UNIT-4
FILE MANAGEMENT

---

1. What is the use of file abstraction?
   File abstraction provides a uniform logical view of physical contents from a wide variety of storage devices that have different characteristics.

2. What all does a file have?
   a. *Name*: for identification.
   b. *Space*: to store data.
   c. *Location*: for convenient access and access restrictions.
   d. *Other Attributes*: for support mechanisms.

3. Define file management system.
   It is a *software layer* which the OS implements on top of the I/O system (device drivers) for users to access data with ease from storage devices.
   The file management system contains files, directories and control information (metadata) and supports convenient and secured access to files and directories.

4. List the functions of the file system.
   a. Organization of files
   b. Execution of file operations
   c. Synchronization of file operations
   d. Protection of file contents
   e. Management of space in the file system

5. What is a file?
   A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes, and optical disks, which normally represents programs and data.
   In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the file creator and user.

6. How is a file named?
   A symbolic file name given to a file, for convenience of users, is a string of characters. When a file is named, it is independent of the process, the user, and the system it created. It is the only information kept in human readable form.

7. Explain the file structures in brief.
   A file has a certain defined structure, which depends on its type of information stored in the file.
   a. *Text File*: a sequence of characters organized into lines.
   b. *Source File*: a sequence of procedures and functions.
   c. *Object File*: a sequence of bytes organized into blocks that are understandable by the machine.

8. Describe the various file attributes.

Attributes define the characteristics of a file, varying with respect to OS and useful for protection, security, and usage monitoring.

a. *Name*: only information kept in human readable form.
b. *Types*: needed by those systems that support different file types.
c. *Location*: pointer to a device, used to locate the file on the device.
d. *Size*: current file size (bytes, words, or blocks) and maximum file size are stored.
e. *Protection*: access control information – who can read, write, and execute.
f. *Time, Date and User Identification*: creation, last modification, and last use.

9. Describe the logical file structure.
   There are 3 possible forms of atomic units in a file:
   a. *Bytes*: File is a sequence of bytes called as flat file (no internal structure).
   b. *Fixed Length Record*: Files are divided into fixed length records. Each record is a collection of information about one thing. Easy to deal with but does not depict the realities of data.
   c. *Variable Length Records*: Used to meet the various workload lengths. The problem is, unless the location of the file is known, it is hard to perform search operation. To solve this problem, *keyed file* is used. Each record has a specified field, which is the key field, used for finding the location of the file.

10. What is file meta data?
    A file contains information. But the system also keeps information about the file, known as metadata – information about information, such as name, type, size, owner, groups of users with special access, access rights, last written time, time of creation, storage disk location, etc.

11. What are the various file types?
    File Type refers to the ability of the operating system to distinguish different types of files. MS-DOS and UNIX have the following file types:
    a. *Ordinary Files*: They contain user information as text, database, or executable program. Add, modify, delete, remove operations can be performed.
    b. *Directory Files*: They contain list of file names and other information related to files.
    c. *Special Files (or) Device Files*: They represent physical devices like disks, terminals, printers, networks, tape drive, etc. They are of two types:
       i. *Character Special Files*: Data is handled character by character, as in terminals and printers.
       ii. *Block Special Files*: Data is handled in blocks, as in disks and tapes.

12. List down the parts of a file system.
    The file system consists of two distinct parts:
    a. *Collection of Files*: each storing related data.
    b. *Directory Structure*: Organizes and provides information about all files in the system.

13. Explain the various file operations.
    The various file operations include: 1) create 2) delete 3) open 4) close 5) read 6) write 7) truncate 8) memory map 9) file pointer 10) reposition 11) seek.

    a. **Create**

i. Essential for system to add files.
ii. Can be merged with open.
iii. Requires filename, directory name and file attributes.
iv. Space in the system must be found.
v. Entry for the new file must be made in the directory.

b. **Delete**
   i. Essential to delete files.
   ii. Requires filename. Directory is searched, once found, all file spaces are released, and the directory entry is released.

c. **Open**
   i. Not essential.
   ii. Optimization: Translation from filename to disk location is performed only once per file, rather than once per access.
   iii. Requires a filename to be opened – file system checks for permission to access, if yes, then it creates a file descriptor that will be used by the application for future reference.

d. **Close**
   i. Not essential.
   ii. Frees resources.
   iii. Requires a file descriptor that was obtained in the open operation.
   iv. Releases the file descriptor and other related resources allocated for the file open session.

e. **Read**
   i. Essential.
   ii. Must specify filename, file location, number of bytes and buffer into which data must be placed.
   iii. Read operation takes file descriptor, a positive integer and buffer address as parameters.
   iv. This operation copies into the buffer, those many numbers of consecutive bytes, starting at the current file pointer position from the file. It repositions the file pointer past the last byte it read.

f. **Write**
   i. Essential if updates are to be supported.
   ii. Write operation takes file descriptor, a byte string and the size of the string as parameters.
   iii. It overwrites file content starting at the current file pointer position within the file. It allocates more space to the file when it needs to write past the last byte in the file. It repositions the file pointer after the last byte is written.

g. **Truncate**
   i. Attributes remain the same, but file content is erased (length reset to 0).
   ii. Truncate operation takes file descriptor and a positive integer as parameters. It reduces the size of the corresponding file to the specified number.

h. **Memory Map**
    i. Creates a region in the process address space, each byte in the region corresponds to a byte in the file.
    ii. Conventional memory read and write operations on the mapped sections by applications are treated by the system as file read and write operations respectively.
    iii. When the mapped file is closed, all the modified data are written back to the file and the file is unmapped from the process address space.

i. **File Pointer**
    i. On systems that do not include file offset as a part of the read and write system calls, the system must track the last read / write location as the current file position pointer.
    ii. Pointer is unique to each process operating on the file, and therefore must be kept separate from the disk file attributes.

j. **Reposition**
    i. Adjusts the file pointer to a new offset.
    ii. Reposition operation takes file descriptor and offset as parameters and sets the associated pointer to the offset.

k. **Seek**
    i. Not essential.
    ii. Seek operation specifies the offset of the next read or write access to the file.

14. What are directories?
    A directory is a name space. The associated name maps each name into either a file or another directory. Directories contain bookkeeping information about files. Directories are generally implemented as files and in the OS point of view, there is no distinction between files and directories. Files in OS are generally named using a hierarchical naming system based on directories.

15. List down some directory operations.
    a. **Create**: Produces an "empty" directory. Normally the directory contains "." and ".." so it is not actually empty.
    b. **Delete**: Directory must be "empty" before doing this operation (contains only "." and ".."). Commands are written that normally empty the directory and then use file and directory delete system calls to perform deletion.
    c. **Opendir**: Same as in files.
    d. **Closedir**: Same as in files.
    e. **Readdir**: Same as in files.
    f. **Rename**: Same as in files.
    g. **Link**: Add a second name for a file.
    h. **Unlink**: Removes a directory entry. But if there are many links and only one is unlinked, the file remains.

16. Explain the various directory structures.

a. **Single-Level Directory**
    i. All files are placed in one directory.
    ii. This is very common on single-user systems.
    iii. Unique names are a problem when there are a greater number of files and more than one user.
b. **Two-Level Directory**
    i. The system maintains a master block that has one entry for each user. It contains addresses of each user's directory.
    ii. This structure isolates one user from another. This is an advantage when each user is completely independent, but disadvantageous when users want to cooperate on some task and access files of other users.
c. **Tree-Structured Directory**
    i. The directories are themselves files each containing subdirectories that contain files and sub-subdirectories.
    ii. How to delete a directory? In general, an empty directory is deleted. But a directory with files and subdirectories will not be deleted until it is emptied.
d. **Acyclic Directory**
    i. Extension of the tree-structured directory.
    ii. One file here can be owned by several users.

17. How is a pathname kept?
    Absolute Pathname – Gives the path from the root directory to the file that is named.
    Relative Pathname – Gives the path from the current working directory to the file.

18. Explain the concept of aliases.
    Generally, all files in a directory are related. But sometimes, a file may be a part of two or more groups. Here, file aliases come to the rescue. A file alias is a filename that refers to a file that also has another name. This file has two absolute pathnames.

19. Explain the various file access mechanisms.
    File access mechanism refers to the manner in which records of a file may be accessed.
    a. **Sequential access**
        i. Records are accessed in a sequence, information in the file is processed in order, one record after the other.
        ii. It is the most primitive form of access.
        iii. Compilers usually access files in this fashion.
    b. **Direct / Random access**
        i. Records are accessed directly.
        ii. Each record has its own address on the file, with the help of which it can be directly accessed for reading and writing.
        iii. Records need not be adjacent or in any sequence.
    c. **Indexed Sequential access**
        i. It is based on sequential access.
        ii. An index is created for each file, which contains pointers to various blocks.
        iii. Index is searched sequentially and the pointer is used to access the file directly.

---

1. Describe the structure of an open file.
    a. Current file position
    b. Other status information about the open file (e.g., is the file locked or not)
    c. Pointer to the file description that is open.

    The open file table contains a structure for each open file.

2. Describe the structure of files which exist on disk.
    a. File descriptor: contains all metadata about the file.
    b. File data: kept in disk blocks.

3. Describe the contents of the file descriptor data structure.
    a. Owner of the file
    b. File protection information
    c. Time of creation, last modification, and use
    d. Other file meta data
    e. Location of the file data

4. How are directories implemented?
    A directory is a table that maps component names to file descriptors.

5. How is space allocated in OS?
    a. **Contiguous Allocation**
        i. Each file occupies a contiguous address space on the disk.
        ii. If the file to be created is n blocks long, n free contiguous blocks must be located.
        iii. First-fit, best-fit and worst-fit strategies are used to select a free hole from the set of available holes.
        iv. External fragmentation and the amount of disk space is a problem.
    b. **Linked Allocation**
        i. Each file carries a list of links to disk blocks.
        ii. Directory contains link / pointer to the first block of a file.
        iii. Write to a file removes the first free block and writes to that block.
        iv. To read a file, pointers are just moved from block to block.
        v. No external fragmentation.
        vi. Effective in sequential access file. Insufficient in case of direct access file.
    c. **Indexed Allocation**
        i. Provides solutions to all problems faced in contiguous and linked allocation.
        ii. An index block is created, having pointers to all files.
        iii. Each file has its own index block, which stores the addresses of the disk space occupied by the file, which is an array of disk sector of addresses.
        iv. The ith entry in the index block points to the ith sector in the file.
        v. It supports direct access without external fragmentation.

6. What is a free-space list?
    To keep track of free disk space, the system maintains a free space list. To create a file, the free-space list is searched for the required amount of space and allocates it to a new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

7. How are free-space lists implemented?
    a. **Bit-Vector**
        i. Each block is represented by one bit – 0 if it is free, 1 if it is allotted.
        ii. It is simple and efficient to find n consecutive free spaces on the disk.
        iii. But bit vectors are inefficient unless the entire vector is kept in memory for most accesses.
    b. **Linked List**
        i. All free disk blocks are linked together, keeping a pointer to the first free block.
        ii. This scheme is not efficient, to traverse the list, each block must be read, which requires substantial I/O time.
    c. **Grouping**
        i. Addresses of the first 'n' free blocks are stored in the first free block.
        ii. The first 'n-1' of these are actually free. The last one is the disk address of another block containing addresses of another 'n' free blocks.
        iii. Addresses of a large number of free blocks can be found quickly.
    d. **Counting**
        i. Address of the first free block is kept, and the number 'n' of free contiguous blocks that follow the first block are kept.
        ii. Each entry in the free-space list then consists of a disk address and a count.
