CHAPTER 11

Mass-Storage Structure

Practice Exercises

11.1 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.

11.2 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.

11.3 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.

11.4 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 bottlenecks in modern systems, as their individual performance cannot keep up with that of the CPU and system bus. When I/O is balanced among disks and controllers, neither an individual disk nor a controller is overwhelmed, so that bottleneck is avoided.

11.5 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.

11.6 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.

11.7 It is sometimes said that tape is a sequential-access medium, whereas a hard 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. In contrast, the ***effective transfer rate*** is the ratio of total bytes to total seconds, including overhead time such as access latency.

Suppose we have a computer with the following characteristics: 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 hard 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 second.

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 a 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 + $8 \text{ X } 10^{-9}$) = 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 on 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.

11.8 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. It could base this choice on the current location of the disk head and could therefore optimize performance by choosing the disk head that is closer to the target data.

11.9 Give three reasons to use HDDs as secondary storage.

Answer:
HDDs are still the most common secondary storage device.

a. They are the largest random-access storage device for the price, providing terabytes of storage at a low cost.

b. Many devices, from external chassis to storage arrays, are designed to use HDDs, providing a wide variety of ways to use HDDs.

c. They maintain the same read and write performance over their lifetime, unlike NVM storage devices, which lose write performance as they get full and as they age.

11.10 Give three reasons to use NVM devices as secondary storage.

Answer:
NVM devices are increasing in size and decreasing in price faster than HDDs.

a. High-speed NVM devices (including SSDs and usually not including USB drives) are much faster than HDDs. Secondary storage speed has a large impact on overall system performance.

b. NVM devices use less power than HDDs, making them very useful for laptops and other portable, battery-powered devices. NVM devices can also be much smaller than HDDs and thus can, for example, be surface-mounted to motherboards in devices like smartphones.

c. Because NVM devices have no moving parts, they tend to be much more reliable than HDDs.

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 requires a hardware update rather than just a kernel or device-driver update.

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

12.2 The example of handshaking in Section 12.2 used two bits: a busy bit and a command-ready bit. Is it possible to implement this handshaking with only one bit? If it is, describe the protocol. If it is not, explain why one 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 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 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 and would be best for very long or very short busy times. It would be inefficient if 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 the 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 that performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%. Doubling both system aspects would increase performance by 100%. Generally, it is important to remove the current system bottleneck, and to increase overall system performance, rather than blindly increasing the performance of individual system components.

12.7 Distinguish between a driver end and a stream module in a STREAMS operation.

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

File-System Interface

Practice Exercises

13.1 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 the user cannot lose files inadvertently by forgetting to save them.

13.2 Why do some systems keep track of the type of a file, while still 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 implement only basic file types to keep the operating system size smaller and allow maximum freedom to the processes on the system.

13.3 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 presumably implement the support 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 require file types other than what is provided by the system may not be able to run on the system.

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 avoiding the situation in which a system may not provide a file definition required for a specific application.

13.4 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 this 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.

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

Answer:

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

13.6 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.

13.7 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 4,990 users.

   ii. Put these 4,990 users in one group, and set the group access accordingly. This scheme cannot always be implemented, since the number of user groups and the number of members per group can be limited by the system.

b. The universal access to files applies to all users unless their names appear in the access-control list with different access permission. Thus, you can simply put the names of the remaining 10 users in the access-control list but give them no access privileges.

13.8 Researchers have suggested that, instead of having an access-control 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-based control list*. Since the access-control information is concentrated in one place, it is easier to change the information, and less space is required.

- *User-based control list*. This requires less overhead when opening a file.

CHAPTER 14

File-System Implementation

Practice Exercises

14.1 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

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

Answer:

In case of a 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.

14.3 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.

14.4 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.

14.5 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 devices they are caching for, so increasing the number or size of caches would increase system cost.

14.6 Why is it advantageous to 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 as a system grows—tables are never exceeded, avoiding artificial use limits. Unfortunately, kernel structures and code are more complicated, so there is more potential for bugs. Dynamic tables use more system resources than static tables, thus potentially taking system resources away from other parts of the system as the system grows.

CHAPTER 15

File-System Internals

Practice Exercises

15.1 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.

15.2 Why have more than one file system type on a given system?

Answer:
File systems can be designed and implemented with specific uses in mind, and optimized for those uses. Consider a virtual memory file system vs. a secondary storage file system. The memory-based one need not concern itself with fragmentation, or persisting data structures in the face of power loss. There are also special-purpose file systems like the procfs file system, designed to give the convenient file system interface to system aspects like the process name space and process resource use.

15.3 On a Unix or Linux system that implements the procfs file system, determine how to use the procfs interface to explore the process name space. What aspects of processes can be viewed via this interface? How would the same information be gathered on a system lacking the procfs file system?

Answer:

On systems containing the procfs psuedo-filesystem, details vary but generally the file system is mounted at /proc, and exploring it with file system commands can reveal the following:

- Each process is represented by its processID, so counting them reveals the number of processes in the system.
- Within each directory under the processID, details of the process state such as its current working directory, command line used to start the process, priority information, memory use information, lock information, open file information, and so on.
- Some procfs also provide interfaces to other kernel structures such as DMA structures, device lists, file system lists and so on.

See the `proc(5)` manual page for details on a given system.

Without procfs, to provide the same information, separate system calls per information type is used, or the ability to open the kernel memory space through `/dev/kmem` or `/dev/sys` is provided. Then programs using these interfaces need to be written to extract the data and present it in human-understandable form. See http://osxbook.com/book/bonus/ancient/procfs for a nice exploration of using procfs vs. not having it available.

15.4 Why do some systems integrate mounted file systems into the root file system naming structure, while others use a separate naming method for mounted file systems?

Answer: As with many aspects of operating system design, choices can be arbitrary or based on some small implementation detail and then exist long after any justify reason. Generally regarding file system mounting, integration with the root file system naming has proven to be more flexible and useful and separate mount point naming and therefore prevails on most file systems.

15.5 Given a remote file access facility such as `ftp`, why were remote file systems like NFS created?

Answer: Users of computer systems value ease-of-use in most cases. The more general purpose, and more widely used and operating system is, the more seamless its operation should be. In the case of remote file access, it is easier for users to use an familiar facility (such as a file system interface) rather than separate commands. And because file systems are tried and true, well integrated, and full featured, existing tools, scripts, and use cases can apply to remote file systems just as local file systems by using a file system interface for remote file access.

Security

Practice Exercises

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

Answer:
The contents of the stack could be compromised by any other processes sharing the stack.

17.2 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:
A hierarchical structure.

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

Answer:
Add an integer counter with the capability.

17.4 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.

17.5 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 whereby 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.

17.6 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 that the user cannot access the capability list directly.

Protection

Review Questions

Section 17.1

17.1 True or False? Policies determine how something will be done, mechanisms describe what will be done.

Section 17.2

17.2 What principle dictates that programs and users be given just enough privileges to perform their tasks?

Section 17.4

17.3 True or False? The need-to-know principle states that a process needs to know everything in order to perform its task.

17.4 True or False? The user mode/kernel mode paradigm is an example of domain switching.

17.5 True or False? In the UNIX operating system, a domain is associated with a process, not with a specific user.

17.6 What is the name of the bit in a UNIX system that associates each file with an owner and domain?

Section 17.5

17.7 What does each row and column represent in an access matrix?

Section 17.6

17.8 What is the problem when using a global table to represent an access matrix?

Section 17.8

17.9 What is the benefit of using role-based access control?