OPERATING SYSTEM
IT-42033

CHAPTER – 10
Q&A

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. 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.

10.7

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

- 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.

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.

Q. Name and explain a file's attributes vary from one operating system to another.

- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type:** This information is needed for systems that support different types of files.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification:** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

Q. The operating system provide system calls to create, write, read, reposition, delete and truncate files. In there, Describe about creating file, reading file, writing file and deleting file.

- **Creating a file:** Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.
- **Writing a file:** To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
- **Reading a file:** To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.
- **Deleting a file:** To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.

Q. Define file pointer, file-open count, disk location of the file, and access rights.

- **File pointer:** On systems that do not include a file offset as part of the read() and write() system calls, the system must track the last read write location as a current-file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes.
- **File-open count:** As files are closed, the operating system must reuse its open-file table entries, or it could run out of space in the table. Because multiple processes may have opened a file, the system must wait for the last file to close before removing the open-file table entry. The file-open counter tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.
- **Disk location of the file:** Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.
- **Access rights:** Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/0 requests.

Q. Define shared lock and exclusive lock.

- A shared lock is akin to a reader lock in that several processes can acquire the lock concurrently.
- An exclusive lock behaves like a writer lock; only one process at a time can acquire such a lock.

Q. Explain File Structure.

- Assuming a system supports two types of files: text files and executable binary files. Now, if users want to define an encrypted file to protect the contents from being read by unauthorized people, we may find neither file type to be appropriate. The encrypted file is not ASCII text lines but rather is random bits. Although it may appear to be a binary file, it is not executable.
- Locating an offset within a file can be complicated for the operating system. Disk systems typically have a well-defined block size determined by the size of a sector. All disk I/0 is performed in units of one block (physical record), and all blocks are the same size. It is unlikely that the physical record size will exactly match the length of the desired logical record.. Logical records even vary in length. Packing a number of logical records into physical blocks is a common solution to this problem.
- For example, the UNIX operating system defines all files to be simply streams of bytes. Each byte is individually addressable by its offset from the beginning (or end) of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks say, 512 bytes per block-as necessary. The logical record size, physical block size, and packing technique determine how many logical records are in each physical block. The packing can be done either by the user's application program or by the operating system. In either case, the file is considered a sequence of blocks. All the basic I/O functions operate in terms of blocks. The conversion from logical records to physical blocks is a relatively simple software problem.

Q. What is the difference between sequential access and direct access method?

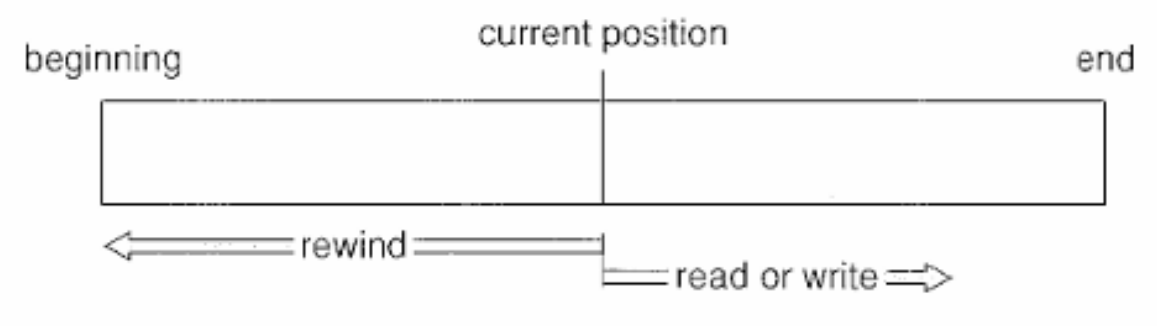
Sequential access method

- The simplest access method is a sequential access. This mode of access is by far the most common; for example, editors and compilers usually access files. Reads and writes make up the bulk of the operations on a file. A read operation-read next-reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation-write next-appends to the end of the file and advances to the end of the newly written material. Such a file can be reset to the beginning; and on some systems. Sequential access, which is depicted in figure, is based on a tape model of a file and works as well on sequential-access devices as it does on random-access ones.

Direct access method

- A file is made up of fixed-length logical access that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. As a sample, it is read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file. Direct-access files are of great use for immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, it computes which block contains the answer and then read that block directly to provide the desired information.

Q. What is the difference between sequential access and direct access method? (cont…)



- Sequential access file

Q. When designing a directory structure, what factors should be considered in relation to the operations performed on the directory?

When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:

Search for a file: We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names may indicate a relationship between files, users want to be able to find all files whose names match a particular pattern.

Create a file: New files need to be created and added to the directory.

Delete a file: When a file is no longer needed, users want to be able to remove it from the directory.

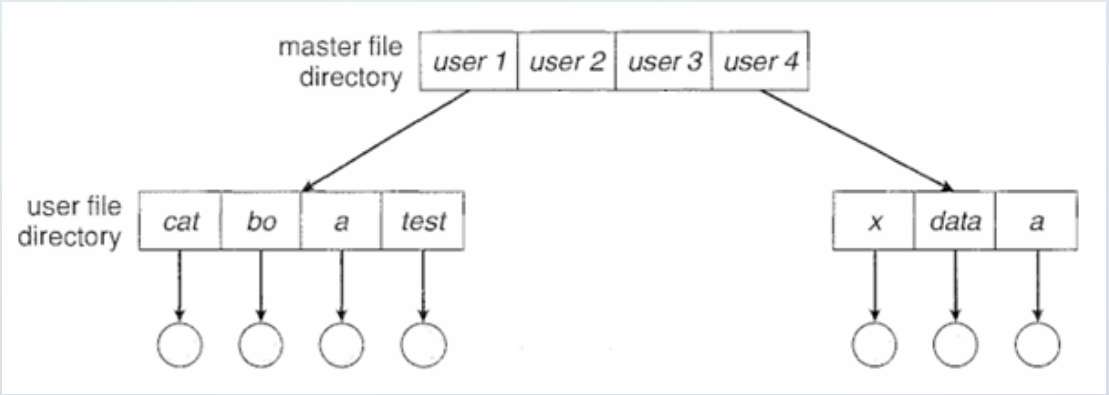
List a directory: users need to be able to list the files in a directory and the contents of the directory entry for each file in the list.

Rename a file: Because the name of a file represents its contents to its users, they must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.

Traverse the file system: Users wish to access every directory and every file within a directory structure.

Q. In a two-level directory structure, what operations or considerations are important when managing directories? To illustrate, *user1*, *user2*, *user3*, and *user4* are in master file directory. And *user1* has *cat*, *bo*, *a*, *test* and *user4* has *x*, *data*, *a* for user file directory,

In a two-level directory structure, when a user accesses a file, the operating system searches only that user's own User File Directory (UFD). This allows different users to have files with the same name, as long as file names within each UFD are unique. When creating a file, the OS checks only the user's UFD to ensure there is no duplicate name, and when deleting a file, it searches only that same UFD, preventing accidental deletion of another user's file. User directories are created and deleted as needed, usually by a special system program run by administrators, which also adds the new UFD to the Master File Directory (MFD). Disk space allocation for UFDs is handled in the same way as for normal files. This structure solves the name-collision problem and provides isolation between users, which is useful when they work independently. However, it makes collaboration difficult, since to access another user's file, both the user name and file name must be specified. Conceptually, it forms a tree of height two, with the MFD as the root, UFDs as the branches, and files as the leaves, where each file has a unique path name consisting of the user name and file name.



Q. What is a mount point in a file system?

In file system mounting, the operating system is given the device name and the location in the directory structure where the file system should be attached. Some systems require the file system type to be specified, while others automatically detect it by inspecting the device's structures. A mount point is usually an empty directory; for example, in UNIX, a file system with user home directories might be mounted at */home*, allowing access to */home/jane*.

Mounting the same file system at /users would make the same directory accessible as */users/jane*. The OS then verifies the device contains a valid file system by asking the device driver to read its directory and checking that it has the correct format. Once validated, the OS records in its directory structure that a file system is mounted at the mount point. This allows the OS to traverse directories seamlessly, switching between different file systems, even of different types, when needed.

References

Images: Internet

Source: Operating System Concepts (8th Edition)