Exercise Solutions

Operating Systems (National University of Computer and Emerging Sciences)

INSTRUCTOR'S MANUAL
TO ACCOMPANY

Operating System Concepts

Ninth Edition

Abraham Silberschatz
Yale University

Peter Baer Galvin
Pluribus Networks

Greg Gagne
Westminster College

Contents

Preface

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

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

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

Introduction

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

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

Exercises

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

a. What are two such problems?

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

Answer:

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

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

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

a. Mainframe or minicomputer systems

b. Workstations connected to servers

c. Handheld computers

Answer:

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

b. Workstations: memory and CPU resources

c. Handheld computers: power consumption, memory resources

1.14 Under what circumstances would a user be better off using a time-sharing system rather than a PC or a single-user workstation?

Answer:

When there are few other users, the task is large, and the hardware is fast, time-sharing makes sense. The full power of the system can be brought to bear on the user's problem. The problem can be solved faster than on a personal computer. Another case occurs when lots of other users need resources at the same time.

A personal computer is best when the job is small enough to be executed reasonably on it and when performance is sufficient to execute the program to the user's satisfaction.

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

Answer:

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

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

Answer:

Clustered systems are typically constructed by combining multiple computers into a single system to perform a computational task distributed across the cluster. Multiprocessor systems on the other hand could be a single physical entity comprising of multiple CPUs. A clustered system is less tightly coupled than a multiprocessor system. Clustered systems communicate using messages, while processors in a multiprocessor system could communicate using shared memory.

In order for two machines to provide a highly available service, the state on the two machines should be replicated and should be consistently updated. When one of the machines fails, the other could then takeover the functionality of the failed machine.

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

Answer:

Consider the following two alternatives: **asymmetric clustering** and **parallel clustering**. With asymmetric clustering, one host runs the database application with the other host simply monitoring it. If the server fails, the monitoring host becomes the active server. This is appropriate for providing redundancy. However, it does not utilize the potential processing power of both hosts. With parallel clustering, the database application can run in parallel on both hosts. The difficulty in implementing parallel clusters is providing some form of distributed locking mechanism for files on the shared disk.

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

Answer:

A network computer relies on a centralized computer for most of its services. It can therefore have a minimal operating system to manage its resources. A personal computer on the other hand has to be capable of providing all of the required functionality in a stand-alone manner without relying on a centralized manner. Scenarios where administrative costs are high and where sharing leads to more efficient use of resources are precisely those settings where network computers are preferred.

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

Answer:

An interrupt is a hardware-generated change of flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction. A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A trap can be used to call operating system routines or to catch arithmetic errors.

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

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

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

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

Answer:
The CPU can initiate a DMA operation by writing values into special registers that can be independently accessed by the device. The device initiates the corresponding operation once it receives a command from the CPU. When the device is finished with its operation, it interrupts the CPU to indicate the completion of the operation.

Both the device and the CPU can be accessing memory simultaneously. The memory controller provides access to the memory bus in a fair manner to these two entities. A CPU might therefore be unable to issue memory operations at peak speeds since it has to compete with the device in order to obtain access to the memory bus.

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

Answer:
An operating system for a machine of this type would need to remain in control (or monitor mode) at all times. This could be accomplished by two methods:

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

b. Require that all programs be written in high-level languages so that all object code is compiler-produced. The compiler would generate (either in-line or by function calls) the protection checks that the hardware is missing.

1.22 Many SMP systems have different levels of caches; one level is local to each processing core, and another level is shared among all processing cores. Why are caching systems designed this way?

Answer:
The different levels are based on access speed as as well as size. In general, the closer the cache is to the CPU, the faster the access. However, faster caches are typically more costly. Therefore, smaller and faster caches are placed local to each CPU, and shared caches that are larger, yet slower, are shared among several different processors.

1.23 Consider an SMP system similar to the one shown in Figure 1.6. Illustrate with an example how data residing in memory could in fact have a different value in each of the local caches.

Answer:
Say processor 1 reads data A with value 5 from main memory into its local cache. Similarly, processor 2 reads data A into its local cache as well. Processor 1 then updates A to 10. However, since A resides in processor 1's local cache, the update only occurs there and not in the local cache for processor 2.

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

a. Single-processor systems

b. Multiprocessor systems

c. Distributed systems

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

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

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

1.26 Which network configuration—LAN or WAN—would best suit the following environments?

a. A campus student union

b. Several campus locations across a statewide university system

c. A neighborhood

Answer:

a. LAN

b. WAN

c. LAN or WAN

1.27 Describe some of the challenges of designing operating systems for mobile devices compared with designing operating systems for traditional PCs.

Answer:
The greatest challenges in designing mobile operating systems include:

- Less storage capacity means the operating system must manage memory carefully.
- The operating system must also manage power consumption carefully.

- Less processing power plus fewer processors mean the operating system must carefully apportion processors to applications.

1.28 What are some advantages of peer-to-peer systems over client-server systems?

Answer:

Peer-to-peer is useful because services are distributed across a collection of peers, rather than having a single, centralized server. Peer-to-peer provides fault tolerance and redundancy. Also, because peers constantly migrate, they can provide a level of security over a server that always exists at a known location on the Internet. Peer-to-peer systems can also potentially provide higher network bandwidth because you can collectively use all the bandwidth of peers, rather than the single bandwidth that is available to a single server.

1.29 Describe some distributed applications that would be appropriate for a peer-to-peer system.

Answer:

Essentially anything that provides content,in addition to existing services such as file services, distributed directory services such as domain name services, and distributed e-mail services.

1.30 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.

Answer:

Open source operating systems have the advantages of having many people working on them, many people debugging them, ease of access and distribution, and rapid update cycles. Further, for students and programmers there is certainly an advantage to being able to view and modify the source code. Typically open source operating systems are free for some forms of use, usually just requiring payment for support services. Commercial operating system companies usually do not like the competition that open source operating systems bring because these features are difficult to compete against. Some open source operating systems do not offer paid support programs. Some companies avoid open source projects because they need paid support, so that they have some entity to hold accountable if there is a problem or they need help fixing an issue. Finally, some complain that a lack of discipline in the coding of open source operating systems means that backward-compatiblity is lacking making upgrades difficult, and that the frequent release cycle exacerbates these issues by forcing users to upgrade frequently.

Operating System Structures

Chapter 2 is concerned with the operating-system interfaces that users (or at least programmers) actually see: system calls. The treatment is somewhat vague since more detail requires picking a specific system to discuss. This chapter is best supplemented with exactly this detail for the specific system the students have at hand. Ideally they should study the system calls and write some programs making system calls. This chapter also ties together several important concepts including layered design, virtual machines, Java and the Java virtual machine, system design and implementation, system generation, and the policy/mechanism difference.

Exercises

2.12 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.

Answer:

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

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

Answer:

a. Pass parameters in registers

b. Registers pass starting addresses of blocks of parameters

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

2.14 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.

Answer:

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

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

Answer:

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

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

Answer:

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

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

Answer:

An user should be able to develop a new command interpreter using the system-call interface provided by the operating system. The command interpreter allows an user to create and manage processes and also determine ways by which they communicate (such as through pipes and files). As all of this functionality could be accessed by an user-level program using the system calls, it should be possible for the user to develop a new command-line interpreter.

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

Answer:

The two models of interprocess communication are message-passing model and the shared-memory model. Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for intercomputer communication. Shared memory allows maximum speed and convenience of communication, since it can be done at memory transfer speeds when it takes place within a computer. However, this method compromises on protection and synchronization between the processes sharing memory.

2.19 Why is the separation of mechanism and policy desirable?

Answer:

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

2.20 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.

Answer:

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

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?

Answer:

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

associated with interprocess communication and the frequent use of the operating system's messaging functions in order to enable the user process and the system service to interact with each other.

2.22 What are the advantages of using loadable kernel modules?

Answer:

It is difficult to predict what features an operating system will need when it is being designed. The advantage of using loadable kernel modules is that functionality can be added to and removed from the kernel while it is running. There is no need to either recompile or reboot the kernel.

2.23 How are iOS and Android similar? How are they different?

Answer:

Similarities

- Both are based on existing kernels (Linux and Mac OS X).
- Both have architecture that uses software stacks.
- Both provide frameworks for developers.

Differences

- iOS is closed-source, and Android is open-source.
- iOS applications are developed in Objective-C, Android in Java.
- Android uses a virtual machine, and iOS executes code natively.

2.24 Explain why Java programs running on Android systems do not use the standard Java API and virtual machine.

Answer:

It is because the standard API and virtual machine are designed for desktop and server systems, not mobile devices. Google developed a separate API and virtual machine for mobile devices.

2.25 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 optimization of system performance.

Answer:

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

Processes

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

Exercises

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

Answer:

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

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

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

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

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

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

3.10 Construct a process tree similar to Figure 3.8. 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. The task manager on Windows systems does not provide the parent process id,yet the **process monitor** tool available from `technet.microsoft.com` provides a process tree tool.
Answer:
Answer: Results will vary widely.

3.11 Explain the role of the `init` process on UNIX and Linux systems in regards to process termination.
Answer:
When a process is terminated, it briefly moves to the zombie state and remains in that state until the parent invokes a call to `wait()`. When this occurs, the process id as well as entry in the process table are both released. However,if a parent does not invoke `wait()`, the child process remains a zombie as long as the parent remains alive. Once the parent process terminates, the `init`process becomes the new parent of the zombie. Periodically, the `init` process calls `wait()` which ultimately releases the pid and entry in the process table of the zombie process.

3.12 Including the initial parent process, how many processes are created by the program shown in Figure Figure 3.32?
Answer:
8 processes are created. The program online includes printf() statements to better understand how many processes have been created.

3.13 Explain the circumstances when the line of code marked `printf("LINE J")` in Figure 3.33 is reached.
Answer:
The call to `exec()` replaces the address space of the process with the program specified as the parameter to `exec()`. If the call to `exec()` succeeds, the new program is now running and control from the call to `exec()` never returns. In this scenario, the line `printf("Line J");` would never be performed. However, if an error occurs in the call to `exec()`, the function returns control and therefor the line `printf("Line J");` would be performed.

3.14 Using the program in Figure Figure 3.34, 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.)
Answer:
Answer: A = 0, B = 2603, C = 2603, D = 2600

3.15 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.

Answer:

Simple communication works well with ordinary pipes. For example, assume we have a process that counts characters in a file. An ordinary pipe can be used where the producer writes the file to the pipe and the consumer reads the files and counts the number of characters in the file. Next, for an example where named pipes are more suitable, consider the situation where several processes may write messages to a log. When processes wish to write a message to the log, they write it to the named pipe. A server reads the messages from the named pipe and writes them to the log file.

3.16 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.

Answer:

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

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

If a system were unable to support either of these 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.

3.17 Using the program shown in Figure 3.35, explain what the output will be at lines `X` and `Y`.

Answer:

Because the child is a copy of the parent, any changes the child makes will occur in its copy of the data and won't be reflected in the parent. As a result, the values output by the child at line X are 0, -1, -4, -9, -16. The values output by the parent at line Y are 0, 1, 2, 3, 4

3.18 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

Answer:

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

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

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

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