CHAPTER 10

File-System Interface

Files are the most obvious object that operating systems manipulate. Everything is typically stored in files: programs, data, output, etc. The student should learn what a file is to the operating system and what the problems are (providing naming conventions to allow files to be found by user programs, protection).

Two problems can crop up in this chapter. First, terminology may be different between your system and the book. This can be used to drive home the point that concepts are important and terms must be clearly defined when you get to a new system. Second, it may be difficult to motivate students to learn about directory structures that are not the ones on the system they are using. This can best be overcome if the students have two very different systems to consider, such as a single-user system for a microcomputer and a large, university time-shared system.

Projects might include a report about the details of the file system for the local system. It is also possible to write programs to implement a simple file system either in memory (allocate a large block of memory that is used to simulate a disk) or on top of an existing file system. In many cases, the design of a file system is an interesting project of its own.

Exercises

10.10 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?

Answer: Let F1 be the old file and F2 be the new file. A user wishing to access F1 through an existing link will actually access F2. Note that the access protection for file F1 is used rather than the one associated with F2.

This problem can be avoided by insuring that all links to a deleted file are deleted also. This can be accomplished in several ways:

a. maintain a list of all links to a file, removing each of them when the file is deleted

b. retain the links, removing them when an attempt is made to access a deleted file

c. maintain a file reference list (or counter), deleting the file only after all links or references to that file have been deleted

10.11 The open-file table is used to maintain information about files that are currently open. Should the operating system maintain a separate table for each user or just maintain one table that contains references to files that are being accessed by all users at the current time? If the same file is being accessed by two different programs or users, should there be separate entries in the open file table?
Answer: By keeping a central open-file table, the operating system can perform the following operation that would be infeasible otherwise. Consider a file that is currently being accessed by one or more processes. If the file is deleted, then it should not be removed from the disk until all processes accessing the file have closed it. This check can be performed only if there is centralized accounting of number of processes accessing the file. On the other hand, if two processes are accessing the file, then two separate states need to be maintained to keep track of the current location of which parts of the file are being accessed by the two processes. This requires the operating system to maintain separate entries for the two processes.

10.12 What are the advantages and disadvantages of a system providing mandatory locks instead of providing advisory locks whose usage is left to the users' discretion?
Answer: In many cases, separate programs might be willing to tolerate concurrent access to a file without requiring the need to obtain locks and thereby guaranteeing mutual exclusion to the files. Mutual exclusion could be guaranteed by other program structures such as memory locks or other forms of synchronization. In such situations, the mandatory locks would limit the flexibility in how files could be accessed and might also increase the overheads associated with accessing files.

10.13 What are the advantages and disadvantages of recording the name of the creating program with the file's attributes (as is done in the Macintosh Operating System)?
Answer: By recording the name of the creating program, the operating system is able to implement features (such as automatic program invocation when the file is accessed) based on this information. It does add overhead in the operating system and require space in the file descriptor, however.

10.14 Some systems automatically open a file when it is referenced for the first time, and close the file when the job terminates. Discuss the advantages and disadvantages of this scheme as compared to the more traditional one, where the user has to open and close the file explicitly.
Answer: Automatic opening and closing of files relieves the user from the invocation of these functions, and thus makes it more convenient to the user; however, it requires more overhead than the case where explicit opening and closing is required.

10.15 If the operating system were to know that a certain application is going to access the file data in a sequential manner, how could it exploit this information to improve performance?
Answer: When a block is accessed, the file system could prefetch the subsequent blocks in anticipation of future requests to these blocks. This prefetching optimization would reduce the waiting time experienced by the process for future requests.

10.16 Give an example of an application that could benefit from operating system support for random access to indexed files.
Answer: An application that maintains a database of entries could benefit from such support. For instance, if a program is maintaining a student database, then accesses to the database cannot be modeled by any predetermined access pattern. The access to records are random and locating the records would be more efficient if the operating system were to provide some form of tree-based index.

10.17 Discuss the merits and demerits of supporting links to files that cross mount points (that is, the file link refers to a file that is stored in a different volume).
Answer: The advantage is that there is greater transparency in the sense that the user does not need to be aware of mount points and create links in all scenarios. The disadvantage however is that the file system containing the link might be mounted while the file system containing the target file might not be, and therefore one cannot provide transparent access to the file in such a scenario; the error condition would expose to the user that a link is a dead link and that the link does indeed cross file system boundaries.

10.18 Some systems provide file sharing by maintaining a single copy of a file; other systems maintain several copies, one for each of the users sharing the file. Discuss the relative merits of each approach.
Answer: With a single copy, several concurrent updates to a file may result in user obtaining incorrect information, and the file being left in an incorrect state. With multiple copies, there is storage waste and the various copies may not be consistent with respect to each other.

10.19 Discuss the advantages and disadvantages of associating with remote file systems (stored on file servers) a different set of failure semantics from that associated with local file systems.
Answer: The advantage is that the application can deal with the failure condition in a more intelligent manner if it realizes that it incurred an error while accessing a file stored in a remote file system. For instance, a file open operation could simply fail instead of hanging when accessing a remote file on a failed server and the application could deal with the failure in the best possible manner; if the operation were simply to hang, then the entire application hangs, which is not desirable. The disadvantage however is the lack of uniformity in failure semantics and the resulting complexity in application code.

10.20 What are the implications of supporting UNIX consistency semantics for shared access for those files that are stored on remote file systems?

Answer: UNIX consistency semantics requires updates to a file to be immediately available to other processes. Supporting such a semantics for shared files on remote file systems could result in the following inefficiencies: all updates by a client have to be immediately reported to the file server instead of being batched (or even ignored if the updates are to a temporary file), and updates have to be communicated by the file server to clients caching the data immediately, again resulting in more communication.