Introduction

Practice Exercises

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

Answer:

The three main puropses are:

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

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

Answer:

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

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

Answer:

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

1.4 Keeping in mind the various definitions of **operating system,** consider whether the operating system should include applications such as Web browsers and mail programs. Argue both that it should and that it should not, and support your answers.

Answer:

An argument in favor of including popular applications with the operating system is that if the application is embedded within the operating system, it is likely to be better able to take advantage of features in the kernel and therefore have performance advantages over an application that runs outside of the kernel. Arguments against embedding applications within the operating system typically dominate however: (1) the applications are applications - and not part of an operating system, (2) any performance benefits of running within the kernel are offset by security vulnerabilities, (3) it leads to a bloated operating system.

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

Answer:

The distinction between kernel mode and user mode provides a rudimentary form of protection in the following manner. Certain instructions could be executed only when the CPU is in kernel mode. Similarly, hardware devices could be accessed only when the program is executing in kernel mode. Control over when interrupts could be enabled or disabled is also possible only when the CPU is in kernel mode. Consequently, the CPU has very limited capability when executing in user mode, thereby enforcing protection of critical resources.

1.6 Which of the following instructions should be privileged?

a. Set value of timer.

b. Read the clock.

c. Clear memory.

d. Issue a trap instruction.

e. Turn off interrupts.

f. Modify entries in device-status table.

g. Switch from user to kernel mode.

h. Access I/O device.

Answer:

The following operations need to be privileged: Set value of timer, clear memory, turn off interrupts, modify entries in device-status table, access I/O device. The rest can be performed in user mode.

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

Answer:

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

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

Answer:

Although most systems only distinguish between user and kernel modes, some CPUs have supported multiple modes. Multiple modes could be used to provide a finer-grained security policy. For example, rather than distinguishing between just user and kernel mode, you could distinguish between different types of user mode. Perhaps users belonging to the same group could execute each other's code. The machine would go into a specified mode when one of these users was running code. When the machine was in this mode, a member of the group could run code belonging to anyone else in the group.

Another possibility would be to provide different distinctions within kernel code. For example, a specific mode could allow USB device drivers to run. This would mean that USB devices could be serviced without having to switch to kernel mode, thereby essentially allowing USB device drivers to run in a quasi-user/kernel mode.

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

Answer:

A program could use the following approach to compute the current time using timer interrupts. The program could set a timer for some time in the future and go to sleep. When it is awakened by the interrupt, it could update its local state, which it is using to keep track of the number of interrupts it has received thus far. It could then repeat this process of continually setting timer interrupts and updating its local state when the interrupts are actually raised.

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

Answer:

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

retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.

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

Answer:

The client-server model firmly distinguishes the roles of the client and server. Under this model, the client requests services that are provided by the server. The peer-to-peer model doesn't have such strict roles. In fact, all nodes in the system are considered peers and thus may act as *either* clients or servers—or both. A node may request a service from another peer, or the node may in fact provide such a service to other peers in the system.

For example, let's consider a system of nodes that share cooking recipes. Under the client-server model, all recipes are stored with the server. If a client wishes to access a recipe, it must request the recipe from the specified server. Using the peer-to-peer model, a peer node could ask other peer nodes for the specified recipe. The node (or perhaps nodes) with the requested recipe could provide it to the requesting node. Notice how each peer may act as both a client (it may request recipes) and as a server (it may provide recipes).

Operating System Structures

Practice Exercises

2.1 What is the purpose of system calls?

Answer:

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

2.2 What are the five major activities of an operating system with regard to process management?

Answer:

The five major activities are:

a. The creation and deletion of both user and system processes

b. The suspension and resumption of processes

c. The provision of mechanisms for process synchronization

d. The provision of mechanisms for process communication

e. The provision of mechanisms for deadlock handling

2.3 What are the three major activities of an operating system with regard to memory management?

Answer:

The three major activities are:

a. Keep track of which parts of memory are currently being used and by whom.

b. Decide which processes are to be loaded into memory when memory space becomes available.

c. Allocate and deallocate memory space as needed.

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

Answer:

The three major activities are:

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

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

Answer:

It reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls. It is usually not part of the kernel since the command interpreter is subject to changes.

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

Answer:

In Unix systems, a *fork* system call followed by an *exec* system call need to be performed to start a new process. The *fork* call clones the currently executing process, while the *exec* call overlays a new process based on a different executable over the calling process.

2.7 What is the purpose of system programs?

Answer:

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

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

Answer:

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

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

Answer:

The five services are:

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

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

should have access to and to access them only when they are otherwise unused.

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

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

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

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

Answer:

For certain devices, such as handheld PDAs and cellular telephones, a disk with a file system may be not be available for the device. In this situation, the operating system must be stored in firmware.

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

Answer:

Consider a system that would like to run both Windows XP and three different distributions of Linux (e.g., RedHat, Debian, and Mandrake). Each operating system will be stored on disk. During system boot-up, a special program (which we will call the **boot manager**) will determine which operating system to boot into. This means that rather initially booting to an operating system, the boot manager will first run during system startup. It is this boot manager that is responsible for determining which system to boot into. Typically boot managers must be stored at

certain locations of the hard disk to be recognized during system startup. Boot managers often provide the user with a selection of systems to boot into; boot managers are also typically designed to boot into a default operating system if no choice is selected by the user.

Processes

Practice Exercises

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

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

3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

Answer:
There are 8 processes created.

3.3 Original versions of Apple's mobile iOS operating system provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

Answer: FILL

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

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

3.5 When a process creates a new process using the fork() operation, which of the following state is shared between the parent process and the child process?

a. Stack

b. Heap

c. Shared memory segments

Answer:
Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process.

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

Answer:
The "exactly once" semantics ensure that a remore procedure will be executed exactly once and only once. The general algorithm for ensuring this combines an acknowledgment (ACK) scheme combined with timestamps (or some other incremental counter that allows the server to distinguish between duplicate messages).

The general strategy is for the client to send the RPC to the server along with a timestamp. The client will also start a timeout clock. The client will then wait for one of two occurrences: (1) it will receive an ACK from the server indicating that the remote procedure was performed, or (2) it will time out. If the client times out, it assumes the server was unable to perform the remote procedure so the client invokes the RPC a second time, sending a later timestamp. The client may not receive the ACK for one of two reasons: (1) the original RPC was never received by the server, or (2) the RPC was correctly received—and performed—by the server but the ACK was lost. In situation (1), the use of ACKs allows the server ultimately to receive and perform the RPC. In situation (2), the server will receive a duplicate RPC and it will use the timestamp to identify it as a duplicate so as not to perform the RPC a second time. It is important to note that the server must send a second ACK back to the client to inform the client the RPC has been performed.

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

Answer:
The server should keep track in stable storage (such as a disk log) information regarding what RPC operations were received, whether they were successfully performed, and the results associated with the operations. When a server crash takes place and a RPC message is received, the server can check whether the RPC had been previously performed and therefore guarantee "exactly once" semanctics for the execution of RPCs.

Threads

Practice Exercises

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

a. A Web server that services each request in a separate thread.

b. A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel.

c. An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

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

a. User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.

b. On systems using either M:1 or M:N mapping, user threads are scheduled by the thread library and the kernel schedules kernel threads.

c. Kernel threads need not be associated with a process whereas every user thread belongs to a process. Kernel threads are generally more expensive to maintain than user threads as they must be represented with a kernel data structure.

4.3 Describe the actions taken by a kernel to context-switch between kernel-level threads.
Answer:
Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

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

Answer:

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

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.

Answer:

Yes. Timing is crucial to real-time applications. If a thread is marked as real-time but is not bound to an LWP, the thread may have to wait to be attached to an LWP before running. Consider if a real-time thread is running (is attached to an LWP) and then proceeds to block (i.e. must perform I/O, has been preempted by a higher-priority real-time thread, is waiting for a mutual exclusion lock, etc.) While the real-time thread is blocked, the LWP it was attached to has been assigned to another thread. When the real-time thread has been scheduled to run again, it must first wait to be attached to an LWP. By binding an LWP to a real-time thread you are ensuring the thread will be able to run with minimal delay once it is scheduled.

Process Synchronization

Practice Exercises

5.1 In Section 5.4, we mentioned that disabling interrupts frequently can affect the system's clock. Explain why this can occur and how such effects can be minimized.

Answer:

The system clock is updated at every clock interrupt. If interrupts were disabled—particularly for a long period of time—it is possible the system clock could easily lose the correct time. The system clock is also used for scheduling purposes. For example, the time quantum for a process is expressed as a number of clock ticks. At every clock interrupt, the scheduler determines if the time quantum for the currently running process has expired. If clock interrupts were disabled, the scheduler could not accurately assign time quantums. This effect can be minimized by disabling clock interrupts for only very short periods.

5.2 Explain why Windows, Linux, and Solaris implement multiple locking mechanisms. Describe the circumstances under which they use spinlocks, mutex locks, semaphores, adaptive mutex locks, and condition variables. In each case, explain why the mechanism is needed.

Answer:

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

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

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

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

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

5.6 Illustrate how a binary semaphore can be used to implement mutual exclusion among n processes.
Answer:
The n processes share a semaphore, `mutex`, initialized to 1. Each process P_i is organized as follows:

```
do {
   wait(mutex);

      /* critical section */

   signal(mutex);

      /* remainder section */
} while (true);
```

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

- Two cars crossing a single-lane bridge from opposite directions.

- A person going down a ladder while another person is climbing up the ladder.
- Two trains traveling toward each other on the same track.

5.8 Is it possible to have a deadlock involving only a single process? Explain your answer.

Answer: No. This follows directly from the hold-and-wait condition.

Process Synchronization

Practice Exercises

6.1 In Section 6.4 we mentioned that disabling interrupts frequently could affect the system's clock. Explain why it could and how such effects could be minimized.

Answer: The system clock is updated at every clock interrupt. If interrupts were disabled—particularly for a long period of time—it is possible the system clock could easily lose the correct time. The system clock is also used for scheduling purposes. For example, the time quantum for a process is expressed as a number of clock ticks. At every clock interrupt, the scheduler determines if the time quantum for the currently running process has expired. If clock interrupts were disabled, the scheduler could not accurately assign time quantums. This effect can be minimized by disabling clock interrupts for only very short periods.

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

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

6.3 Give the reasons why Solaris, Windows XP, and Linux implement multiple locking mechanisms. Describe the circumstances under which they

use spinlocks, mutexes, semaphores, adaptive mutexes, and condition variables. In each case, explain why the mechanism is needed.

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

6.4 Explain the differences, in terms of cost, among the three storage types volatile, nonvolatile, and stable.

Answer: Volatile storage refers to main and cache memory and is very fast. However, volatile storage cannot survive system crashes or powering down the system. Nonvolatile storage survives system crashes and powered-down systems. Disks and tapes are examples of nonvolatile storage. Recently, USB devices using erasable program read-only memory (EPROM) have appeared providing nonvolatile storage. Stable storage refers to storage that technically can *never* be lost as there are redundant backup copies of the data (usually on disk).

6.5 Explain the purpose of the checkpoint mechanism. How often should checkpoints be performed? Describe how the frequency of checkpoints affects:

- System performance when no failure occurs
- The time it takes to recover from a system crash
- The time it takes to recover from a disk crash

Answer: A checkpoint log record indicates that a log record and its modified data has been written to stable storage and that the transaction need not to be redone in case of a system crash. Obviously, the more often checkpoints are performed, the less likely it is that redundant updates will have to be performed during the recovery process.

- System performance when no failure occurs—If no failures occur, the system must incur the cost of performing checkpoints that are essentially unnecessary. In this situation, performing checkpoints less often will lead to better system performance.
- The time it takes to recover from a system crash—The existence of a checkpoint record means that an operation will not have to be redone during system recovery. In this situation, the more often checkpoints were performed, the faster the recovery time is from a system crash.
- The time it takes to recover from a disk crash—The existence of a checkpoint record means that an operation will not have to be redone during system recovery. In this situation, the more often

checkpoints were performed, the faster the recovery time is from a disk crash.

6.6 Explain the concept of transaction atomicity.
Answer: A transaction is a series of read and write operations upon some data followed by a commit operation. If the series of operations in a transaction cannot be completed, the transaction must be aborted and the operations that did take place must be rolled back. It is important that the series of operations in a transaction appear as one indivisible operation to ensure the integrity of the data being updated. Otherwise, data could be compromised if operations from two (or more) different transactions were intermixed.

6.7 Show that some schedules are possible under the two-phase locking protocol but not possible under the timestamp protocol, and vice versa.
Answer: A schedule that is allowed in the two-phase locking protocol but not in the timestamp protocol is:

step	T_0	T_1	Precedence
1	**lock-S**(A)		
2	**read**(A)		
3		**lock-X**(B)	
4		**write**(B)	
5		**unlock**(B)	
6	**lock-S**(B)		
7	**read**(B)		$T_1 \rightarrow T_0$
8	**unlock**(A)		
9	**unlock**(B)		

This schedule is not allowed in the timestamp protocol because at step 7, the W-timestamp of B is 1.

A schedule that is allowed in the timestamp protocol but not in the two-phase locking protocol is:

step	T_0	T_1	T_2
1	**write**(A)		
2		**write**(A)	
3			**write**(A)
4	**write**(B)		
5		**write**(B)	

This schedule cannot have lock instructions added to make it legal under two-phase locking protocol because T_1 must unlock (A) between steps 2 and 3, and must lock (B) between steps 4 and 5.

6.8 The `wait()` statement in all Java program examples was part of a `while` loop. Explain why you would always need to use a `while` statement when using `wait()` and why you would never use an `if` statement.
Answer: This is an important issue to emphasize! Java only provides anonymous notification—you cannot notify a certain thread that a cer-

tain condition is true. When a thread is notified, it is its responsibility to re-check the condition that it is waiting for. If a thread did not re-check the condition, it might have received the notification without the condition having been met.

Main Memory

Practice Exercises

7.1 Name two differences between logical and physical addresses.

Answer:

A logical address does not refer to an actual existing address; rather, it refers to an abstract address in an abstract address space. Contrast this with a physical address that refers to an actual physical address in memory. A logical address is generated by the CPU and is translated into a physical address by the memory management unit(MMU). Therefore, physical addresses are generated by the MMU.

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

Answer:

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

7.3 Why are page sizes always powers of 2?

Answer:

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

7.4 Consider a logical address space of 64 pages of 1024 words each, mapped onto a physical memory of 32 frames.

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

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

Answer:

a. Logical address: 16 bits

b. Physical address: 15 bits

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

Answer:

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

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

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

Answer:

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

7.7 Sharing segments among processes without requiring that they have the same segment number is possible in a dynamically linked segmentation system.

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

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

Answer:

Both of these problems reduce to a program being able to reference both its own code and its data without knowing the segment or page number associated with the address. MULTICS solved this problem by associating four registers with each process. One register had the address of the current program segment, another had a base address for the stack, another had a base address for the global data, and so on. The idea is

that all references have to be indirect through a register that maps to the current segment or page number. By changing these registers, the same code can execute for different processes without the same page or segment numbers.

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

a. Bare machine

b. Single-user system

c. Multiprogramming with a fixed number of processes

d. Multiprogramming with a variable number of processes

e. Paging

f. Segmentation

Answer:

a. Protection not necessary, set system key to 0.

b. Set system key to 0 when in supervisor mode.

c. Region sizes must be fixed in increments of 2k bytes, allocate key with memory blocks.

d. Same as above.

e. Frame sizes must be in increments of 2k bytes, allocate key with pages.

f. Segment sizes must be in increments of 2k bytes, allocate key with segments.

Virtual Memory

Practice Exercises

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

Answer:

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

8.2 Assume that you have a page-reference string for a process with m frames (initially all empty). The page-reference string has length p; n distinct page numbers occur in it. Answer these questions for any page-replacement algorithms:

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

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

Answer:

a. n

b. p

8.3 Consider the page table shown in Figure 9.30 for a system with 12-bit virtual and physical addresses and with 256-byte pages. The list of free page frames is D, E, F (that is, D is at the head of the list, E is second, and F is last).

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. (A dash for a page frame indicates that the page is not in memory.)

- 9EF
- 111

- 700
- 0FF

Answer:

- $9EF$ - $0EF$
- 111 - 211
- 700 - $D00$
- $0FF$ - EFF

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

a. LRU replacement

b. FIFO replacement

c. Optimal replacement

d. Second-chance replacement

Answer:

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

8.5 Discuss the hardware support required to support demand paging.

Answer:

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

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

- 1 percent of all instructions executed accessed a page other than the current page.
- Of the instructions that accessed another page, 80 percent accessed a page already in memory.

- When a new page was required, the replaced page was modified 50 percent of the time.

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

Answer:

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

8.7 Consider the two-dimensional array `A`:

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

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

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

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

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

Answer:

a. 5,000

b. 50

8.8 Consider the following page reference string:

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

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

- LRU replacement
- FIFO replacement
- Optimal replacement

Answer:

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

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

Answer:

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

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

Answer:

No. An optimal algorithm will not suffer from Belady's anomaly because —by definition—an optimal algorithm replaces the page that will not be used for the longest time. Belady's anomaly occurs when a page-replacement algorithm evicts a page that will be needed in the immediate future. An optimal algorithm would not have selected such a page.

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

Answer:

a. **FIFO**. Find the first segment large enough to accommodate the incoming segment. If relocation is not possible and no one segment is large enough, select a combination of segments whose memories are contiguous, which are "closest to the first of the list" and which can accommodate the new segment. If relocation is possible, rearrange the memory so that the first N segments large enough for the incoming segment are contiguous in memory. Add any leftover space to the free-space list in both cases.

b. **LRU**. Select the segment that has not been used for the longest period of time and that is large enough, adding any leftover space to the free space list. If no one segment is large enough, select a combination of the "oldest" segments that are contiguous in memory (if relocation is not available) and that are large enough. If relocation is available, rearrange the oldest N segments to be contiguous in memory and replace those with the new segment.

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

a. CPU utilization 13 percent; disk utilization 97 percent

b. CPU utilization 87 percent; disk utilization 3 percent

c. CPU utilization 13 percent; disk utilization 3 percent

Answer:

a. Thrashing is occurring.

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

c. Increase the degree of multiprogramming.

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

Answer:

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

Mass Storage Structure

Practice Exercises

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

Answer:

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

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

Answer:

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

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

Answer:

Most disks do not export their rotational position information to the host. Even if they did, the time for this information to reach the scheduler would be subject to imprecision and the time consumed by the scheduler is variable, so the rotational position information would become incorrect. Further, the disk requests are usually given in terms

of logical block numbers, and the mapping between logical blocks and physical locations is very complex.

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

Answer:

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

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

Answer:

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

9.6 Is there any way to implement truly stable storage? Explain your answer.

Answer:

Truly stable storage would never lose data. The fundamental technique for stable storage is to maintain multiple copies of the data, so that if one copy is destroyed, some other copy is still available for use. But for any scheme, we can imagine a large enough disaster that all copies are destroyed.

9.7 It is sometimes said that tape is a sequential-access medium, whereas a magnetic disk is a random-access medium. In fact, the suitability of a storage device for random access depends on the transfer size. The term *streaming transfer rate* denotes the rate for a data transfer that is underway, excluding the effect of access latency. By contrast, the *effective transfer rate* is the ratio of total bytes per total seconds, including overhead time such as access latency.

Suppose that, in a computer, the level-2 cache has an access latency of 8 nanoseconds and a streaming transfer rate of 800 megabytes per second, the main memory has an access latency of 60 nanoseconds and a streaming transfer rate of 80 megabytes per second, the magnetic disk has an access latency of 15 milliseconds and a streaming transfer rate of 5 megabytes per second, and a tape drive has an access latency of 60 seconds and a streaming transfer rate of 2 megabytes per seconds.

a. Random access causes the effective transfer rate of a device to decrease, because no data are transferred during the access time. For the disk described, what is the effective transfer rate if an average access is followed by a streaming transfer of (1) 512 bytes, (2) 8 kilobytes, (3) 1 megabyte, and (4) 16 megabytes?

b. The utilization of a device is the ratio of effective transfer rate to streaming transfer rate. Calculate the utilization of the disk drive for each of the four transfer sizes given in part a.

c. Suppose that a utilization of 25 percent (or higher) is considered acceptable. Using the performance figures given, compute the smallest transfer size for disk that gives acceptable utilization.

d. Complete the following sentence: A disk is a random-access device for transfers larger than _____ bytes and is a sequential-access device for smaller transfers.

e. Compute the minimum transfer sizes that give acceptable utilization for cache, memory, and tape.

f. When is a tape a random-access device, and when is it a sequential-access device?

Answer:

a. For 512 bytes, the effective transfer rate is calculated as follows.
ETR = transfer size/transfer time.
If X is transfer size, then transfer time is ((X/STR) + latency).
Transfer time is 15ms + (512B/5MB per second) = 15.0097ms.
Effective transfer rate is therefore 512B/15.0097ms = 33.12 KB/sec.
ETR for 8KB = .47MB/sec.
ETR for 1MB = 4.65MB/sec.
ETR for 16MB = 4.98MB/sec.

b. Utilization of the device for 512B = 33.12 KB/sec / 5MB/sec = .0064 = .64
For 8KB = 9.4%.
For 1MB = 93%.
For 16MB = 99.6%.

c. Calculate .25 = ETR/STR, solving for transfer size X.
STR = 5MB, so 1.25MB/S = ETR.
1.25MB/S * ((X/5) + .015) = X.
.25X + .01875 = X.
X = .025MB.

d. A disk is a random-access device for transfers larger than K bytes (where K > disk block size), and is a sequential-access device for smaller transfers.

e. Calculate minimum transfer size for acceptable utilization of cache memory:
STR = 800MB, ETR = 200, latency = $8 * 10^{-9}$.
200 (XMB/800 + 8 X 10^{-9}) = XMB.
.25XMB + 1600 * 10^{-9} = XMB.
X = 2.24 bytes.
Calculate for memory:
STR = 80MB, ETR = 20, L = $60 * 10^{-9}$.
20 (XMB/80 + $60 * 10^{-9}$) = XMB.
.25XMB + $1200 * 10^{-9}$ = XMB.

X = 1.68 bytes.
Calculate for tape:
STR = 2MB, ETR = .5, L = 60s.
.5 (XMB/2 + 60) = XMB.
.25XMB + 30 = XMB.
X = 40MB.

f. It depends upon how it is being used. Assume we are using the tape to restore a backup. In this instance, the tape acts as a sequential-access device where we are sequentially reading the contents of the tape. As another example, assume we are using the tape to access a variety of records stored on the tape. In this instance, access to the tape is arbitrary and hence considered random.

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

Answer:

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

File-System Interface

Practice Exercises

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

Answer:

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

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

Answer:

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

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

Answer:
An advantage of having the system support different file structures is that the support comes from the system; individual applications are not required to provide the support. In addition, if the system provides the support for different file structures, it can implement the support presumably more efficiently than an application.

The disadvantage of having the system provide support for defined file types is that it increases the size of the system. In addition, applications that may require different file types other than what is provided by the system may not be able to run on such systems.

An alternative strategy is for the operating system to define no support for file structures and instead treat all files as a series of bytes. This is the approach taken by UNIX systems. The advantage of this approach is that it simplifies the operating system support for file systems, as the system no longer has to provide the structure for different file types. Furthermore, it allows applications to define file structures, thereby alleviating the situation where a system may not provide a file definition required for a specific application.

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

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

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

10.5 Explain the purpose of the `open()` and `close()` operations.

Answer:
The purpose of the `open()` and `close()` operations is:

- The `open()` operation informs the system that the named file is about to become active.
- The `close()` operation informs the system that the named file is no longer in active use by the user who issued the close operation.

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

a. Describe the protection problems that could arise.

b. Suggest a scheme for dealing with each of these protection problems.

Answer:

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

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

10.7 Consider a system that supports 5,000 users. Suppose that you want to allow 4,990 of these users to be able to access one file.

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

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

Answer:

a. There are two methods for achieving this:

   i. Create an access control list with the names of all 4990 users.

   ii. Put these 4990 users in one group and set the group access accordingly. This scheme cannot always be implemented since user groups are restricted by the system.

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

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

Answer:

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

CHAPTER 11

File-System Implementation

Practice Exercises

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

a. The block is added at the beginning.

b. The block is added in the middle.

c. The block is added at the end.

d. The block is removed from the beginning.

e. The block is removed from the middle.

f. The block is removed from the end.

Answer:
The results are:

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

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

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

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

Answer:

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

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

Answer:

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

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

Answer:

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

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

Answer:

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

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

Answer:

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

11.8 Explain how the VFS layer allows an operating system to support multiple types of file systems easily.

Answer:

VFS introduces a layer of indirection in the file system implementation. In many ways, it is similar to object-oriented programming techniques. System calls can be made generically (independent of file system type). Each file system type provides its function calls and data structures to the VFS layer. A system call is translated into the proper specific functions for the target file system at the VFS layer. The calling program has no file-system-specific code, and the upper levels of the system call structures likewise are file system-independent. The translation at the VFS layer turns these generic calls into file-system-specific operations.

File-System Implementation

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

Exercises

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

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

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

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

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

11.10 What are the advantages of the variation of linked allocation that uses a FAT to chain together the blocks of a file?
Answer: The advantage is that while accessing a block that is stored at the middle of a file, its location can be determined by chasing the

pointers stored in the FAT as opposed to accessing all of the individual blocks of the file in a sequential manner to find the pointer to the target block. Typically, most of the FAT can be cached in memory and therefore the pointers can be determined with just memory accesses instead of having to access the disk blocks.

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

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

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

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

Answer:

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

b. Reading the contents of the small local file */a/b/c* involves 4 separate disk operations: (1) Reading in the disk block containing the root directory /, (2) & (3) reading in the disk block containing the directories *b* and *c*, and reading in the disk block containing the file *c*.

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

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

11.13 Discuss how performance optimizations for file systems might result in difficulties in maintaining the consistency of the systems in the event of computer crashes.

Answer: The primary difficulty that might arise is due to delayed updates of data and metadata. Updates could be delayed in the hope that the same data might be updated in the future or that the updated data might be temporary and might be deleted in the near future. However, if the system were to crash without having committed the delayed updates, then the consistency of the file system is destroyed.

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

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

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

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

- **Contiguous**. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.
  a. Add X to Z to obtain the physical block number. Y is the displacement into that block.
  b. 1
- **Linked**. Divide the logical physical address by 511 with X and Y the resulting quotient and remainder respectively.
  a. Chase down the linked list (getting $X + 1$ blocks). $Y + 1$ is the displacement into the last physical block.
  b. 4
- **Indexed**. Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively.
  a. Get the index block into memory. Physical block address is contained in the index block at location X. Y is the displacement into the desired physical block.
  b. 2

11.15 Consider a file system that uses inodes to represent files. Disk blocks are 8-KB in size and a pointer to a disk block requires 4 bytes. This file system has 12 direct disk blocks, plus single, double, and triple indirect disk blocks. What is the maximum size of a file that can be stored in this file system?
Answer: (12 * 8 /KB/) + (2048 * 8 /KB) + (2048 * 2048 * 8 /KB/) + (2048 * 2048 * 2048 * 8 /KB) = 64 terabytes

11.16 Fragmentation on a storage device could be eliminated by recompaction of the information. Typical disk devices do not have relocation or base registers (such as are used when memory is to be compacted),

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

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

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

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

11.20 Consider the following backup scheme:

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

This contrasts to the schedule given in Section 11.7.4 by having all subsequent backups copy all files modified since the first full backup. What are the benefits of this system over the one in Section 11.7.4? What are the drawbacks? Are restore operations made easier or more difficult? Explain your answer.

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

I/O Systems

Practice Exercises

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

Answer:

Three advantages:

a. Bugs are less likely to cause an operating system crash

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

c. The kernel is simplified by moving algorithms out of it

Three disadvantages:

a. Bugs are harder to fix—a new firmware version or new hardware is needed

b. Improving algorithms likewise require a hardware update rather than just a kernel or device-driver update

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

12.2 The example of handshaking in Section 13.2 used 2 bits: a busy bit and a command-ready bit. Is it possible to implement this handshaking with only 1 bit? If it is, describe the protocol. If it is not, explain why 1 bit is insufficient.

Answer:

It is possible, using the following algorithm. Let's assume we simply use the busy-bit (or the command-ready bit; this answer is the same regardless). When the bit is off, the controller is idle. The host writes to data-out and sets the bit to signal that an operation is ready (the equivalent of setting the command-ready bit). When the controller is finished, it clears the busy bit. The host then initiates the next operation. This solution requires that both the host and the controller have read and write access to the same bit, which can complicate circuitry and increase the cost of the controller.

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

Answer:

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

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

Answer:

A hybrid approach could switch between polling and interrupts depending on the length of the I/O operation wait. For example, we could poll and loop N times, and if the device is still busy at N+1, we could set an interrupt and sleep. This approach would avoid long busy-waiting cycles. This method would be best for very long or very short busy times. It would be inefficient it the I/O completes at N+T (where T is a small number of cycles) due to the overhead of polling plus setting up and catching interrupts.

Pure polling is best with very short wait times. Interrupts are best with known long wait times.

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

Answer:

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

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

Answer:

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

rather than blindly increasing the performance of individual system components.

12.7 Distinguish between a STREAMS driver and a STREAMS module.

Answer:

The STREAMS driver controls a physical device that could be involved in a STREAMS operation. The STREAMS module modifies the flow of data between the head (the user interface) and the driver.

Mass Storage Structure

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

The basic implementation of disk scheduling should be fairly clear: requests, queues, servicing; so the main new consideration is the actual algorithms: FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK. Simulation may be the best way to involve the student with the algorithms. Exercise 12.21 provides a question amenable to a small but open-ended simulation study.

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

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

Exercises

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

a. Explain why this assertion is true.

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

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

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

Answer:

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

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

c. To prevent unusually long response times.

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

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

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

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

a. FCFS

b. SSTF

c. SCAN

d. LOOK

e. C-SCAN

Answer:

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

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

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

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

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

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

12.18 From elementary physics, we know that when an object is subjected to a constant acceleration a, the relationship between distance d and time t is given by $d = \frac{1}{2}at^2$. Suppose that, during a seek, the disk in Exercise 12.17 accelerates the disk arm at a constant rate for the first half of the seek, then decelerates the disk arm at the same rate for the second half of the seek. Assume that the disk can perform a seek to an adjacent cylinder in 1 millisecond and a full-stroke seek over all 5000 cylinders in 18 milliseconds.

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

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

c. Calculate the total seek time for each of the schedules in Exercise 12.17. Determine which schedule is the fastest (has the smallest total seek time).

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

Answer:

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

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

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

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

12.19 Suppose that the disk in Exercise 12.18 rotates at 7200 RPM.

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

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

Answer:

a. 7200 rpm gives 120 rotations per second. Thus, a full rotation takes 8.33 ms, and the average rotational latency (a half rotation) takes 4.167 ms.

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

12.20 Write a Java program for disk scheduling using the SCAN and C-SCAN disk-scheduling algorithms.
Answer: Please refer to the supporting Web site for source code solution.

12.21 Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request's service), the variation in response time, and the effective bandwidth. How does performance depend on the relative sizes of seek time and rotational latency?
Answer:
There is no simple analytical argument to answer the first part of this question. It would make a good small simulation experiment for the students. The answer can be found in Figure 2 of Worthington et al. [1994]. (Worthington et al. studied the LOOK algorithm, but similar results obtain for SCAN.) Figure 2 in Worthington et al. shows that C-LOOK has an average response time just a few percent higher than LOOK but that C-LOOK has a significantly lower variance in response time for medium and heavy workloads. The intuitive reason for the difference in variance is that LOOK (and SCAN) tend to favor requests near the middle cylinders, whereas the C-versions do not have this imbalance. The intuitive reason for the slower response time of C-LOOK is the "circular" seek from one end of the disk to the farthest request at the other end. This seek satisfies no requests. It causes only a small performance degradation because the square-root dependency of seek time on distance implies that a long seek isn't terribly expensive by comparison with moderate-length seeks.
For the second part of the question, we observe that these algorithms do not schedule to improve rotational latency; therefore, as seek times decrease relative to rotational latency, the performance differences between the algorithms will decrease.

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

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

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

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

Answer:

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

b. Here are some ideas. Place the hot data near the middle of the disk. Modify SSTF to prevent starvation. Add the policy that if the disk becomes idle for more than, say, 50 ms, the operating system generates an *anticipatory seek* to the hot region, since the next request is more likely to be there.

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

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

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

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

a. A write of one block of data

b. A write of seven continuous blocks of data

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

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

a. Read operations on single blocks

b. Read operations on multiple contiguous blocks

Answer: a) The amount of throughput depends on the number of disks in the RAID system. A RAID Level 5 comprising of a parity block for every set of four blocks spread over five disks can support four to five operations simultaneously. A RAID Level 1 comprising of two disks can support two simultaneous operations. Of course, there is greater flexibility in RAID Level 1 as to which copy of a block could be accessed and that could provide performance benefits by taking into account position of disk head. b) RAID Level 5 organization achieves greater bandwidth for accesses to multiple contiguous blocks since the adjacent blocks could be simultaneously accessed. Such bandwidth improvements are not possible in RAID Level 1.

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

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

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

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

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

once per week, once per day, once per hour, once per minute, or once per second?

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

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

Answer:

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

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

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

12.30 Discuss the relative advantages and disadvantages of sector sparing and sector slipping.

Answer:

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

12.31 Discuss the reasons why the operating system might require accurate information on how blocks are stored on a disk. How could the operating system improve file system performance with this knowledge?

Answer: While allocating blocks for a file, the operating system could allocate blocks that are geometrically close by on the disk if it had more information regarding the physical location of the blocks on the disk. In particular, it could allocate a block of data and then allocate the second block of data in the same cylinder but on a different surface at a rotationally optimal place so that the access to the next block could be made with minimal cost.

12.32 The operating system generally treats removable disks as shared file systems but assigns a tape drive to only one application at a time. Give three reasons that could explain this difference in treatment of disks and tapes. Describe additional features that would be required of the operating system to support shared file-system access to a tape jukebox. Would the applications sharing the tape jukebox need any

special properties, or could they use the files as though the files were disk-resident? Explain your answer.

Answer:

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

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

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

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

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

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

12.33 What would be the effect on cost and performance if tape storage were to achieve the same areal density as disk storage? (**Areal density** is the number of gigabits per square inch.)

Answer: To achieve the same areal density as a magnetic disk, the areal density of a tape would need to improve by two orders of magnitude. This would cause tape storage to be much cheaper than disk storage. The storage capacity of a tape would increase to more than 1 terabyte, which could enable a single tape to replace a jukebox of tapes in today's technology, further reducing the cost. The areal density has no direct bearing on the data transfer rate, but the higher capacity per tape might reduce the overhead of tape switching.

12.34 You can use simple estimates to compare the cost and performance of a terabyte storage system made entirely from disks with one that incorporates tertiary storage. Suppose that magnetic disks each hold 10 gigabytes, cost $1000, transfer 5 megabytes per second, and have an average access latency of 15 milliseconds. Suppose that a tape library costs $10 per gigabyte, transfers 10 megabytes per second, and has

an average access latency of 20 seconds. Compute the total cost, the maximum total data rate, and the average waiting time for a pure disk system. If you make any assumptions about the workload, describe and justify them. Now, suppose that 5 percent of the data are frequently used, so they must reside on disk, but the other 95 percent are archived in the tape library. Further suppose that 95 percent of the requests are handled by the disk system and the other 5 percent are handled by the library. What are the total cost, the maximum total data rate, and the average waiting time for this hierarchical storage system?

Answer: First let's consider the pure disk system. One terabyte is 1024 GB. To be correct, we need 103 disks at 10 GB each. But since this question is about approximations, we will simplify the arithmetic by rounding off the numbers. The pure disk system will have 100 drives. The cost of the disk drives would be $100,000, plus about 20% for cables, power supplies, and enclosures; that is, around $120,000. The aggregate data rate would be 100×5 MB/s, or 500 MB/s. The average waiting time depends on the workload. Suppose that the requests are for transfers of size 8 KB, and suppose that the requests are randomly distributed over the disk drives. If the system is lightly loaded, a typical request will arrive at an idle disk, so the response time will be 15 ms access time plus about 2 ms transfer time. If the system is heavily loaded, the delay will increase, roughly in proportion to the queue length.

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

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

12.35 Imagine that a holographic storage drive has been invented. Suppose that a holographic drive costs $10,000 and has an average access time of 40 milliseconds. Suppose that it uses a $100 cartridge the size of a CD. This cartridge holds 40,000 images, and each image is a square black-and-white picture with resolution 6000×6000 pixels (each pixel

stores 1 bit). Suppose that the drive can read or write 1 picture in 1 millisecond. Answer the following questions.

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

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

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

Answer: First, calculate performance of the device. 6000×6000 bits per millisecond = 4394 KB per millisecond = 4291 MB/sec(!). Clearly this is orders of magnitude greater than current hard disk technology, as the best production hard drives do less than 40 MB/sec. The following answers assume that the device cannot store data in smaller chunks than 4 MB.

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

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

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

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

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

Answer: The area of a 5.25-inch disk is about 19.625 square inches. If we suppose that the diameter of the spindle hub is 1.5 inches, the hub occupies an area of about 1.77 square inches, leaving 17.86 square

inches for data storage. Therefore, we estimate the storage capacity of the optical disk to be 2.2 gigabytes.

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

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

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

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

Protection

Practice Exercises

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

Answer:

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

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

Answer:

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

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

Answer:

D_j is a subset of D_i.

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

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

13.5 What protection problems may arise if a shared stack is used for parameter passing?
Answer:
The contents of the stack could be compromised by other process(es) sharing the stack.

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

13.7 Consider a computing environment where a process is given the privilege of accessing an object only n times. Suppose a scheme for implementing this policy.
Answer:
Add an integer counter with the capability.

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

13.9 Why is it difficult to protect a system in which users are allowed to do their own I/O?
Answer:
In earlier chapters we identified a distinction between kernel and user mode where kernel mode is used for carrying out privileged operations such as I/O. One reason why I/O must be performed in kernel mode is that I/O requires accessing the hardware and proper access to the hardware is necessary for system integrity. If we allow users to perform their own I/O, we cannot guarantee system integrity.

13.10 Capability lists are usually kept within the address space of the user. How does the system ensure that the user cannot modify the contents of the list?
Answer:
A capability list is considered a "protected object" and is accessed only indirectly by the user. The operating system ensures the user cannot access the capability list directly.

The Linux System

Practice Exercises

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

Answer:

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

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

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

Where a kernel is to be built that must be usable on a large variety of very different machines, then building it with modules is clearly preferable to using a single kernel with dozens of unnecessary drivers

consuming memory. This is particularly the case for commercially distributed kernels, where supporting the widest variety of hardware in the simplest manner possible is a priority.

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

15.2 Multithreading is a commonly used programming technique. Describe three different ways to implement threads, and compare these three methods with the Linux clone() mechanism. When might using each alternative mechanism be better or worse than using clones?

Answer:

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

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

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

In Linux, threads are implemented within the kernel by a clone mechanism that creates a new process within the same virtual address space as the parent process. Unlike some kernel-based thread packages, the Linux kernel does not make any distinction between threads and processes: a thread is simply a process that did not create a new virtual address space when it was initialized.

The main advantage of implementing threads in the kernel rather than in a user-mode library are that:

- kernel-threaded systems can take advantage of multiple processors if they are available; and
- if one thread blocks in a kernel service routine (for example, a system call or page fault), other threads are still able to run.

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

User-mode implementations do not have these advantages. Because such implementations run entirely within a single kernel execution

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

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

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

Answer:

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

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

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

15.4 Discuss three advantages of dynamic (shared) linkage of libraries compared with static linkage. Describe two cases in which static linkage is preferable.

Answer:

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

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

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

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

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

There are also performance advantages that sometimes make static linkage preferable in special cases. For a start, dynamic linkage does increase the startup time for a program, as the linking must now be

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

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

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

Answer:

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

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

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

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

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

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

Answer:

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

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

A second feature of modern disks is that their geometry can be very complex. The number of sectors per cylinder can vary according to the position of the cylinder: more data can be squeezed into the longer tracks nearer the edge of the disk than at the center of the disk. For an

operating system to optimize the rotational position of data on such disks, it would have to have complete understanding of this geometry, as well as the timing characteristics of the disk and its controller. In general, only the disk's internal logic can determine the optimal scheduling of I/Os, and the disk's geometry is likely to defeat any attempt by the operating system to perform rotational optimizations.

Distributed Coordination

Exercises

18.1 Discuss the advantages and disadvantages of the two methods we presented for generating globally unique timestamps.

Answer: Globally unique timestamps can be generated using either a centralized or distributed approach. The centralized approach uses a single site for generating timestamps. A disadvantage of this approach is that if this site fails, timestamps can no longer be produced.

Generating timestamps using the distributed approach provides more of a fail-safe mechanism, however care must be taken to ensure the logical clocks at each site are synchronized.

18.2 The logical clock timestamp scheme presented in this chapter provides the following guarantee: If event A happens before event B, then the timestamp of A is less than the timestamp of B. Note, however, that one cannot order two events based only on their timestamps. The fact that an event C has a timestamp that is less than the timestamp of event D does not necessarily mean that event C happened before event D; C and D could be concurrent events in the system. Discuss ways in which the logical clock timestamp scheme could be extended to distinguish concurrent events from events that can be ordered by the *happens-before* relationship.

Answer: Vector clocks could be used to distinguish concurrent events from events ordered by the happens-before relationship. A vector clock works as follows. Each process maintains a vector timestamp that comprises of a vector of scalar timestamp, where each element reflects the number of events that have occurred in each of the other processes in the system. More formally, a process i maintains a vector timestamp t_i such that t_i^j is equal to the number of events in process j that have oc-

curred before the current event in process i. When a local event occurs in process i, t_i^i is incremented by one to reflect that one more event has occurred in the process. In addition, any time a message is sent from a process to another process it communicates the timestamp vector of the source process to the destination process, which then updates its local timestamp vector to reflect the newly obtained information. More formally, when s sends a message to d, s communicates t_s along with the message and d updates t_d such that for all $i \neq d$, $t_d^i = max(t_s^i, t_d^i)$.

18.3 Your company is building a computer network, and you are asked to write an algorithm for achieving distributed mutual exclusion. Which scheme will you use? Explain your choice.
Answer: The options are a (1) centralized, (2) fully-distributed, or (3) token-passing approach. We reject the centralized approach as the centralized coordinator becomes a bottleneck. We also reject the token-passing approach for its difficulty in re-establishing the ring in case of failure.
We choose the fully-distributed approach for the following reasons:

- Mutual exclusion is obtained.
- Freedom from deadlock is ensured.
- Freedom from starvation is ensured, since entry to the critical section is scheduled according to the timestamp ordering.
- The number of messages per critical-section entry is $2 \times (n - 1)$. This number is the minimum number of required messages per critical-section entry when processes act independently and concurrently.

18.4 Why is deadlock detection much more expensive in a distributed environment than it is in a centralized environment?
Answer: The difficulty is that each site must maintain its own local wait-for graph. However, the lack of a cycle in a local graph does not ensure freedom from deadlock. Instead, we can only ensure the system is not deadlocked if the union of **all** local wait-for graphsis acyclic.

18.5 Your company is building a computer network, and you are asked to develop a scheme for dealing with the deadlock problem.

a. Would you use a deadlock-detection scheme or a deadlock-prevention scheme?

b. If you were to use a deadlock-prevention scheme, which one would you use? Explain your choice.

c. If you were to use a deadlock-detection scheme, which one would you use? Explain your choice.

Answer:

a. Would you use a deadlock-detection scheme, or a deadlock-prevention scheme?

We would choose deadlock prevention as it is systematically easier to prevent deadlocks than to detect them once they have occurred.

b. If you were to use a deadlock-prevention scheme, which one would you use? Explain your choice.
A simple resource-ordering scheme would be used; preventing deadlocks by requiring processes to acquire resources in order.

c. If you were to use a deadlock-detection scheme which one would you use? Explain your choice.
If we were to use a deadlock detection algorithm, we would choose a fully-distributed approach as the centralized approach provides for a single point of failure.

18.6 Under what circumstances does the wait–die scheme perform better than the wound–wait scheme for granting resources to concurrently executing transactions?
Answer: In the wound(–)wait scheme an older process never waits for a younger process; it instead rolls back the younger process and preempts its resources. When a younger process requests a resource held by an older process, it simply waits and there are no rollbacks. In the wait(–)die scheme, older processes wait for younger processes without any rollbacks but a younger process gets rolled back if it requests a resource held by an older process. This rollback might occur multiple times if the resource is being held by the older process for a long period of time. Repeated rollbacks do not occur in the wound(–)wait scheme. Therefore, the two schemes perform better under different circumstances depending upon when older processes are more likely to wait for resources held by younger processes or not.

18.7 Consider the centralized and the fully distributed approaches to deadlock detection. Compare the two algorithms in terms of message complexity.
Answer: The centralized algorithm for deadlock detection requires individual processors or sites to report its local waits-for graph to a centralized manager. The edges in the waits-for graph are combined and a cycle detection algorithm is performed in the centralized manager. The cost of this algorithm is the cost of communicating the various waits-for graph to the centralized server. In the distributed approach, each site builds its own local waits-for graph and predicts whether there is a possibility of a cycle based on local observations. If indeed there is a possibility of a cycle, a message is sent along the various sites that might constitute a cyclic dependency. Multiple sites that are potentially involved in the cyclic dependency could initiate this operation simultaneously. Therefore, the distributed algorithm is sometimes better than the centralized algorithm and is sometimes worse than the centralized algorithm in terms of message complexity. It is better than the centralized algorithm since it does not communicate the entire local waits-for graph to the centralized server. (Also note that that there could be performance bottlenecks due to the use of the centralized server in the centralized algorithm.) However, the distributed algorithm could

incur more messages when multiple sites are simultaneously exploring the existence of a cyclic dependency.

18.8 Consider the following *hierarchical* deadlock-detection algorithm, in which the global wait-for graph is distributed over a number of different *controllers*, which are organized in a tree. Each non-leaf controller maintains a wait-for graph that contains relevant information from the graphs of the controllers in the subtree below it. In particular, let S_A, S_B, and S_C be controllers such that S_C is the lowest common ancestor of S_A and S_B (S_C must be unique, since we are dealing with a tree). Suppose that node T_i appears in the local wait-for graph of controllers S_A and S_B. Then T_i must also appear in the local wait-for graph of

- Controller S_C
- Every controller in the path from S_C to S_A
- Every controller in the path from S_C to S_B

In addition, if T_i and T_j appear in the wait-for graph of controller S_D and there exists a path from T_i to T_j in the wait-for graph of one of the children of S_D, then an edge $T_i \rightarrow T_j$ must be in the wait-for graph of S_D.

Show that, if a cycle exists in any of the wait-for graphs, then the system is deadlocked.

Answer: A proof of this can be found in the article *Distributed deadlock detection algorithm* which appeared in ACM Transactions on Database Systems, Volume 7 , Issue 2 (June 1982), ages: 187 - 208.

18.9 Derive an election algorithm for bidirectional rings that is more efficient than the one presented in this chapter. How many messages are needed for n processes?

Answer: The following algorithm requires $O(n \log n)$ messages. Each node operates in phases and performs:

- If a node is still active, it sends its unique node identifier in both directions.
- In phase k, the tokens travel a distance of 2^k and return back to their points of origin.
- A token might not make it back to the originating node if it encounters a node with a lower unique node identifier.
- A node makes it to the next phase only if it receives its tokens back from the previous round.

The node with the lowest unique node identifies is the only that will be active after $\log n$ phases.

18.10 Consider a setting where processors are not associated with unique identifiers but the total number of processors is known and the processors are organized along a bidirectional ring. Is it possible to derive an election algorithm for such a setting?

Answer: It is impossible to elect a leader in a deterministic manner when the processors do not contain unique identifiers. Since different

processors have no distinguishing mark, they are assumed to start in the same state and transmit and receive the same messages in every timestep. Hence, there is no way to distinguish the processors even after an arbitarily large number of timesteps.

18.11 Consider a failure that occurs during 2PC for a transaction. For each possible failure, explain how 2PC ensures transaction atomicity despite the failure.
Answer: Possible failures include (1) failure of a participating site, (2) failure of the coordinator, and (3) failure of the network. We consider each approach in the following:

- **Failure of a Participating Site** - When a participating site recovers from a failure, it must examine its log to determine the fate of those transactions that were in the midst of execution when the failure occurred. The system will then accordingly depending upon the type of log entry when the failure occurred.
- **Failure of the Coordinator** - If the coordinator fails in the midst of the execution of the commit protocol for transaction *T,* then the participating sites must decide on the fate of *T.* The participating sites will then determine to either commit or abort *T* or wait for the recovery of the failed coordinator.
- **Failure of the Network** - When a link fails, all the messages in the process of being routed through the link do not arrive at their destination intact. From the viewpoint of the sites connected throughout that link, the other sites appears to have failed. Thus, either of the approaches discussed above apply.

18.12 Consider the following failure model for faulty processors. Processors follow the protocol but might fail at unexpected points in time. When processors fail, they simply stop functioning and do not continue to participate in the distributed system. Given such a failure model, design an algorithm for reaching agreement among a set of processors. Discuss the conditions under which agreement could be reached.
Answer: Assume that each node the following multicast functionality which we refer to as *basic multicast* or *b-multicast*. *b-multicast(v)*: node simply iterates through all of the nodes in the system and sends an unicast message to each node in the system containing v.
Also assume that each node performs the following algorithm in a synchronous manner assuming that node i starts with the value v_i.

- At round 1, the node performs b-multicast(v_i).
- In each round, the node gathers the values received since the previous round, computes the newly received values, and performs a b-multicast of all the newly received values.
- After round $f + 1$, if the number of failures is less than f, then each node has received exactly the same set of values from all of the other nodes in the system. In particular, this set of values includes all values from nodes that managed to send its local value to any of the nodes that do not fail.

The above algorithm works only in a synchronous setting and the message delays are bounded. It also works only when the messages are delivered reliably.