**Unit 5**

**File Management**

A **file** is a named collection of related information that is recorded on secondary storage. Commonly file represents program both source and object forms and data. Data files may be numeric, alphabetic, alphanumeric or binary. Files may be of free form such as text file or may be formatted rigidly. In general, file is a sequence of bit, bytes, lines or record the meaning of which is defined by the file’s creator and users. The file system consist of two distinct part: a collection of file each storing related data and a directory structure which organizes and provides information about all the files in the system.

A file has a certain defined structure which depends on its type and information in a file is defined by its creator. A **text file** is a sequence of characters organized into lines and possibly pages. A **source file** is a sequence of function each of which is further organized as declarations followed by executable statement. An **executable file** is a series of code section that the loader can bring into memory and execute.

**File Attributes:**

A file is names for the convenience of human user and is referred by its name. A name is usually a string of character followed by an extension. For example student.doc. Here, student is the file name and doc is a file’s extension. When a file is named it becomes independent of the process, the user and even the system that created it. The system uses the extension to indicate the type of the file and the type of the operation that can be done on that file. A file’s attributes vary from one operating system to another but typically consists of following terms:

* **Name:** it is the information kept in human readable form for the convenience of the user. It may be sequence of character or alphanumeric values.
* **Identifier:** this unique tag usually a number uniquely identifies the file within the file system. It is the non-human readable name for the file.
* **Type:** this information is needed for the system that supports 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 which determines who can do reading, writing, executing etc.
* **Time, date and user identification:** this information may be kept for recording creation of file, last modification and last use. These data can be useful for protection, security and usage monitoring.

The information about all files is kept in the directory structure which is also resides in secondary memory. A directory entry consist of the file’s name and its unique identifier and identifier in turn locates the other file attributes.

Some typical file extension are:

|  |  |
| --- | --- |
| **Extension** | **Meaning** |
| .bak | Backup file |
| .bas | Basic source program |
| .bin | Executable binary program |
| .c | C source program |
| .dat | Data file |
| .doc | Document file |
| .hlp | Text for help command |
| .obj | Object file (output of compiler) |
| .txt | General text file |

**File Operations:**

The operating system can provide system calls to create, write, read, reposition, delete and truncate files. Following are the operation that can be performed in file:

* **Creating a file:**

Its purpose is to create a blank file in which two steps are required. 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:**

Its purpose is to write some data into file. To do this, system call is made to specify both the name of the file and the information to be written into file. Given the name of the file, the system searches the directory to find the file’s location. The system must keep the write pointer to the location in the file where the write is to take place. The write pointer must be updated whenever a write occur.

* **Reading a file:**

Its purpose is to read some data from a file. To do this, system call is made to specify the name of the file and where in memory the next block of the file should be put. The directory is searched for the entry and the system needs to keep a read pointer to the location in the file where the next read is to take place. The read pointer must be updated once the read has taken palace.

* **Repositioning within the file:**

The directory is searched for the appropriate entry and the current file position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek.

* **Deleting a file:**

The purpose of this system call is to delete a file. To do this, directory is searched for the named file. If the directory is found all the file spaces are releases so that it can be reused by other files and erase the directory entry.

* **Truncating a file:**

Its purpose is to erase the content of the file but keeping its attributes unchanged. The file is reset to length zero and its file space is released.

* **Appending a file:**

Its purpose is to add some data to the end of the existing file.

* **Renaming a file:**

Its purpose is to put a new file’s name from current name.

Most of the file operation mentioned above requires searching a directory for the entry associated with the named file. To avoid this constant searching many system requires that an open() system call be made before a file is first used. The operating system keeps a table called the open file table which contains the information about all the open file. When a file operation is requested the file is specified via an index into this table so no searching is required. When a file is no longer being actively used it is closed by the process and the OS removes its entry from the open file table.

**File Access Method:**

When a file is used, the information it contained must be accessed and read into computer memory. The information in the file can be accessed in several ways. Some system provide only one access method for files while other may support many access method. Following are the common file access methods:

* **Sequential Access:**

In this access method, information in the file is processed in order, one record after the other. This mode of access is a common access method. For example editors and compilers usually access file in this fashion. Data records are retrieve in the same order in which they have been stored in disk.

Read 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 causes a pointer to move ahead by one. Similarly, the write operation write\_next() appends to the end of the file and advances to the end of newly written material i.e. the new end of a file.

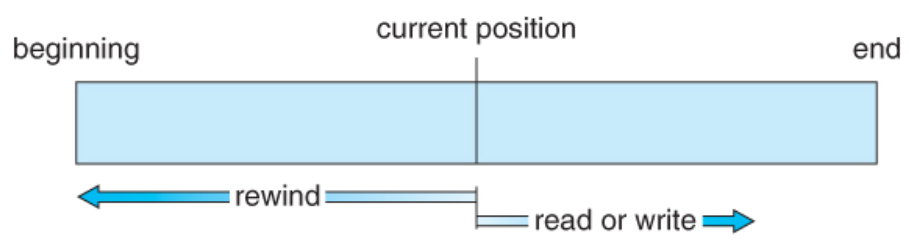


Figure: Sequential access file

* **Direct Access:**

Direct access is also known as relative access in which a file is made up of fixed length logical record that allow programs to read and write records rapidly in no particular order. This method is based on the disk model of a file since disk allow random access to any file block. Here, the file is viewed as a numbered sequence of blocks or records. There are no restriction on the order of reading or writing for a direct access i.e. file can be read for block 14, then block 54 and then write 57. User say read “n” rather than read next.

Direct access files are of great use for immediate access to large amount of information. For the direct access the file operation are modified to include a block number as a parameter. That is read\_next() is modified as read(n) and write\_next() is modified as write(n) where n is the block number. The block number provided by the user is a relative block number.

A relative block number is the index relative to the beginning of the file. That is the first relative block of the file is 0, the next is 1 and so on even though absolute address may be different from relative such as for relative address 0 absolute address can be 14703. The use of the relative block number allows the operating system to decide where the file should be placed and helps to prevent the user from accessing portion of the file system that may not be the part of his/her file. The techniques used to calculate or obtain absolute address is hashing in which every record is associated with key number to preprocess the address calculation.

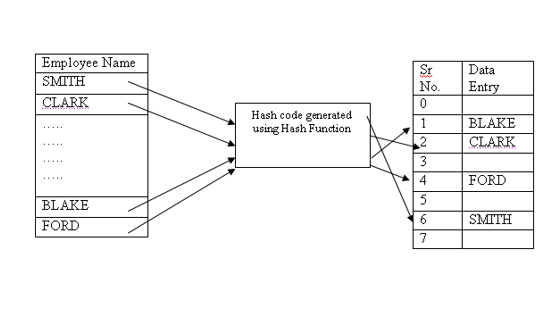


Figure: random access

* **Index Access:**

This method generally involves the construction of an index for the file. The index contains pointers to the various blocks. To find a record in the file, first the search is made for the index and then use the pointer to access the file directly and to find the desired record. While querying a data the index key is kept in the memory and related records are fetched from the disk.

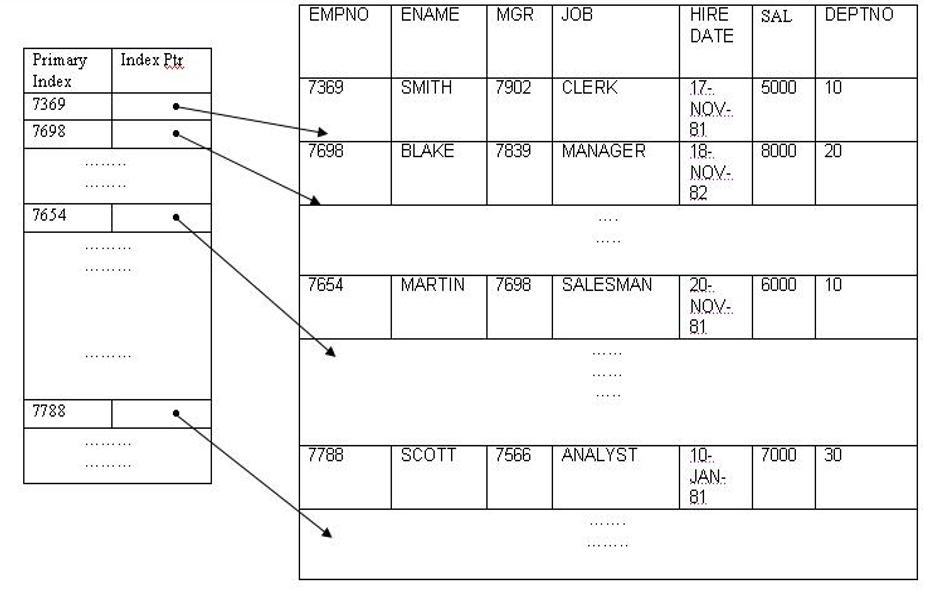


Figure: index access

With large file the index file itself may become too large to be kept in memory. Solution to it is to create an index for the index file. The primary index fie contains pointer to secondary index files which point to the actual data items.

**Directory and Disk Structure:**

There may be a thousand, millions of file within a computer. Files are stored on random access storage device including hard disk, optical disk and solid state disks. A whole storage device can be used for a file system or it can also be divided into fine grained control. For e.g. disk can be partitioned into quarters and each quarter can hold a separate file system.

Partitioning is done for limiting the size of individual file system, putting multiple file-system types on the same device or leaving part of the device available for other uses such as swap space or unformatted (raw) disk space. A file system can be created in each of this partition and such partition containing a file system is generally known as a **volume**. The volume may be the subset of a device, a whole device or multiple device linked together.

Each volume that contains a file system must also contains information about the files in the system. This information is kept in entries in a device directory (directory) or volume table of content. Directory records the information such as name, location, size and type for all files on that volume.

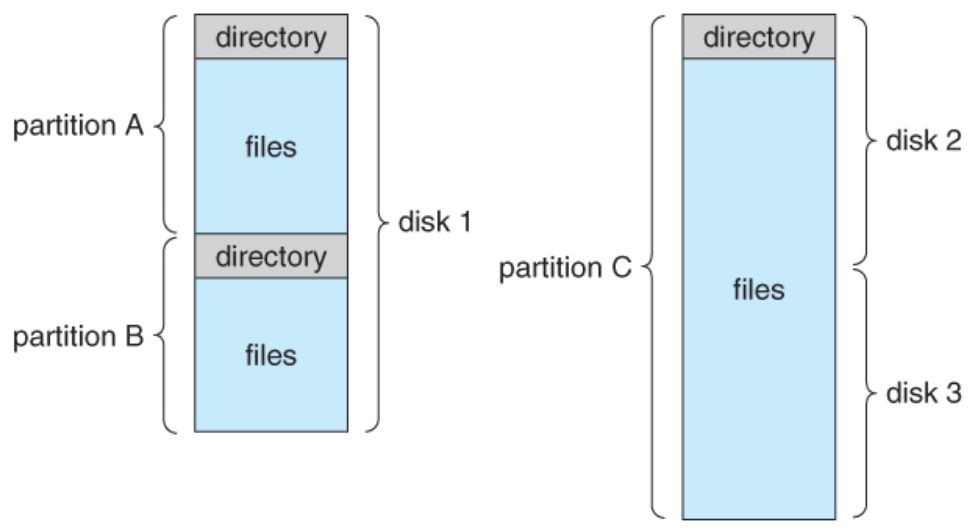


Figure: file system organization

**Directory:**

A **directory** is a location for storing a file on a computer i.e. a container that is used to contain folder and file. It organizes the file and folder into hierarchical manner. The directory can be organized by itself and must allow to insert entries, to delete entries, to search for a named entry and to list all the entries in the directory. The operation that can be performed in directory are:

* Search for a file: to be able to search a file, first directory structure should be search to find an entry for a particular file.
* Create a file: need file need to be created and added to the directory.
* Delete a file: when a file is no longer needed, file should be able to be deleted from a directory.
* List a directory: files are needed to be listed in a directory and the content of the directory entry for each file in the list.
* Rename a file: as a name of the file represents its contents to its users, the name of the file should be able to change when the content 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: sometimes it is needed to access every directory and every file within a directory structure. It is good idea to save the content and structure of the entire file system at regular interval. Files are copied to magnetic tape because of which backup copy is provided in case of system failure.

**Structure of Directory:**

1. Single level directory:

It is a simplest directory structure in which all files are contained in the same directory which is easy to support and understand. Since all the files are in same directory they must have a unique name. If the two user put their data file name same for e.g. test.doc then the unique name rule is violated. But most file system supports file name up to 255 character so it is relatively easy to select unique file name.

**Advantages:**

* Its implementation is very easy as it contains single directory
* If files are smaller in size then searching will be faster.
* The operation like creation, deletion, updating are very easy in such a directory structure.

**Disadvantages:**

* There may be a chance of name collision.
* As the number of files increases then it will be difficult to remember the name of the files.
* Searching will became time consuming if directory will be large.

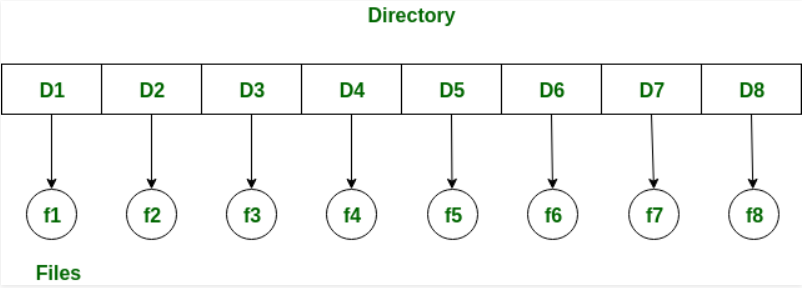


Figure: Single Level Directory

1. **Two level Directory:**

In this structure, separate directory is created for each user. In this structure, each user has its own user file directory (UFD). The UFDs have similar structure but each lists only the files of a single user. When a user job starts or a user logs in the system’s master file directory (MFD) is searched. The MFD is indexed by user name or account number and each entry points to the UFD for that user.

When a user refer to a particular file only his own UPD is searched. Thus different user may have same file name in different directory but the file name should be unique within one particular directory. For e.g. the file name called std.doc must be unique in user’s directory no. 1. To create a file for a user the operating system searches only that user’s UFD to check whether another file of that name exist or not. To delete a file, the OS confines its search to the local UFD thus it cannot accidently delete another user’s file that has the same name.

To search a file of other user’s directory a path name (which is composed of user name and file name) should be given. For example if a user A wishes to access the file name text.doc of user B with directory entry name of user b. then he or she might have to refer as /userb/test.doc. This access method or syntax are based on types of system i.e. every system has its own syntax for naming files in directory.

In windows access can be done by: C:\userb\text.doc\.

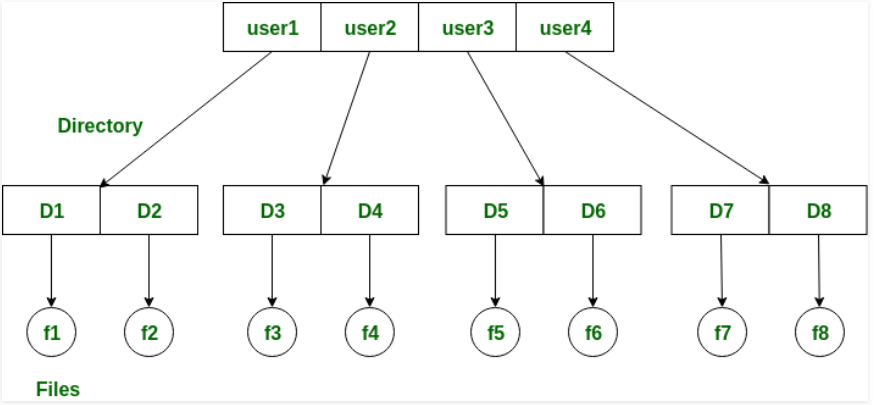


Figure: Two-level directory structure

**Advantages:**

* Different user can have same directory as well as file name
* Searching of a file becomes more easy due to path name and user-grouping.

**Disadvantages:**

* This structure isolates one user from another which will create a problem when one user want to access the file of another user.

1. **Tree structure Directory:**

It is an extended of the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly. The tree has a root directory and every file in a system has a unique path name. A directory contains a set of files or subdirectories and all directories have same internal format. Special system calls are used to create and delete directories.

In normal use, each process have a current directory and current directory should contains most of the files that are of current interest to the process. When a reference is made to a file the current directory is searched. If a needed file is not in the current directory then the user must specify its path name or change the current directory to the directory holding that file.

For the deletion of the directory, if a directory is empty then it can simply be deleted. If the directory to be deleted is not empty then one of the two approach can be taken. Some system will not delete a directory until it is empty. Thus to delete a directory, user must first delete all the subdirectories and files in the directory. Next approach is to provide an option: when a request is made to delete a directory, all that directory’s files and subdirectories are also to be deleted or not.

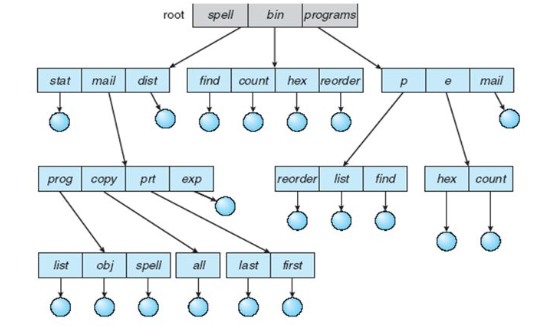


Figure: Tree-Structured Directories

1. **Acyclic graph Directories:**

An acyclic graph is a graph with no cycle which allows directory to share subdirectories and files. The same file or subdirectories may be in two different directories. The acyclic graph is a natural generalization of the tree structured directory scheme. With a shared file only one actual file exists so any changes made by one person are immediately visible to the other. Sharing is particularly important for the subdirectories; a new file created by one person will automatically appear in all the shared subdirectories.

When working in a team, all the file that needs to be share can be put into one directory. The UFD of each team member will contains the directory of shared file as subdirectory. Shared files and subdirectories can be implemented in several ways.

Shared file and subdirectories can be implemented in several ways. A common way is to create a new directory entry called a link. A link is a pointer to another file or subdirectory and can be implemented as an absolute or a relative path name. When a reference to a file is made, first a directory is searched and if a directory entry is marked as a link then the name of the real file is included in the link information.

Another common approach to implement shared file is simply to duplicate all information about them in both sharing directories. Thus, both entries are identical and equal. A major problem associated with it is maintaining consistency when a file is modified.

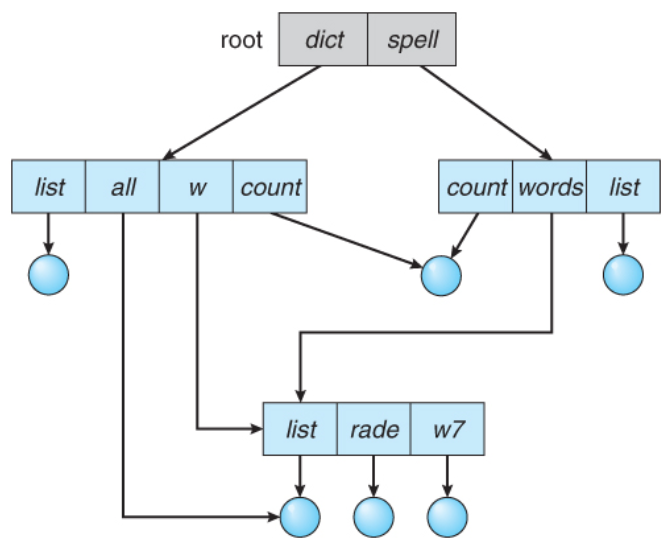
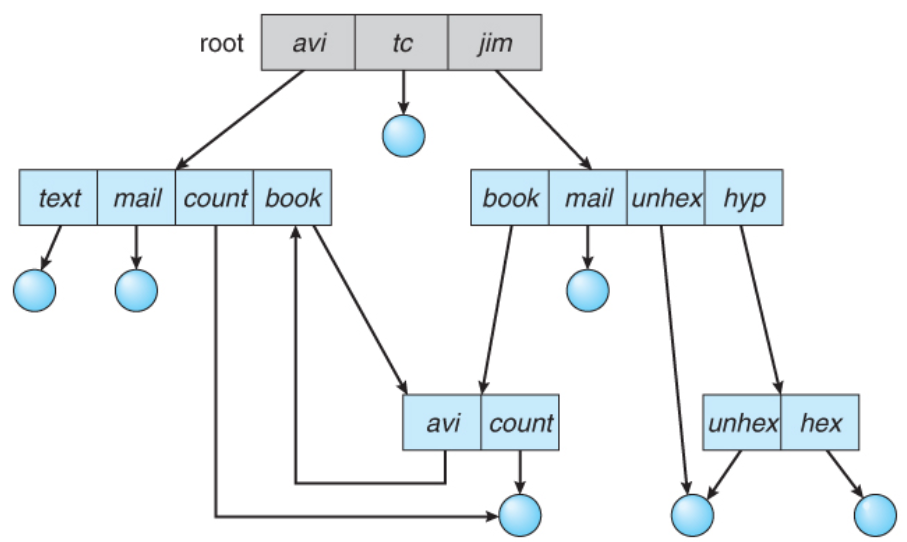


Figure: Acyclic graph directories

1. **General graph Directory:**

In this structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory. If we starts with two level directory and allow users to create subdirectories a tree structure directory will results. When a link is added a tree structure is destroyed resulting in a simple graph structure. If a cycle are allowed then the searching for more than one time is avoided for any component.



There are two types of path name:

* **Relative path name:** it defines the path from current directory. Instead of being specified from the root directory are specified relative to working directory. For example: if the current working directory is user/current/ then the file whose absolute path is “user/current/student can be refer as “student”.
* **Absolute path name**: this path name begins at the root and follows a path down to the specified file, giving a directory name on the path. It is a listing of the directories and files from the root directory to the intendent file. For example: “C:/window/user/sys.exe” means that the root directory C contains a sub-directory window which in turns contains a directory user that contains an executable “sys.exe”.

**File System Mounting:**

**Mounting** is a process by which operating system makes files and directories on a storage device (such as hard drive, CD-ROM etc.) available for users to access via the computers file system. **Unmounting** is a process in which the operating system cuts off all the user access to file and directories on the mount file, making the storage device safe for removal. A file system must be mounted before it can be available to processes on the system. The directory structure may be built out of multiple volumes which must be mounted to make them available within the file system name space.

For mounting: an operating system is given the name of the device and the mount point (location within the file structure where the file system is to be attached or it is a name which can be used to refer to the disk. For example “C :”). Some operating system require the type of the system to be provided while other inspect the structure of the device and determine the type of the file system. For example: in windows OS mounting the file system under the path to a specific file takes the form of drive-letter:\path name \to\file i.e. “C:/windows/user/student”.

Next the operating verifies that the device contains a valid file system. It does so by asking the device driver to read the device directory and verifying that the directory has the expected format. Finally the operating system notes in its directory structure that a file system is mounted at the specified mount point.

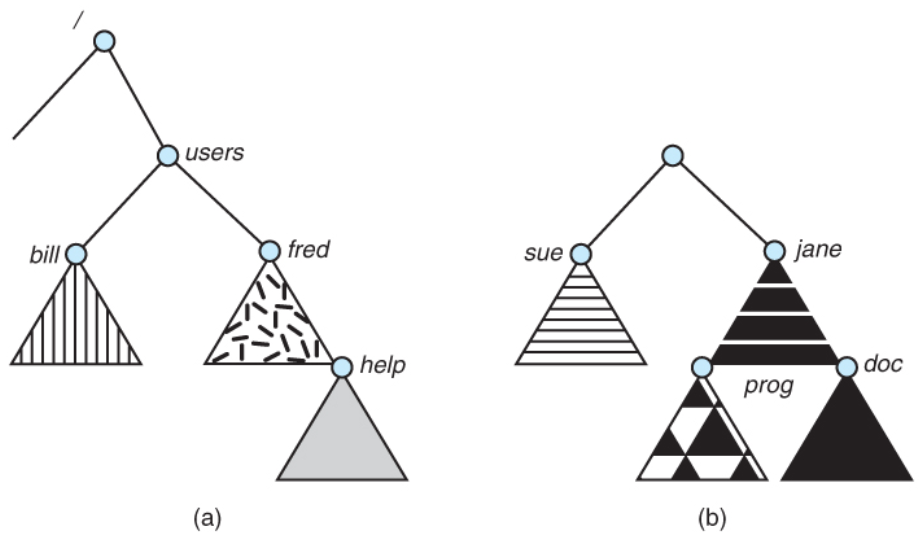


Figure: a) Existing system b) Unmount volume

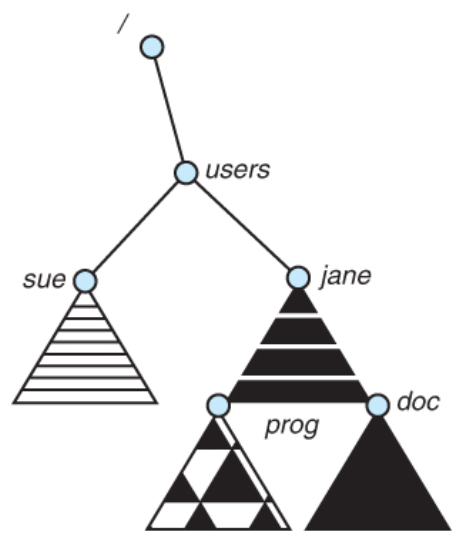


Figure c) Mount point

In the above figure, triangles represents the subtree of directories. Figure a) shows the existing file system whereas figure b) shows the unmounted volume resides on /device/disk. At this point only the files on the existing file system can be accessed.

Figure c) shows the effect of the mounting the volume residing on /device/disk/users. If the volume is unmounted the file system is restored to the situation depicted in figure b.

**File Sharing:**

Here, the general issues and solution related to sharing a file are discussed:

1. **Multiple Users:**

When an operating system accommodates multiple users the issues of the file sharing, file naming and file protection becomes important. Given a directory structure that allows file to be shared by users, the system must implement file sharing. The system can either allows a user to access the files of other users by default or require that a user specifically grant access to the files.

To implement the sharing and protection, most system have evolved to use the concept of file or directory owner and group. The owner is the user who can change attributes, grant access and who has the most control over the file. The group attribute defines a subset of users who can share access to file. Out of all operation group member can execute only a subset of such operation and exactly which operation can be executed is defined by file’s owner.

The owner and the group id of the given file are stored within file attributes. When a user request some operation on the file then the user id or group id is checked with owner attributes to determine if the requesting user is the owner of the file or not. The result indicates which permission are applicable. The system then applies those permissions to the requested operation and allows or denies it.

1. **Remote file System:**

Networking allows the sharing of resources spread across around the world. One obvious resource to share is data in the form of files. The first implemented method involves manually transferring files between machines via program like ftp. The second major method used a distributed file system (DFS) in which remote directories are visible from a local machine. The third method if World Wide Web in which browser is needed to gain access to the remote files and separate operation (like ftp) are used to transfer a file. Ftp if used for both anonymous and authenticated access. Anonymous access allows a user to transfer a files without having an account on the remote system whereas authenticated access allows a user to transfer a file if user is authenticated to the system.

Remote file system are used in various model. Some of them are:

* **Client Server Model:**

The remote file system allows a computer to mount one or more file system from one or more remote machines. Here, the machine containing the file is server and the machine seeking access to the files is the client. The server declares that a resource is available to clients, specifies exactly which resource (file) and exactly which clients. Such available resource are specifies on a volume or directory level. Client identification is more difficult and can be specified by the network address such as IP address but these can be spoofed. Some cryptographic authentication mechanism can be used but is very difficult to maintain.

Once the remote file system is mounted, file operation requests are send on behalf of user which includes file-open request with ID of the user across network to a server via DFS protocol. The server then checks the user’s credentials to determine if the user can access the file in requested mode or not. If it is allowed then file handle is returned to the client application and the application can then perform read, write and other operation on the file. The client closes the file when access is completed.

* **Distributed Information System:**

To make client server systems, easier to manage distributed information systems (distributed naming services) provides unified access to the information needed for remote computing. The domain name system (DNS) provides host name to network address translation for the entire internet. Other distributed information system provides user name/password/user ID/ group ID space for a distributed facility.

In case of Microsoft’s common internet file system (CIFS), network information is used in conjunction with user authentication (user name and password) to create a network login that the server uses to decide whether to allow or deny access to a requested file system. For this authentication should be valid, the username must match from machine to machine.

* **Failure Mode:**

Remote file system have more failure than local file system because of the complexity of the network system and the required interaction between remote machine, many more problem can interfere with the proper operation of remote file system. The network can be interrupted between two hosts which can be resulted from the hardware failure, poor hardware configuration or network implementation issues. Any single failure can thus interrupts the flow of DFS command.

To implement a recovery from a failure some kind of state information may be maintained on both the client and server. If the client and server both maintain the knowledge of their current activities and open files then they can recover from the failure.

**File Protection:**

When the information is stored in the computer system it should be keep safe physical damage and improper access. File system can be damage by the hardware problem such as error in reading or writing, power supply failure, temperature extreme etc. This problem can be addressed by having a backup of all the file system on another secondary disk. Many system have a system program that automatically copy disk file into tape at regular interval to maintain a copy of original file.

Protection can be provided in several ways. For single user system protection might be provided by locking the computer using password. In a larger multiuser system other mechanism are needed which are described below:

1. **Type of Access:**

It is the process of imposing some control on the file or imposing some restriction on mode of access in file. Thus, complete protection can be provided by prohibiting access. Protection mechanism provide controlled access by limiting the type of file access that can be made. Access is permitted or denied depending upon the type of access requested. Different types of operations may be controlled such as: read, write, execute, append, delete, list, copying, editing etc.

1. **Access Control:**

In this process, protection is enforced depending on the identity of the user i.e. access to file is dependent on the identity of the user. The most general scheme to implements identity dependent access is to associate with each file and directory an access control list (ACL) specifying the user name and types of access allowed for each user.

When the user requests access to a particular file the operating system checks the access control list and if that user is listed for the requested access the access is allowed. Otherwise the user job is denied access to file.

This method have two undesirable consequences:

* If the list of the user in the system is not known in advance then constructing such list may be tedious and time consuming.
* The directory structure previously of fixed size must be of variable size resulting in more complicated space management.

To solve this problem many system recognize three classification of users in connection with each file:

* Owner: the user who created the file.
* Group: a set of users who are sharing the file and need similar access is a group
* Universe: all other users in the system constitute the universe

The common approach is to combine access control list with the more general owner, group and universe and allowing the access based on the control information.

1. **Other Protection Approaches:**

Another approach to maintain protection is to associate a password with each file. If the password are chosen randomly and changed often, this scheme may be effective in limiting access to a file. The use of the password have several disadvantages such as the number of password that the user set might be longer and difficult to remember which makes them impracticable and if only one password is used and if discovered all files are accessible. To solve this problem some system allows a user to associate a password with subdirectories rather than in individual file.

**File System Structure:**

File system provides an efficient and convenient access to the disk by allowing data to be stored, located and retrieved easily. A file system poses two kind of design problem. The first problem is defining how the file system should look to user which involves defining a file and its attributes, the operation allowed in a file and the directory structure for organizing files. The second problem is creating algorithm and data structure to map the logical file system onto the physical secondary storage device.

The file system is generally composed of many different levels and each level in the design uses the features of lower levels to create new features for use by higher level. The structure of file system is shown in figure below:

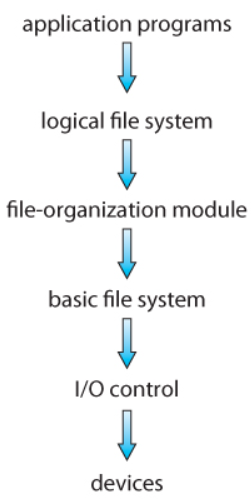


Figure: layered file system

* At the lowest layer are the **physical devices** consisting of the magnetic media, motors and control and electronics connected to them and controlling to them.
* The **I/O control level** consists of device driver and interrupt handlers to transfer information between the main memory and the disk system. A device driver can be thought of as a translator which input consists of high level command such as retrieve block 123 and output consists of low level hardware specific instruction which are used by hardware controller which interface the I/O device to the rest of the system.
* The **basic file system** issues generic command to the appropriate device driver to read and write physical blocks on the disk. It works directly with the device driver in term of retrieving and storing raw block of data without any consideration of what is in that block. This layer also manages memory buffer and caches that hold various file system, directory and data block.
* The **file system organization module** knows about files and their logical blocks, as well as the physical blocks. By knowing the types of file allocation used and the location of the file, the file organization module can translate logical block addresses into physical block addresses for basic file system to transfer. It also includes free space manger which tracks unallocated block and provides these space when requested.
* The **logical file system** manages the metadata information which includes all of the file system structure except the actual data. It also manages the directory structure to provide the file organization module with the information the latter needs given a symbolic file name. It maintains a file structure via file control block (FCB) which contains the information about the file including ownership, permission, and location of the file contents.

**Advantages:**

* Duplication of code is minimized
* I/O control and basic file system code can be used by multiple file system. Each file system can then have its own logical file system and file organization modules.

**Disadvantages:**

* Layering can introduce more operating system overhead
* Decision should include how many layer to use and what each layer should do is major challenges in designing new systems.

**File system implementation:**

Several on-disk and in-memory structures are used to implements a file system. These structure vary depending on the operating system and the file system.

On-disk structure:

Here, the file system may contain information about how to boot OS stored there, the total number of blocks, the number and location of free blocks, the directory structure and individual files. This structure includes:

* A **boot control block** per volume contains the information needed by the system to boot an operating system from that volume. If the block does not contains OS, then the block can be empty. It is the first block of the volume. In UFS, it is called the boot block and in NTFS it is called partition boot sector.
* A **volume control block** contains volume or partition details such as the number of block in partition, the size of the block, a free block of a volume. In UFS this is called a superblock and in NTFS it is stored in the master file table.
* A **directory structure** (per file system) is used to organize the files. In UFS this includes file names and associated inode number and in NTFS it is stored in the master file table.
* A **per file FCB** contains many details about the file. It has a unique identifier number to allow association with a directory entry.

**In-Memory Structure:**

It is used for both file system management and performance improvement via caching. The data are loaded at mount time, updating during file system operation and discarded at dismount. It includes:

* An **in-memory mount table** contains information about each mounted volume.
* An **in-memory directory** structure cache holds the directory information of recently accessed directories.
* The **system-wide-open file** table contains a copy of FCB of each open file as well as other information.
* The **per-process-open-file** table contains a pointer to the appropriate entry in the system wide open file table as well as other information.
* **Buffers** hold file system blocks when they are being read form disk or written to disk.

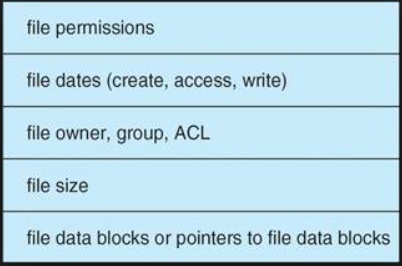


Figure: a typical file control block

To create a new file an application program calls the logical file system which knows the format of the directory structure and allocates a new FCB. The system then reads appropriate directory into memory, update it with new file name and FCB and write it back to the disk. As the file has been created it can be used for I/O.

**For I/O operation:**

For this the file should be opened which is done by **open ()** system call that passes a file name to the logical file system. The open () system call first searches the system-wide-open file table to see if the file is already in use by another process. If it is already created then per-process open file table entry is created pointing to the existing system-wide-open file table. If the file is not opened the directory structure is searched for the given file name. Once the file is found the FCB is copied into a system-wide-open- file table in memory. This table also keeps tracks of the number of processes that have the file open.

Next, an entry is made in the per process open-file table with a pointer to the entry in the system-wide open file table and some other field. The other field may include pointer to the current location in the file for **read () or write ()** operation and access mode in which the file is open. The open () call returns a pointer to the appropriate entry in the per-process file system table and all the operation are then performed via this pointer. When a process closes the file, the per-process table entry is removed and the system-wide entry open count is decremented.

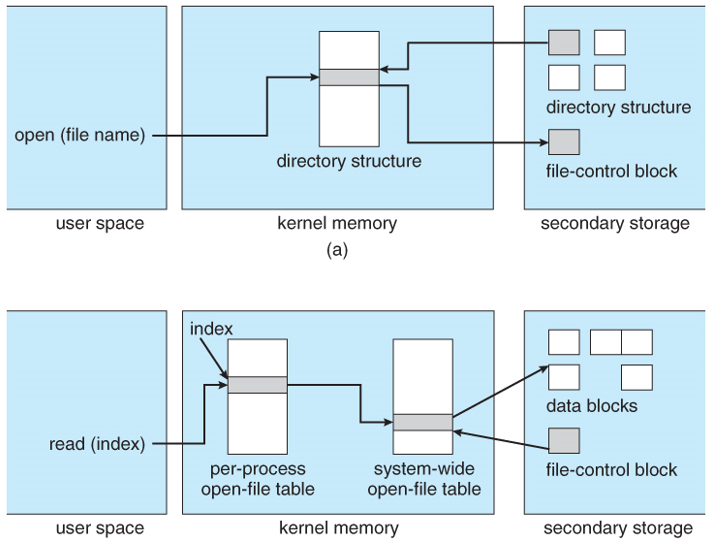


Figure: in-memory file system structure: a) file open b) file read

**Virtual File System:**

It is used to support multiple types of file system. The file system implementation consists of three major implementation details as shown in figure below. The first layer is the file system interface based on the open (), read (), write () and close () calls on the file descriptor. The second layer is the virtual file system (VFS) layer which serves two important function:

* It separates file-system-generic operation from their implementation by defining a clean VFS interface. Several implementation for the VFS interface may coexist on the same machine allowing transparent access to different types of file system mounted locally.
* It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on the file representing structure called vnode that contains a numerical designator for a network wide unique file. This network wide unique file is required for support of network file system. The kernel maintain one vnode structure for each active node (file or directory).

The VFS distinguishes local files from remote ones and local file further distinguished according to their file-system type. The VFS activates file-system specific operations to handle local request according to their file system types and calls the NFS protocol procedures for remote requests. File handles are constructed from relevant vnodes and are passed as arguments to these procedure.

The third layer is the layer implementing the file system type or the remote file system protocol.

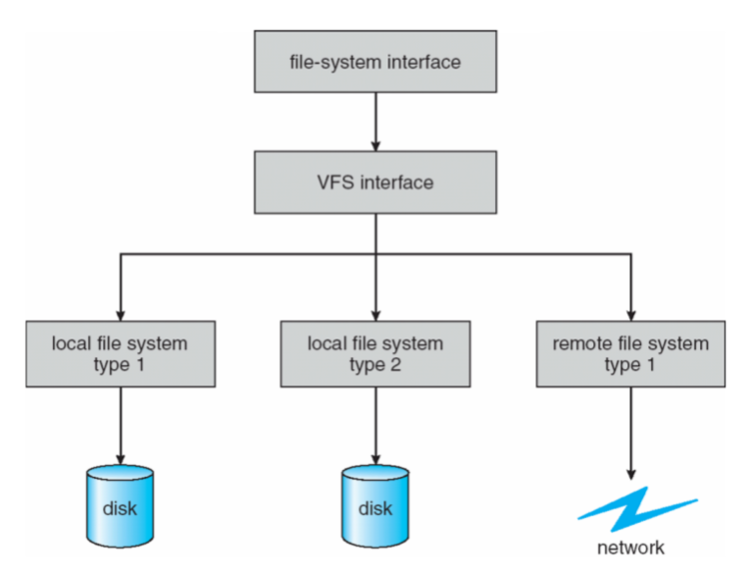


Figure: View of virtual file system.

**Directory Implementation:**

The selection of directory allocation and directory management algorithm significantly affects the efficiency, performance and reliability of the file system. Some ways to implement directory are:

1. **Linear List:**

It is a simplest method to implement a directory in which linear list of file name is used with a pointer to the data block. To create a new file, first directory is searched to be sure that no existing file has the same name. Then new entry is added at the end of the directory. To delete a file a directory is searched for a specific file name then releases the space allocated to it. To reuse a directory entry, the entry can be marked as unused by assigning it a special name such as all blank name or including a special unused used bit in a memory or attach it to a list of free directory. A link list can also be used to decrease a time required to delete a file.

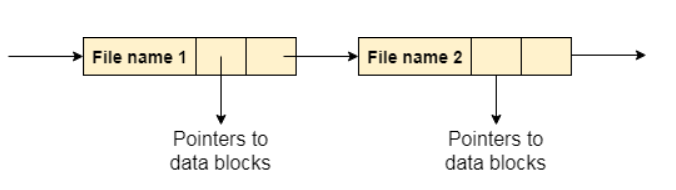


Figure: linear list

**Advantages**

* Simple to program

**Disadvantages:**

* To find a file requires linear search which is very time consuming. For this cache can be used to store recently used directory.

1. **Hash table:**

Another data structure used for a file directory is a hash table in which linear hash table takes a value computed from the file name and returns pointer to the file name in the linear list. A key value pair for each file in the directory gets generated and stored in the hash table. The key is generated by applying a hash function to each file and key points to the corresponding file stored in the directory. Insertion and deletion is also straight forward although some provision must be made for collision (situation in which two fie names hash to the same location).

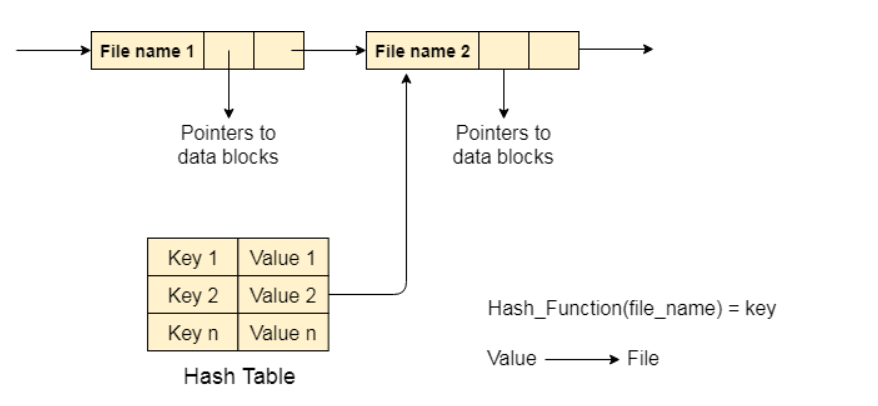


Figure: hash table for directory implementation

**Drawback:**

* Hash table are of fixed size and the dependence of the hash function on that size.

**File Allocation Method:**

Many files are stored in same disk. The main problem is how to allocate space to all the file so that the disk space is utilized effectively and file can be accessed quickly. The methods of allocating disk space are as follows:

1. **Contiguous Allocation:**

Contiguous allocation requires that each file occupy a set of contiguous block on the disk and disk address define a linear ordering on the disk. This allocation is defined by the disk address and length (in block unit) of the first block. For example if the file in n blocks long and starts at location b, then it occupies block b, b+1, b+2… b+n-1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

Accessing the file that has been allocated contiguously is easy. For sequential access the file system remembers the disk address of the last block referenced and when necessary reads the next block. For direct access to block “i” to the file that starts at block b we can immediately access block b + i.

Following figure shows the implementation of contiguous allocation:

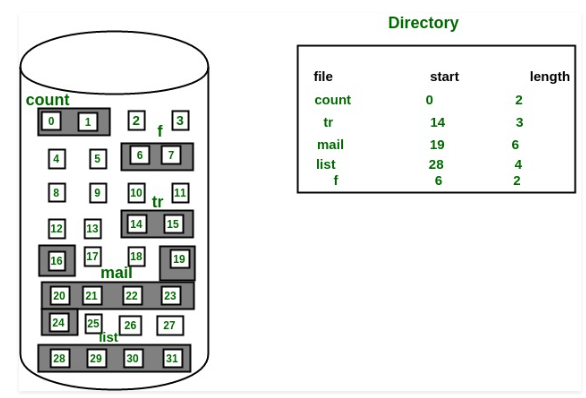


Figure: Linked allocation of disk space.

**Advantages:**

* Both sequential and direct accesses are supported.
* The number of seeks are minimal

**Disadvantages**

* Suffers from both internal and external fragmentation.
* Increasing the file size is difficult because it depends on the availability of contiguous memory at a particular instance.

1. **Linked Allocation:**

In this scheme, each file is a linked list of disk block and the disk blocks may be scattered anywhere on the disk. The directory contains the pointer to the first and the last block of the file. For example: a file of five block might start at block 9 and continue at block 16 then block 1 then block 10 and finally block 25. Each block contains a pointer to the next block and pointer are not available to the user. If each block is 512 byte in the size and a disk address (pointer) requires 4 bytes then the user sees blocks of 508 byte.

To create a new file new entry in the directory is created and each directory entry has a pointer to the first disk block of the file. A write to the file causes the free space management system to find a free block and this new block is written to and is linked to the end of the file. To read a file, blocks are read by following the pointer from block to block.

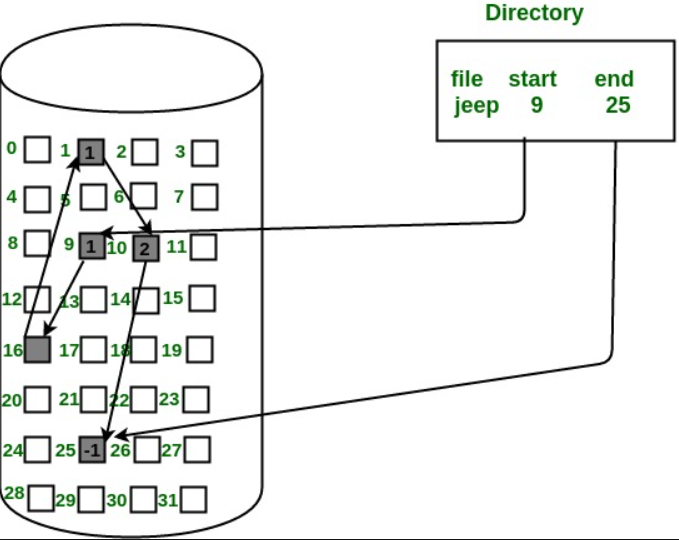


Figure: linked allocation of disk space

**Advantages:**

* Overcome the problem of external fragmentation.

**Disadvantages:**

* Only effective for sequential access. For direct access to ith block we have to start from first of the block and follow the pointer.
* Pointer will consume the space of total disk block.

Another variation used for linked allocation is File Allocation Table (FAT)

**File Allocation Table (FAT) for file allocation:**

Here, a section of the disk at the beginning of each volume is set aside to contain the table. The table has one entry for each disk block and is indexed by block number. The directory entry contains a block number of the first block of the file. The table entry indexed by that block number contains the block number of the next block in the file. The chain continues until it reaches the last block which has special end of file value as the table entry.

An unused block is indicated by table value of 0. Allocating a new block to a file is to find the first 0 valued table entry and replacing the previous end of file value with the address of the new block. The 0 is then replaced with the end of file value.

Following figure shows the implementation of FAT:

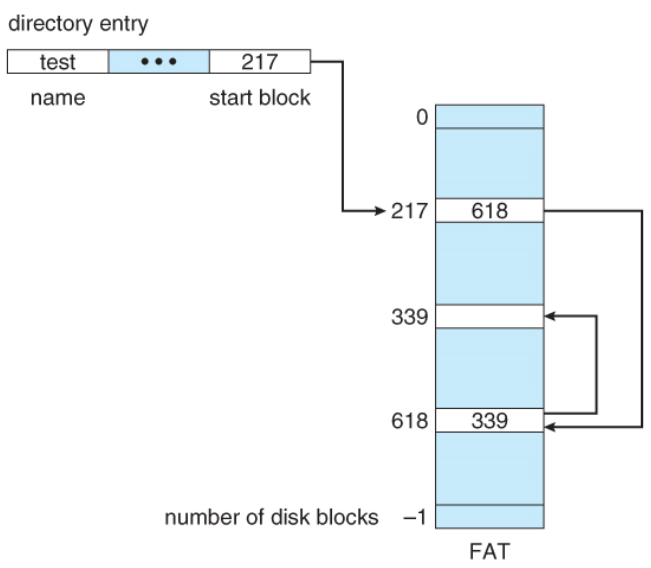


Figure: file allocation table (FAT)

1. **Indexed Allocation:**

Index allocation solves the problem of direct access exist in linked allocation by bringing all the pointers together into one location called index block. Each file has its own index block which is an array of disk block address. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block. To find and read the ith block we use the pointer in the ith index block entry.

When the file is created all pointers in the index block are set to null and when the ith block is first written a block is obtained from the free space manger and its address is put in the ith index block entry.

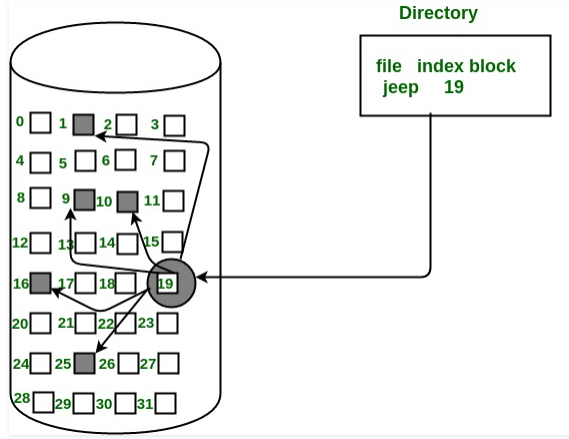


Figure: index allocation of disk space

For the file that are very large, single index bock may not be able to hold all the pointer. Following are the mechanism to solve this problem:

* **Linked scheme**: this scheme links two or more index block together for holding a pointers. Every index block would then contains a pointer or the address of the next index block.
* **Multilevel index:** in this scheme first index block is used to point the second index block which in turns point to the file block.
* **Combined scheme**: this scheme is used in UNIX file system where say, 15 pointers of the index block is kept in the file inode. The first 12 of this pointer point to the direct block which contains address of the block that contains data of the file. The next three pointer points to the indirect block. The first points to the single indirect block which is an index block containing not data but address block that do contain data. The second points to the double indirect block that contains the address of the block that contains pointer to the actual data block. The last pointer contains the address of a triple indirect block.

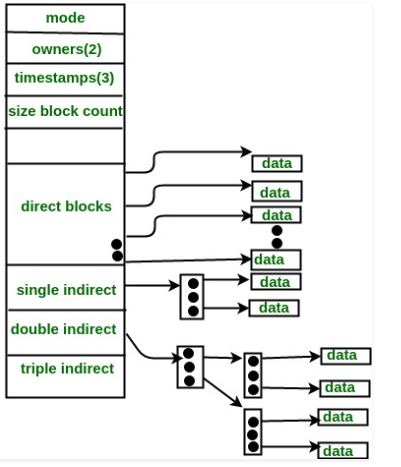


Figure: the UNIX inode

**Free Space Management:**

It is the process of reusing the space of disk from the deleted files for new files. To keep the track of free disk space the system maintains a free space list. The free space list records all free disk block (those not allocated to some file or directory). To create a file, a free space list is searched for the required amount of space and allocate that space to new file. This space is then removed from the free space list and when the file is deleted then its disk space is added to the free space list. The free space list can be implemented as:

1. **Bit Vector:**

It is also known as bit map in which each block is represented by 1 bit. If the block is free the bit is 1. If the block is allocated then the bit is 0.

Following figure shows the instance of a disk block on the disk where green block are allocated can be represented by the bitmap of 16 bit as: 0000111000000110.

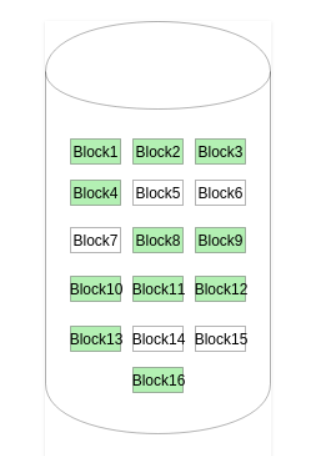


Figure: Bit map

One technique to find first free block on the system is to sequentially check each word in the bit map to see whether that value is not 0 since 0 valued word contains only 0 bits and represents a set of allocated blocks. The calculation of block number is:

Number of bits per word \* number of 0 valued word +offset of first 1 bit.

1. **Linked List:**

Another approach to free space management is to link together all the free blocks by keeping a pointer to the first block in a special block on the disk and caching it in memory. The first block contains the pointer to the next free block and so on.

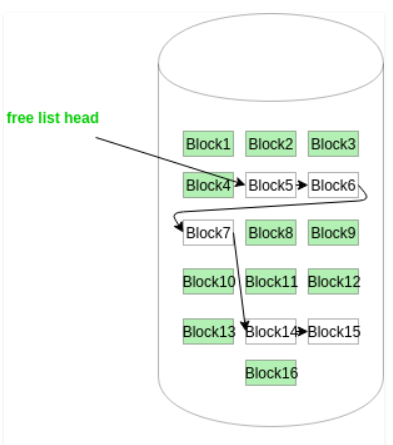


Figure: Linked list

The first block that is free is block number 5 so the block 5 contains the pointer to block 6 and block 6 to block 7 and so on. This scheme is not efficient for traversing as we must read each block which requires substantial I/O time.

1. **Grouping:**

This approach stores the n free block into first free block i.e. first block that is free stores the address of n free block. Out of these n block, n-1 block are actually free and last block contains the address of next free n block. Because of this the address of large number of free block can be found quickly.

1. **Counting:**

Here, rather than keeping the list of n free disk’s address the address of the first free block and the number of n of free contiguous free block that follow the first block is recorded in list. Each entry in the free space list then consists of a disk address and a count. For example: from the figure of bit map: the first entry of the free space list using counting would be {[address of block 5], 2} where 2 is the number contiguous free block after block 5.