API available to an application programmer.

Exercise 2.8 Passing Parameters

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

Answer 2.8 (p 58)

a) Pass parameters in registers,
b) If there are more parameters than registers then the parameters are stored in a block, or table in memory. The address of the block is passed in a register
c) Parameters can be placed, or pushed, onto the stack by the program, and popped off the stack by the operating system

Exercise 2.9 Native Methods

Why does Java provide the ability to call from a Java program native methods that are written in, say, C or C ++? Provide an example of a situation in which a native method is useful.

Answer 2.9 (p 58)

Java programs are intended to be platform I/ O independent. Therefore, the language does not provide access to most specific system re sources 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.

Exercise 2.10 System Calls

System calls can be grouped roughly into six major categories

Answer 2.10 (p 59)

- Process control: end, abort; load, execute; create process, terminate process; get process attributes, set process attributes; wait for time; wait event, signal event; allocate and free memory
- File management: create file, delete file; open, close; read, write, reposition; get file attributes, set file attributes
- Device management: request device, release device; read, write, reposition; get device attributes, set device attributes; logically attach or detach devices
- Information maintenance: get time or date, set time or date; get system data, set system data; get process, file, or device attributes; set process, file, or device attributes
- Communications: create, delete communication connection; send, receive messages; transfer status information; attach or detach remote devices
- Protection: provides a mechanism for controlling access to a computer system's resources (permissions)

Exercise 2.11 Separation of Policy from Mechanism

One important principle in the design of operating systems is the separation of policy from mechanism. Describe what it means and give an example.

Answer 2.11 (p 68)

- Mechanisms determine how to do something; policies determine what will be done. The separation of policy and mechanism is important for flexibility. Policies are likely to change across places or over time. A general mechanism insensitive to changes in policy would be more desirable.
  If the mechanism is properly separated from policy, it can be used either to support a policy decision that I/O -intensive programs should have priority over CPU-intensive ones or to support the opposite policy.
  For example, the timer construct (see Section 1.5.2) is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision.
- Policy decisions are important for all resource allocation. Whenever it is necessary to decide whether or not to allocate a resource, a policy decision must be made. Whenever the question is how rather than what,it is a mechanism that must be determined.

Exercise 2.12 Separation of mechanism and policy

Why is the separation of mechanism and policy desirable?

Answer 2.12

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.

Exercise 2.13 Emulators

Describe the use of emulators.

Answer 2.13 (p 70)

Emulators are programs that duplicate the functionality of one system in another system.

Exercise 2.14 Monolithic and Microkernel

Describe the difference between a monolithic and microkernel structure.

Answer 2.14 (p 70)

- A microkernel (also known as μ-kernel) is the near-minimum amount of software that can provide the mechanisms needed to implement an operating system (OS). These mechanisms include low-level address space management, thread management, and inter-process communication (IPC).
  Traditional operating system functions, such as device drivers, protocol stacks and file systems, are removed from the microkernel to run in user space.[citation needed] In source code size, microkernels tend to be under 10,000 lines of code, as a general rule.
  A microkernel runs most of the operating system's background process in user space, to make the operating system more modular and, therefore, easier to maintain.
- A monolithic kernel is an operating system architecture where the entire operating system is working in kernel space and is alone in supervisor mode. The monolithic model differs from other operating system architectures (such as the microkernel architecture)[1][2] in that it alone defines a high-level virtual interface over computer hardware. A set of primitives or system calls implement all operating system services such as process management, concurrency, and memory management. Device drivers can be added to the kernel as modules.
  A monolithic kernel all the operating system instructions are executed in the same address space to improve the performance of the system.

Exercise 2.15 Layered Approach

It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.

Answer 2.15 (p 73)

The virtual memory subsystem and the storage subsystem are typically tightly coupled and requires careful design in a layered system due to the following interactions. Many systems allow files to be mapped into the virtual memory space of an executing process. On the other hand, the virtual memory subsystem typically uses the storage system to provide the backing store for pages that do not currently reside in memory. Also, updates to the file system are sometimes buffered in physical memory before it is flushed to disk, thereby requiring careful coordination of the usage of memory between the virtual memory subsystem and the file system.

Exercise 2.16 Microkernel

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

Answer 2.16 (p 73)

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

Exercise 2.17 Modular Kernel

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

Answer 2.17 (p 73)

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

Exercise 2.18 Kernel Problems

How are the problems with monolithic and microkernel structure solved?

Answer 2.18 (p 74)

Perhaps the best current methodology for operating-system design involves using object-oriented programming techniques to create a modular kernel. Here, the kernel has a set of core components and links in additional services either during boot time or during run time. Such a strategy uses dynamically loadable modules and is common in modern implementations of UNIX,such as Solaris, Linux, and MacOS X.
The overall result resembles a layered system in that each kernel section has defined, protected interfaces; but it is more flexible than a layered system in that any module can call any other module. Furthermore, the approach is like the microkernel approach in that the primary module has only core functions and knowledge of how to load and communicate with other modules; but it is more efficient, because modules do not need to invoke message passing in order to communicate.

Exercise 2.19 Virtual Machine

Describe the fundamental idea behind a virtual machine.

Answer 2.19 (p 76)

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.
The virtual machine provides an interface that is identical to the underlying bare hardware.

Exercise 2.20 Virtual Machine

Describe the benefits of using virtual machines.

Answer 2.20 (p 77)

- A major advantage of virtual machines in production data-center use is system consolidation, which involves taking two or more separate systems and running them in virtual machines on one system. Such physical-to-virtual conversions result in resource optimization
- They share the same hardware yet can run several different execution environments (that is,

different operating systems) concurrently
- The host system is protected from the virtual machines, just as the virtual machines are protected from each other. A virus inside a guest operating system might damage that operating system but is unlikely to affect the host or the other guests.

Exercise 2.21 Sharing Of Resources

Describe the two approaches of implementation to provide sharing of resources with virtual machines.

Answer 2.21 (p 77)

- It is possible to share a file-system volume and thus to share files
- It is possible to define a network of virtual machines and enable each machine to send information over the virtual communication network. The network is modeled after physical communication networks but is implemented in software.

Exercise 2.22 Virtual Machine

Describe a major disadvantage using virtual machines.

Answer 2.22 (p 77)

If the host system is off-line or must be taken off-line all the virtual machine will be off-line.

Exercise 2.23 Virtual Machine Architecture

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 2.23

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

Exercise 2.23 Virtual Machine Concept

Describe why it is difficult to implement the virtual machine concept.

Answer 2.23 (p 79)

- Much work is required to provide an exact duplicate of the underlying machine.
  Remember that the underlying machine typically has two modes: user mode and kernel mode. The virtual-machine software can run in kernel mode, since it is the operating system. The virtual machine itself can execute only in user mode.
- We must have a virtual user mode and a virtual kernel mode, both of which run in a physical user mode. Those actions that cause a transfer from user mode to kernel mode on a real machine (such as a system call or an attempt to execute a privileged instruction) must also cause a transfer from virtual user mode to virtual kernel mode on a virtual machine.
- Such a transfer can be accomplished as follows. When a system call, for example, is made by a program running on a virtual machine in virtual user mode, it will cause a transfer to the virtual-machine monitor in the real machine.
  When the virtual-machine monitor gains control, it can change the register contents and program counter for the virtual machine to simulate the effect of the system call. It can then restart the virtual machine, noting that it is now in virtual kernel mode.

Exercise 2.24 Compiler

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

Answer 2.24 (p 82)

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

Exercise 2.25 VMware

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

Answer 2.25

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

Exercise 2.26 Virtual Machine Concept

Describe in what differs simulation from the virtual machine concept.

Answer 2.26 (p 80)

simulation , in which the host system has one system architecture and the guest system was compiled for a different architecture.
The programs could be run in an emulator that translates each of the outdated system’s instructions into the native instruction set of the new system.
Instruction-set emulation can run an order of magnitude slower than native instructions.

Exercise 2.27 Para-virtualization

Describe the concept of para-virtualization.

Answer 2.27 (p 80)

Rather than try to trick a guest operating system into believing it has a system to itself, para-virtualization presents the guest with a system that is similar but not identical to the guest’s preferred system. The guest must be modified to run on the para-virtualized hardware. The gain for this extra work is more efficient use of resources and a smaller virtualization layer.

Exercise 2.28 Java Technology

Explain why Java is a technology and not just a programming language.

Answer 2.28 (p 81)

Because it provides more than a conventional programming language. Java technology consists of two essential components:

1. Programming-language specification
2. Virtual-machine specification

Exercise 2.29 Java Virtual Machine

Describe the working of the Java Virtual Machine.

Answer 2.29 (p 82)

- The Java virtual machine (JVM) is a specification for an abstract computer. It consists of a class loader and a Java interpreter that executes architecture-neutral bytecodes. The class loader loads the compiled .classfiles from both the Java program and the Java API for execution by the Java interpreter.
- After a class is loaded, it verifies that the .class file is valid Java bytecode. If the class passes verification, it is run by the Java interpreter.
- The JVM also automatically manages memory by performing garbage collection— the practice of reclaiming memory from objects no longer in use and returning it to the system.
- An instance of the JVM is created whenever a Java application or applet is run. This instance of the JVM starts running when the main()method of a program is invoked. In the case of applets the browser executes the main() method before creating the applet.
  If we simultaneously run two Java programs and a Java applet on the same computer, we will have three instances of the JVM.

Exercise 2.30 Debugging

Explain what debugging is and explain the two major activities.

Answer 2.30 (p 85)

Broadly, debugging is the activity of finding and fixing errors, or bugs,in a system. Debugging seeks to find and fix errors in both hardware and software.
Performance problems are considered bugs, so debugging can also include performance tuning, which improves performance by removing bottlenecks in the processing taking place within a system.

- • Failure Analysis
  - If a process fails, most operating systems write the error information to a log file to alert system operators or users that the problem occurred. The operating system can also take a core dump — a capture of the memory of the process and store it in a file for later analysis.
  - A kernel failure is called a crash. As with a process failure, error information is saved to a log file, and the memory state is saved to a crash dump.
- • Performance Tuning
  - To identify bottlenecks, we must be able to monitor system performance. Code must be added to compute and display measures of system behavior. In a number of systems, the operating system does this task by producing trace listings of system behavior.
  - Another approach to performance tuning is to include interactive tools with the system that allow users and administrators to question the state of various components of the system to look for bottlenecks.

Exercise 2.31 General-Purpose OS

Describe the procedure of starting a computer by loading the kernel on a computer with a general-purpose operating systems like Windows, Mac OS X,and UNIX.

Answer 2.31 (p 92)

- PCs use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel. the bootstrap loader is stored in firmware (ROM), and the operating system is on disk.
- When a CPU receives a reset event (e.g. powering on or reboot) the instruction register is loaded with a predefined memory location called the initial bootstrap program, and execution starts there.
- The initial bootstrap program is in the form of read-only memory (ROM, a.k.a. firmware). It runs diagnostics and read a single block at a fixed location from disk into memory and execute the code from that boot block
- The program stored in the boot block loads the entire operating system into memory and begin its execution.
  it is simple code and knows only the address on disk and length of the remainder of the bootstrap program. GRUB is an example of an open-source bootstrap program for Linux systems.
- The full bootstrap program has been loaded, it can traverse the file system to find the operating system kernel, load it into memory, and start its execution.
- At this point the system is running.

Exercise 2.32 Concurrent Processes

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

Answer 2.32

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

Exercise 2.33 System Call Interface

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

Answer 2.33 (p 55)

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

Exercise 2.34 Command Interpreter

What is the purpose of the command interpreter? Why is it usually separate from the kernel? Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

Answer 2.34 (p 52)

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

Chapter 3 Process Concepts

Exercise 3.1 Process and Program

What is the difference between a process and a program?

Answer 3.1 (p 103)

A program is a passive entity. Stored on a disk, a file containing a list of instructions, often called an executable file. Sometimes known as the text section.
A process is a program in execution, an active entity when the program is loaded into memory. It includes the current activity such as program counter, contents of processor registers, process stack, and data section.
And can been seen as a unit of work in a modern time-sharing system.
The term job and process are used almost interchangeable in this context, but process superseeds job.
Although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. For instance, several users may be running different copies of the mail program, or the same user may invoke many copies of the Web browser program.

Exercise 3.2 Process States

What are the five process states that defines the current activity of a process.

Answer 3.2 (p 105)

Each process may be in one of the following states:

- New: The process is being created.
- Running: Instructions are being executed.
- Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready: The process is waiting to be assigned to a processor.
- Terminated: The process has finished execution.

Exercise 3.3 Process And Thread

What is the difference between a process and a thread?

Answer 3.3 (p 107)

When a process is running, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at a time. Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time. On a system that supports threads, the PCB is expanded to include information for each thread.

Exercise 3.4 Time Sharing

What is the objective of time sharing.

Answer 3.4

The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

Exercise 3.5 Schedulers

Explain the three types of schedulers.

Answer 3.5 (p 110)

The long-term scheduler, or job scheduler , selects processes from this pool and loads them into memory for execution. The short-term scheduler,or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.
The primary distinction between these two schedulers lies in frequency of execution. The short-term scheduler must select a new process for the CPU frequently. A process may execute for only a few milliseconds before waiting for an I/O request. Often, the short-term scheduler executes at least once every 100 milliseconds. Because of the short time between executions, the short-term scheduler must be fast. If it takes 10 milliseconds to decide to execute a process for 100 milliseconds, then 10/(100 + 10) = 9 percent of the CPU is being used (wasted) simply for scheduling the work.
The long-term scheduler executes much less frequently; minutes may separate the creation of one new process and the next. The long-term scheduler controls the degree of multiprogramming (the number of processes in memory). If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling. This medium-term scheduler is diagrammed in Figure 3.8. The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called swapping

Exercise 3.6 I/O-bound and CPU-bound

Explain the difference between an I/O-bound and a CPU-bound process

Answer 3.6 (p 111)

An I/O -bound process is one that spends more of its time doing I/O than it spends doing computations. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.

Exercise 3.7 Context Switch

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

Answer 3.7 (p 112)

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.

Exercise 3.8 Process Creation

3 8 Q Describe the process creation in Unix, Windows, and Java

Answer 3.8 (p 114)

Unix: A new process is created by the fork() system call.
The new process consists of a copy of the address space of the original process.
Both processes (the parent and the child) continue execution at the instruction after the fork(), with one difference: the return code for the fork()is zero for the new (child) process, whereas the (nonzero) process identifier of the child is returned to the parent. Typically, the exec() system call is used after a fork() system call by one of the two processes to replace the process's memory space with a new program. The child process inherits privileges and scheduling attributes from the parent, as well as certain resources, such as open files.

Windows: Processes are created in the Win32 API using the CreateProcess() function.
However, whereas fork() has the child process inheriting the address space of its parent, CreateProcess() requires loading a specified program into the address space of the child process at process creation. Furthermore, whereas fork()is passed no parameters, CreateProcess() expects no fewer than ten parameters.

Java: When a Java program begins execution, an instance of the Java virtual machine is created.
On most systems, the JVM appears as an ordinary application running as a separate process on the host operating system. Each instance of the JVM provides support for multiple threads of control; but Java does not support a process model, which would allow the JVM to create several processes within the same virtual machine.
Java currently does not support a process model is that it is difficult to isolate one process's memory from that of another within the same virtual machine.
It is possible to create a process external to the JVM, however, by using the ProcessBuilder class, which allows a Java program to specify a process that is native to the operating system (such as /usr/bin/ls or C:\\ WINDOWS \\ system32\\ mspaint.exe).
Communication between the virtual machine and the external process occurs through the InputStream and OutputStream of the external process.

Exercise 3.9 Cooperating Process

Explain the terms independant- and cooperating process

Answer 3.9 (p 119)

A process is independent if it cannot affect or be affected by the other processes executing in the system.
Any process that does not share data with any other process is independent. A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

Exercise 3.10 Interprocess Communication

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

Answer 3.10 (p 120)

The two models of interprocess communication are (1) shared memory and (2) message passing.
The fundamental difference between the two models has to do with performance.

- A shared memory segment is set up through a system call. However, once a shared memory segment is established between two — or more — processes, the processes can communicate via the shared memory segment without any intervention from the kernel. This leads to an efficient mechanism for interprocess communication.

- Message passing on the other hand typically involves a system call when the send()and receive() operations are invoked. As a result, be cause the kernel is directly involved during interprocess communication, it typically is less efficient than shared memory. However, message passing can be used as a mechanism for synchronizing the actions between communicating processes. That is, the send() and receive() statements can be used to coordinate the actions of two communicating processes. Shared memory on the other hand provides no such process synchronization mechanisms.

Exercise 3.11 Process Cooperation

There are several reasons for providing an environment that allows process cooperation. Explain four of them.

Answer 3.11 (p 119)

- Information sharing. Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to such information.
- Computation speedup. If we want a particular task to run faster, we must break it into subtasks, each of which will execute in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing elements (such as CPUs or I/O channels).
- Modularity. We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads, as we discussed in Chapter 2.
- Convenience . Even an individual user may work on many tasks at the same time. For instance, a user may be editing, printing, and compiling in parallel.

Exercise 3.12 Interprocess Communication (IPC)

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information. Explain the two fundamental models of interprocess communication.

Answer 3.12 (p 120)

- In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes. Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network. For example, a chat program.

Exercise 3.13 Shared Memory

How can the shared memory support be implemented in Java?

Answer (p 121)

Although Java does not provide support for shared memory, we can design a solution to the producer – consumer problem in Java that emulates shared memory by allowing the producer and consumer processes to share an instance of the BoundedBuffer class (Figure 3.18), which implements the Buffer interface. Such sharing involves passing a reference to an instance of the BoundedBuffer class to the producer and consumer processes. This is illustrated in Figure 3.19.

The shared buffer is implemented as a circular array with two logical pointers: in and out. The variable in points to the next free position in the buffer; out points to the first full position in the buffer. The variable count is the number of items currently in the buffer. The buffer is empty when count ==0 and is full when count == BUFFER SIZE.
Note that both the producer and the consumer will block in the while loop if the buffer is not usable to them. In Chapter 6, we discuss how synchronization among cooperating processes can be implemented effectively in a shared-memory environment.

Exercise 3.14

There are several methods for logically implementing a link and the send()/ receive() operations. Name three.

Answer 3.14 (p 124)

- Direct communication. Each process that wants to communicate must explicitly name the recipient or sender of the communication.
- Synchronous or asynchronous communication. Communication between processes takes place through calls to send() and receive() primitives.
- Automatic or explicit buffering. Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue.

Exercise 3.15

There are several other strategies for communication in client–server systems. Explain three.

Answer 3.15 (p 131)

Sockets. A socket is defined as an endpoint for communication.
A socket is identified by an IP address concatenated with a port number.
Java provides three different types of sockets. Connection-oriented (TCP) sockets are implemented with the Socketclass. Connectionless (UDP) sockets use the Datagram Socketclass. Finally, the MulticastSocket class is a subclass of the DatagramSocket class. A multicast socket allows data to be sent to multiple recipients

Remote Procedure Calls (RPCs). The RPC was designed as away to abstract the procedure-call mechanism for use between systems with network connections. It is similar in many respects to the IPC mechanism described in Section 3.4, and it is usually built on top of such a system. Because we are dealing with an environment in which the processes are executing on separate systems, we must use a message-based communication scheme to provide remote service.

Java's Remote Method Invocation (RMI). RMI is a Java feature similar to RPC s. RMI allows a thread to invoke a method on a remote object. Objects are considered remote if they reside in a different Java virtual machine (JVM).

Exercise 3.16

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

Answer 3.16 (p 110)

a) Short-term (CPU scheduler) — selects from jobs in memory those jobs that are ready to execute and allocates t he CPU to them.
b) Medium-term— used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and re-instate them later to continue where they left off.

c) Long-term (job scheduler) — determines which jobs are brought into memory for processing. The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

Exercise 3.17

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

Answer 3.17 (p 112)

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.

Exercise 3.18

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

Answer 3.18 (p 136)

If an RPC mechanism cannot support either the“at most once” or “at least once” semantics, then the RPC server cannot guarantee that a remote procedure will not be invoked multiple occurrences. Consider if a remote procedure were withdrawing money from a bank account on a system that did not support these semantics. It is possible that a single invocation of the remote procedure might lead to multiple withdrawals on the server.
For a system to support either of these semantics generally requires the server maintain some form of client state such as the timestamp described in the text. If a system were unable to support either of these semantics, then such a system could only safely provide remote procedures that do not alter data or provide time-sensitive results. Using our bank account as an example, we certainly require “at most once” or “at least once” semantics for performing a withdrawal (or deposit!). However, an inquiry into an account balance or other account information such as name, address, etc. does not require these semantics.

Exercise 3.19

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

Answer 3.19

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

Exercise 3.20

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 3.20 (p 119)

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

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

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

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

Chapter 4 Multithreading

Exercise 4.1 Multithreading

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

Answer 4.1

1. Any kind of sequential program i s not a good candidate to be threaded. A n example of this is a program that calculates an individual tax return.
2. Another example i s a “shell” program such as the C -shell or Korn shell. Such a program must closely monitor its own working s pace such as ope n files , environment variables, and current working directory.

Exercise 4.2 Thread Library

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

Answer 4.2

Context s witching 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 s witching 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.

Exercise 4.3 Performance

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

Answer 4.3

When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand , will not be capable of performing useful work when a page fault takes place . Therefore , in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multithreaded solution would perform better even on a single-processor system.

Exercise 4.4 Multithreaded Process

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

a) Register values
b) Heap memory
c) Global variables
d) Stack memory

Answer 4.4

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

Exercise 4.5 Multithreaded Solution

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

Answer 4.5

A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there i s no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

Exercise 4.6 Java API

The Java API provides several different thread-pool architectures:

a) newFixedThreadPool(int)
b) newCachedThreadPool()
c) newSingleThreadExecutor()

Discuss t he merits of each.

Answer 4.6

No Answer

Exercise 4.7 Processes and Threads

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 where in the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

Answer 4.7

On one hand, in systems where processes and threads are considered as similar entities, some of the operating system code could be simplified. A scheduler, for instance , can consider the different processes and threads on an equal footing without requiring special code to examine the threads associated with a process during every scheduling step. On the other hand, this uniformity could make it harder to impose process-wide resource constraints in a direct manner. Instead, some extra complexity is required to identify which threads correspond to which process and perform the relevant accounting tasks.

Exercise 4.8 Pthreads API

The program shown in Figure 4.11 uses the Pthreads API. What would be output from the program at LINE C and LINE P?

Answer 4.8

Output at LINE C is 5. Output at LINE P is 0.

Exercise 4.9 Many-to-many Threading Model

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

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

Answer 4.9

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

Exercise 4.10 Prime Numbers

Write a multithreaded Java, Pthreads, or Win32 program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line . The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.

Answer 4.10

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

Exercise 4.11 Fibonacci

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

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

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

Answer 4.11

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

Exercise 4.12 Date Server

Modify the socket-based date server (Figure 3.19) in Chapter 3 so that the server services each client request in a separate thread.

Answer 4.12

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

Exercise 4.13 Java threading API

Exercise 3. 9 in Chapter 3 specifies designing an echo server using the Java threading API. However, this server is single-threaded, meaning the server cannot respond to concurrent echo clients until the current client exits. Modify the solution to Exercise 3.9 so that the echo server services each client in a separate request.

Answer 4.13

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

Exercise 4.14 Separate Thread

In Exercise 4.13, you are asked to service each client in a separate thread. Modify your solution so that each client is serviced using a thread pool, as discussed i n Section 4.5.4. Experiment with the three different models of thread-pool architecture presented in that section.

Answer 4.14

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

Exercise 4.15 Definition of a Thread

Give a definition of a thread.

Answer 4.15 (p 153)

A thread is a basic unit of CPU utilization; it comprises a threadID, a program counter, a register set, and a stack.

Exercise 4.16 Multithreaded Programming

What are the four benefits of multithreaded programming?

Answer 4.16 (p 155)

The benefits of multithreaded programming can be broken down into four major categories:

1. Responsiveness. Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. For instance, a multithreaded Web browser can allow user interaction in one thread while an image is being loaded in another thread.
2. Resource sharing. Processes may only share resources through techniques such as shared memory or message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
3. Economy. Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is much more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times slower than is creating a thread, and context switching is about five times slower.
4. Scalability. The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-

threaded process can only run on one processor, regardless how many are available. Multithreading on a multi-CPU machine increases parallelism. We explore this issue further in the following section.

Exercise 4.17 Multicore Systems

What are the five challenges in programming for multicore systems?

Answer 4.17 (p 156)

1. Dividing activities. This involves examining applications to find areas that can be divided into separate, concurrent tasks and thus can run in parallel on individual cores.
2. Balance. While identifying tasks that can run in parallel, programmers must also ensure that the tasks perform equal work of equal value. In some instances, a certain task may not contribute as much value to the overall process as other tasks; using a separate execution core to run that task may not be worth the cost.
3. Data splitting. Just as applications are divided into separate tasks, the data accessed and manipulated by the tasks must be divided to run on separate cores.
4. Data dependency. The data accessed by the tasks must be examined for dependencies between two or more tasks. In instances where one task depends on data from another, programmers must ensure that the execution of the tasks is synchronized to accommodate the data dependency. We examine such strategies in Chapter 6.
5. Testing and debugging. When a program is running in parallel on multiple cores, there are many different execution paths. Testing and debugging such concurrent programs is inherently more difficult than testing and debugging single-threaded applications.

Exercise 4.18 Multithreading

Describe the different model of multithreading and their advantages and disadvantages.

Answer 4.18 (p 157)

- Many-to-One Model
  Advantage: Thread management is done in user space, so it is efficient
  Disadvantage: the entire process will block if a thread makes a blocking system call. Only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.
- One-to-One Model
  Advantage: It provides more concurrency and it also allows multiple threads to run in parallel on multiprocessors.
  Disadvantage: Creating a user thread requires creating the corresponding kernel thread. which can burden the performance of an application
- Many-to-Many Model
  multiplexes many user-level threads to a smaller or equal number of kernel threads.
  Advantage: Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor. Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.
  Disadvantage: No specific disadvantages

Exercise 4.19 Java Threads

Explain the different types of Java threads.

Answer 4.19 (p 164)

Java actually identifies two different types of threads: (1) daemon (pronounced “demon”) and (2)

non-daemon threads.
The fundamental difference between the two types is the simple rule that the JVM shutsdown when all non-daemon threads have exited. When the JVM starts up, it creates several internal daemon threads for performing tasks such as garbage collection. A daemon thread is created by invoking the setDaemon() method of the Thread class.

Exercise 4.20 Thread States

Describe the six possible thread states in the JVM.

Answer 4.20 (p 166)

1. New. A thread is in this state when an object for the thread is created with the new command but the thread has not yet started.
2. Runnable. Calling the start() method allocates memory for the new thread in the JVM and calls the run() method for the thread object. When a thread's run() method is invoked, the thread moves from the new to the runnable state. A thread in the runnable state is eligible to be run by the JVM. Note that Java does not distinguish between a thread that is eligible to run and a thread that is currently running. A running thread is still in the runnable state.
3. Blocked . A thread is in this state as it waits to acquire a lock — a tool used for thread synchronization. We cover such tools in Chapter 6.
4. Waiting. A thread in this state is waiting for an action by another thread. For example, a thread invoking the join()method enters this state as it waits for the thread it is joining on to terminate.
5. Timed waiting . This state is similar to waiting, except a thread specifies the maximum amount of time it will wait. For example, the join() method has an optional parameter that the waiting thread can use to specify how long it will wait until the other thread terminates. Timed waiting prevents a thread from remaining in the waiting state indefinitely.
6. Terminated. exits run() method

Exercise 4

The fork() and exec() System Calls

Answer 4

Exercise 4 Cancellation

Describe the two different methods for cancellation of a thread.

Answer 4

A thread that is to be canceled is often referred to as the target thread. Cancellation of a target thread may occur in two different scenarios:
1. Asynchronous cancellation. One thread immediately terminates the target thread.
2. Deferred cancellation . The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Exercise 4 Signaling

Describe the differences between Windows and Unix signaling.

Answer 4

Although Windows does not explicitly provide support for signals, they can be emulated using asynchronous procedure calls (APCs). The APC facility allows a user thread to specify a function that is to be called when the user thread receives notification of a particular event. As indicated by its name, an APC is roughly equivalent to an asynchronous signal in UNIX. However, whereas UNIX must contend with how to deal with signals in a multithreaded environment, the APC facility is more straightforward, because an APC is delivered to a particular thread rather than a process.

Exercise 4 Thread Pool Architectures

Describe tree ways how the Java API java.util.concurrent provides thread-pool architectures.

Answer 4

The Java API provides several varieties of thread-pool architectures; we focus on the following three models, which are available as static methods in the java.util.concurrent.Executors class:

1. Single thread executor — newSingleThreadExecutor()— creates a pool of size 1.
2. Fixed thread executor —newFixedThreadPool(int size) —creates a thread pool with a specified number of threads.
3. Cached thread executor — newCachedThreadPool()— creates an unbounded thread pool, reusing threads in many instances.

Exercise 4 Thread Creation

Describe how Java provides thread-specific data when a developer has no control over the thread-creation process.

Answer 4

when the developer has no control over the thread-creation process — for example, when a thread pool is being used — then an alternative approach is necessary.

Exercise 4 Lightweight Process

Describe the function of the Lightweight process (LWP)

Answer 4

Many systems implementing either the many-to-many or two-level model place an intermediate data structure between the user and kernel threads. This data structure — typically known as a lightweight process, or LWP
To the user-thread library, the LWP appears to be a virtual processor on which the application can schedule a user thread to run. Each LWP is attached to a kernel thread, and it is kernel threads that the operating system schedules to run on physical processors. If a kernel thread blocks (such as while waiting for an I/O operation to complete), the LWP blocks as well. Up the chain, the user-level thread attached to the LWP also blocks.

Exercise 4 Multithreading

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

Exercise 4 Threads

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

Exercise 4 Context Switch

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

Exercise 4 Resources

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

Exercise 4 Mapping Threads

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.

Exercise 4 Summation Function

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

Exercise 4 Multithreading

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

Exercise 4 Thread Library

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

Exercise 4 Multithreaded

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?

Exercise 4 Shared

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

Exercise 4 Performance

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.

Exercise 4 Processes and Threads

4.12 As described in Section 4.6.2, 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.

Exercise 4 Output

4.13 The program shown in Figure 4.22 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

Exercise 4 Performance

4.14 Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be 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.

Chapter 5 Process Scheduling

Exercise 5.1. Process Scheduling

Explain why the term process scheduling is actually incorrect.

Answer 5.1 (p 193)

In Chapter 4, we introduced threads to the process model. On operating systems that support them, it is kernel-level threads — not processes — that are in fact being scheduled by the operating system. However, the terms process scheduling and thread scheduling are often used interchangeably.

Exercise 5.2 (non-) Preemptive

Explain the difference between preemptive and non-preemtive scheduling

Answer 5.2 (p 195)

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU,either by terminating or by switching to the waiting state.
A preemptive scheduler can switch between processes without the cooperation of them. Such a change is also known as a context switch.

Exercise 5.3 CPU Burst

Explain the term CPU burst.

Answer 5.3 (p 194)

A CPU burst is a cycle of CPU execution.

Exercise 5.4 CPU Scheduler and Dispatcher

Explain the difference between a CPU scheduler and a dispatcher.

Answer 5.4 (p 197)

The CPU scheduler (or short-term scheduler) selects one of the processes in the ready queue to be executed. The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

Exercise 5.5 Dispatch Latency

Explain the term dispatch latency

Answer 5.5 (p 197)

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

Exercise 5.6 CPU Scheduling

Explain five criteria for comparing CPU-scheduling.

Answer 5.6 (p 197)

- CPU utilization: We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent.
- Throughput: One measure of work is the number of processes that are completed per time unit, called throughput.
- Turnaround time: Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O .
- Waiting time: Waiting time is the sum of the periods spent waiting in the ready queue.
- Response time: Another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device.

Exercise 5.7 Average Turnaround Time

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

Process	Arrival Time	Burst Time
P1	0.0	7
P2	0.5	3
P3	1.0	2

Scheduling criteria, see page 197

- Arrival Time (AT): The first time of submission of a process
- Burst Time (BT): The time a process runs on a CPU
- Turnaround Time (TT): The interval from the time of submission of a process to the time of completion
- The sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O
- Waiting Time (WT): The sum of the periods spent spent in the ready queue

a) a) ATT with FCFS
What is the average turnaround time (ATT) for these processes with the Fist Come First Server (FCFS) scheduling algorithm?

b) ATT with SJF
What is the average turnaround time (ATT) for these processes with the Shortest Job First (SJB) scheduling algorithm?

c) ATT with SJB and idle
The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 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 P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

Answer 5.7 (p 179)

Exercise 5.8 Scheduling Algorithms

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

Process	Burst Time	Priority
P1	8	3
P2	1	1
P3	3	3
P4	1	4
P5	5	2

The process assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a) Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a smaller priority number implies a higher priority) and RR (quantum = 1)
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 these scheduling algorithms?
d) Which of the algorithms results in the minimum average waiting time (over all the processes)?

Answer 5.8 (p 197)

Exercise 5.9 Scheduling Algorithms

Which of t he following scheduling algorithms could result in starvation?

a) First-come, first-served
b) Shortest job first
c) Round robin
d) Priority

Answer 5.9

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

Chapter 6 Synchronization

Exercise 6.1 Critical Sections

Consider the two general approaches to handle critical sections in operating systems. Discuss the favor for the preemptive approach and the difficulties with SMP architectures.

Answer 6.1 (p 244)

Two general approaches are used to handle critical sections in operating systems: (1) preemptive kernels and (2) non-preemptive kernels. A preemptive kernel allows a process to be preempted while it is running in kernel mode.

- A non-preemptive kernel does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU. Obviously, a non-preemptive kernel is essentially free from race conditions on kernel data structures, as only one process is active in the kernel at a time. We cannot say the same about preemptive kernels, so they must be carefully designed to ensure that shared kernel data are free from race conditions.
- Preemptive kernels are especially difficult to design for SMP architectures, since in these environments it is possible for two kernel-mode processes to run simultaneously on different processors.

Why, then, would anyone favor a preemptive kernel over a non-preemptive one? A preemptive kernel is more suitable for real-time programming, as it will allow a real-time process to preempt a process currently running in the kernel.
Furthermore, a preemptive kernel may be more responsive, since there is less risk that a kernel-mode process will run for an arbitrarily long period before relinquishing the processor to waiting processes. Of course, this effect can be minimized by designing kernel code that does not behave in this way.

Exercise 6.2 Semaphores

Explain the differences between a counting semaphore and a binary semaphore.

Answer 6.2 (p 250)

The value of a counting semaphore can range over an unrestricted domain.
The value of a binary semaphore can range only between 0 and 1.
On some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion.
We can use binary semaphores to control access to the critical section for a process or thread.
Counting semaphores can be used for a variety of purposes, such as controlling access to a given resource consisting of a finite number of instances.
The semaphore is initialized to the number of resources available. Each thread that wishes to use a resource performs an acquire() operation on the semaphore (thereby decrementing t he count).
When a thread releases a resource, it performs are lease() operation (incrementing the count). When the count for the semaphore goes to 0, all resources are being used. After that, threads that wish to use a resource will block until the count becomes greater than 0.

Exercise 6.3. Busy Waiting

Describe how a program can overcome the need for busy waiting.

Answer 6.3 (p 252)

To overcome the need for busy waiting, we can modify the definitions of the acquire() and release() semaphore operations. When a process executes the acquire() operation and finds that the semaphore value is not positive, it must wait. However, rather than using busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then, control is transferred to the CPU scheduler, which selects another process to execute.
A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a release() operation. The process is restarted by a wakeup() operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue. (The CPU may or may not be switched from the running process to the newly ready process, depending on the CPU-scheduling algorithm.)

Exercise 6.4. TimedServer

Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections open at any point in time. After N connections have been made, the server will not accept another incoming connection until an existing connection is released. In the source code available on Wiley PLUS, there is a program named TimedServer.java that listens to port 2500. When a connection is made (via telnet or the supplied client program TimedClient.java), the server creates a new thread that maintains the connection for 10 seconds (writing the number of seconds remaining while the connection remains open). At the end of 10 seconds, the thread closes the connection.
Currently, TimedServer.java will accept an unlimited number of connections. Using semaphores, modify this program so that it limits the number of concurrent connections.

Answer 6.4 (p 305)

Exercise 6.5 Manager

Assume that a finite number of resources of a single resource type must be managed. Processes may ask for a number of these resources and — once finished — will return them. As an example, many commercial software packages provide a given number of licenses , indicating the number of applications that may run concurrently. When the application is started, the license count is decremented. When the application is terminated, the license count is incremented. If all licenses are in use, requests to start the application are denied. Such requests will only be granted when an existing license holder terminates the application and a license is returned.
The following Java class is used to manage a finite number of instances of an available resource. Note that when a process wishes to obtain a number of resources, it invokes the decreaseCount() method. Similarly, when a process wants to return a number of resources, it calls increaseCount() .

```
public class Manager {
   public static final int MAX RESOURCES = 5;
   private int availableResources = MAX RESOURCES;

  /**
   * Decrease availableResources by count resources.
   * return 0 if sufficient resources available,
   * otherwise return -1
   */

  public int decreaseCount(int count) {
      if (availableResources < count) return -1;
```

```
        else {
            availableResources -= count;
            return 0;
    }

    /* Increase availableResources by count resources. */
    public void increaseCount(int count) {
        availableResources += count;
    }
}
```

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

a. Identify the data involved in the race condition.

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

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

Answer 6.5