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.