File System

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.

10.1 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.

10.2 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.

10.3

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.

10.4 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.

10.5 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.

10.6 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.

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

a. The `open()` operation informs the system that the named file is about to become active.

b. The `close()` operation informs the system that the named file is no longer in active use by the user who issued the close operation.

10.8 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.

10.9 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.

10.10 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.

10.11 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.

10.12 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.

10.13 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.

10.14 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

10.15 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.

10.16 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.