Solution for Homework Assignment 5

Silberschatz et al. exercise 11.1 Consider a file system where a file can be deleted and its disk space reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?

When the new file takes the place of the old file, the links to the old file will now point to the new file. Users might think they are accessing the old file when in fact they are accessing the new file. Depending on the file system, the protection mode for the old file might get used instead of the protection mode for the new file.

The solution is to remove all links to a file when it is deleted. One way to do this is to keep a list of links to a file with the file data. Then when the file is deleted, the links can easily be deleted as well. Another way to do this is to keep a reference count (as does the Unix inode) and only delete the file data when the reference count goes to 0.

Silberschatz et al. exercise 11.3 Why do some systems keep track of the type of a file, while others leave it to the user or simply do not implement multiple file types? Which system is better?

Having types allows the system to specify a different set of operations for each file type. For instance, an ASCII file type might support reading as a stream, and a database file type might support random access to individual database records. Some systems do not implement types because it keeps the file system interface component of the OS simple. On general-purpose operating systems, it is probably better to stick to basic file types only, or too much occasionally used, application specific code will end up in the kernel. On a more specialized system, such as a system running many database applications, however, it might be appropriate to build the most frequently used operations into the kernel.

Silberschatz et al. exercise 11.4 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?

The arguments here are the same as for file types in the previous exercise.

Silberschatz et al. exercise 11.5 What are the advantages and disadvantages of recording the name of the creating program with the file's attributes (as is done in MacOS)?

If the name of the creating program is stored with a file, the creating program can be invoked automatically whenever the file is selected. This metadata requires additional space and adds file creation/manipulation overhead to the kernel.

Silberschatz et al. exercise 11.6 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?

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.pascal.F1* specifies that *F1* is a file in subdirectory *pascal* 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.pascal.F1*) into shorter arbitrary names (such as *XX00743*), which are then used for actual file access.

Silberschatz et al. exercise 11.7 Explain the purpose of the `open` and `close` operations.

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.

Silberschatz et al. exercise 11.9 Give an example of an application in which data in a file should be accessed in the following order:

a. Sequentially

b. Randomly

(a) Print the contents of the file. (b) Print the contents of record i. This record can be found using hashing or index techniques.

Silberschatz et al. exercise 11.12 Consider a system that supports 5000 users. Suppose that you want to allow 4990 of these users to be able to access one file.

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

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

(a) Put the 4990 users in one group and set the group access accordingly. Unfortunately, this requires administrative access. (b) We could use general access control list in which the universe is granted access but the 10 remaining users are given no access.

Silberschatz et al. exercise 11.13 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.

In the first scheme, since the access control information is concentrated in one single place, it is easier to change access control information and less space is required. In the second scheme, there is potentially less kernel overhead in opening the file, since the kernel only has to verify the user control list information provided by the user process, not search an access control list.

Notice that these two schemes correspond to what, in Chapter 18, we called access control lists and capability lists, assuming the protection domains correspond to users.

Silberschatz et al. exercise 12.1 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 in the beginning, but there is room to grow in the end. 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.

	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

Silberschatz et al. exercise 12.2 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. Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.

(a) In order to reconstruct the free list, we would have to search the entire directory structure to determine which blocks are already allocated to files. Those remaining unallocated blocks could be relinked as the free-space list. (b) The free-space list pointer could be stored on the disk, perhaps in several places.

Silberschatz et al. exercise 12.4 Why must the bit map for file allocation be kept on mass storage, rather than in main memory?

If the system crashes, causing a memory failure, storing the free-space bit map on disk prevents it from getting lost. Also, for a very large disk, e.g. an 80 GB disk with 4 KB blocks, the 20 Mbit free block bit map might require too much memory.

Silberschatz et al. exercise 12.5 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?

Contiguous allocation is best for files that are small and accessed sequentially. Linked allocation is best for large files accessed sequentially. Indexed allocation is best for large, randomly accessed files.

Silberschatz et al. exercise 12.6 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 accessedwas block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

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

Contiguous: (a) Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively. Add X to Z to obtain the physical block number. Then Y is the displacement into that block. (b) 1.

Linked: (a) Divide the logical physical address by 511 with X and Y the resulting quotient and remainder respectively. Chase down the linked list (getting $X+1$ blocks). $Y+1$ is the displacement into the last physical block. (b) 4.

Indexed: (a) Divide the logical address by 512 with X and Y the resulting quotient and remainder respectively. First get the index block into memory. The physical block address is contained in the index block at location X. Y is the displacement into the desired physical block. (b) 2.

Silberschatz et al. exercise 12.8 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.

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.

Silberschatz et al. exercise 13.1 State three advantages of placing functionality in a device controller, rather than in the kernel. State three disadvantages.

Advantages: 1) Bugs are less likely to cause an operating system crash. 2) Performance can be improved by utilizing dedicated hardware and hard-coded algorithms. 3) The kernel is simplified by moving algorithms out of it.

Disadvantages: 1) Bugs are harder to fix — a new firmware version or new hardware is needed. 2) Improving algorithms likewise requires a hardware update rather than just kernel or device driver update. 3) Embedded algorithms could conflict with an application's use of the device, causing decreased performance.

From Silberschatz et al. exercise 13.2 For each of the following I/O scenarios, would you use polled I/O or interrupt-driven I/O? Give a reason for your choices.

a. A mouse used with a graphical user interface.

b. A tape drive on a multitasking operating system.

c. A disk drive containing user files.

d. A graphics card with direct bus connection, accessible through memory-mapped I/O.

Interrupt-driven I/O is best for items (a), (b), and (c). A mouse generates small amounts of data in bursts, so wasting time polling is a bad idea. Tape drives and disk drives also have high latency and bursty outputs and therefore benefit from interrupt-driven I/O (DMA to be specific).

The only time polling is a good idea is when the data are transferred at a fairly slow, steady rate, and prompt responses are not necessarily critical. An example is writing output to a printer.

For part (d), the I/O is handled automatically by the memory mapping architecture, so neither polling nor interrupts are necessary.

Silberschatz et al. exercise 13.3 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.

During the protocol, the busy bit is used to inform the host whether the controller is busy or not. The host cannot send a new command until the controller is no longer busy. Therefore, once the controller turns off the busy bit, that bit could be reused by the host to inform the controller that another command is ready. The controller would then notice that the bit has been set and start processing the request, then clear the bit when it is done. Let's call the single bit `bsy-cmd` and modify the protocol as follows:

1. The host reads the `bsy-cmd` bit until it becomes clear.
2. Unchanged.
3. The host sets the `bsy-cmd` bit.
4. The controller notices that the `bsy-cmd` bit has been set.
5. Unchanged.

6. The controller clears the `error` bit in the status register to indicate that the I/O succeeded then clears the `bsy-cmd` bit to indicate that it is finished.

Silberschatz et al. exercise 13.4 Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which nonblocking I/O should be used. Why not just implement nonblocking I/O and have processes busy-wait until their device is ready?

Generally, blocking I/O is appropriate when the process will only be waiting for one specific event. Examples include a disk, tape, or keyboard read by an application program. Non-blocking I/O is useful when I/O may come from more than one source and the order of the I/O arrival is not predetermined. Examples include network daemons listening to more than one network socket, window managers that accept mouse movement as well as keyboard input, and I/O-management programs, such as a copy command that copies data between I/O devices. In the last case, the program could optimize its performance by buffering the input and output and using non-blocking I/O to keep both devices fully occupied. Non-blocking I/O is more complicated for programmers, because of the asynchonous rendezvous that is needed when an I/O occurs. Also, busy waiting is less efficient than interrupt-driven I/O so the overall system performance would decrease.

Silberschatz et al. exercise 13.8 How does DMA increase system concurrency? How does it complicate hardware design?

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. 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.

Silberschatz et al. exercise 13.10 Why is it important to scale up system bus and device speeds as the CPU speed increases?

Consider a system which performs 50% I/O and 50% computation. 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.

Silberschatz et al. exercise 14.1 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.

(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.

Silberschatz et al. exercise 14.2 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.

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.

Silberschatz et al. exercise 14.8 Is disk scheduling, other than FCFS scheduling, useful in a single-user environment? Explain your 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.

Silberschatz et al. exercise 14.9 Explain why SSTF scheduling tends to favor middle cylinders over the innermost and outermost cylinders.

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.