File System Interface management

Unit 7

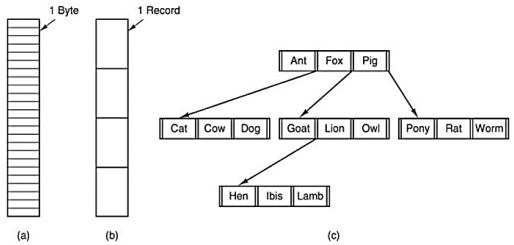
File Concept

- Files are logical units of information created by processes.
- Processes can read existing files and create new ones if needed.
- Information stored in files must be persistent that is not affected by process creation and termination.
- Files should only be disappear if we remove it.
- How files are structured, named, accessed, used, protected, implemented and managed are major topics in operating system design.
- The part of operating system dealing with file is called files system.

File Naming

- Files are an abstraction mechanism. They provide a way to store information on the disk and read it back later. This must be done in such a way as to shield the user from the details of how and where the information is stored, and how the disks actually work.
- The most important characteristics of any abstraction mechanism is the way the objects being named and managed.
- When a process creates a file, it gives the file a name.
- When the process terminates, the file continues to exist and can be accessed by other processes using its name.

File Structure



Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

1. **Byte Sequence:**

- The file in Fig. (a) is just an unstructured sequence of bytes. In effect, the operating system does not know or care what is in the file. All it sees are bytes. Any meaning must be imposed by user-level programs. Both UNIX and Windows 98 use this approach.

2. **Record Sequence:**

- In this model, a file is a sequence of fixed-length records, each with some internal structure. Central to the idea of a file being a sequence of records is the idea that the read operation returns one record and the write operation overwrites or appends one record. As a historical note, when the 80-column punched card was king many (mainframe) operating systems based their file systems on files consisting of 80-character records, in effect, card images

3. **Record Sequence:**

- In this organization, a file consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record. The tree is sorted on the key field, to allow rapid searching for a particular key.

File Type

- File type refers to the ability of operating system to distinguish different types of file such as text files, source files and binary files etc. Many operating systems support many types of files.
- Operating system like Ms-DOS and UNIX have the following types of files.

Ordinary files

- These are the files that contain user information
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.
- These files are of two types;
- **Character special files**: Data is handled character by character as in case of terminals or printers.
- **Block special files**: Data is handled in blocks as in the case of disks and tapes.

File Attributes

- Every file has a name and its data. In addition, all operating systems associate other information with each file, for example, the date and time the file was created and the file's size. We will call these extra items the file's attributes although some people called them metadata. The list of attributes varies considerably from system to system.

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File Access

- When a file is used then the stored information in the file must be accessed and read into the memory of a computer system. Various mechanism are provided to access a file from the operating system.
  i. Sequential access
  ii. Direct Access
  iii. Index Access

Sequential Access:

It is the simplest access mechanism, in which information stored in a file are accessed in an order such that one record is processed after the other. For example editors and compilers usually access files in this manner.

Direct Access:

It is an alternative method for accessing a file, which is based on the disk model of a file, since disk allows random access to any block or record of a file. For this method, a file is viewed as a numbered sequence of blocks or records which are read/written in an arbitrary manner, i.e. there is no restriction on the order of reading or writing. It is well suited for Database management System.

Index Access:

In this method an index is created which contains a key field and pointers to the various block. To find an entry in the file for a key value , we first search the index and then use the pointer to directly access a file and find the desired entry.

With large files , the index file itself may become too large to be keep in memory. One solution is to create an index for the index file. The primary index file would contain pointers to secondary index files, which would point to the actual data items.

File Operations

Files exist to store information and allow it to be retrieved later. Different systems provide different operations to allow storage and retrieval. Below is a discussion of the most common system calls relating to files.

1. **Create:** The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes.
2. **Delete:** When the file is no longer needed, it has to be deleted to free up disk space. A system call for this purpose is always provided.
3. **Open:** Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.
4. **Close:** When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up some internal table space. Many systems encourage this by imposing a maximum number of open files on processes. A disk is written in blocks, and closing a file forces writing of the file's last block, even though that block may not be entirely full yet.
5. **Read:** Data are read from file. Usually, the bytes come from the current position. The caller must specify how much data are needed and must also provide a buffer to put them in.
6. **Write:** Data are written to the file, again, usually at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten and lost forever.

File Operations

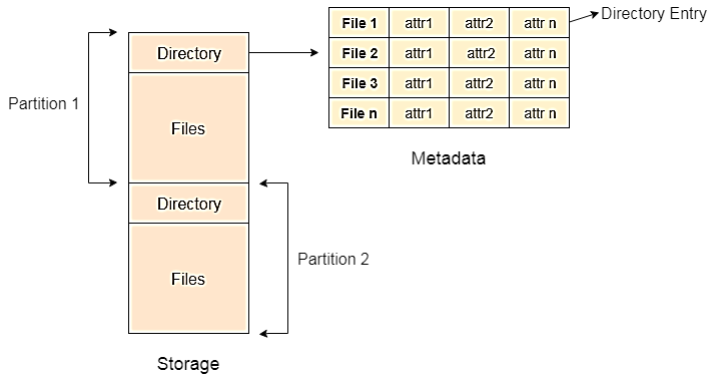
7. **Append:** This call is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide multiple ways of doing the same thing, and these systems sometimes have append.
8. **Seek:** For random access files, a method is needed to specify from where to take the data. One common approach is a system call, seek, that repositions the file pointer to a specific place in the file. After this call has completed, data can be read from, or written to, that position.
9. **Get attributes:** Processes often need to read file attributes to do their work. For example, the UNIX make program is commonly used to manage software development projects consisting of many source files. When make is called, it examines the modification times of all the source and object files and arranges for the minimum number of compilations required to bring everything up to date. To do its job, it must look at the attributes, namely, the modification times.
10. **Set attributes:** Some of the attributes are user settable and can be changed after the file has been created. This system call makes that possible. The protection mode information is an obvious example. Most of the flags also fall in this category.
11. **Rename:** It frequently happens that a user needs to change the name of an existing file. This system call makes that possible. It is not always strictly necessary, because the file can usually be copied to a new file with the new name, and the old file then deleted.
12. **Lock:** Locking a file or a part of a file prevents multiple simultaneous access by different process. For an airline reservation system, for instance, locking the database while making a reservation prevents reservation of a seat for two different travelers.

File Descriptors

- For most operating systems, a file descriptor (FD) is a small non-negative integer that helps in identifying an open file within a process while using input/output resources like network sockets or pipes.
- In a way, it can be considered as an index table of open files. When there are read, write or closing file operations, one of the input parameters considered is the file descriptor.
- The descriptor is identified by a unique non-negative integer, such as 0, 12, or 567. At least one file descriptor exists for every open file on the system.
- In the Unix operating system, the standard input is represented by file descriptor 0, the standard output is represented by file descriptor 1 and standard error file is represented by file descriptor 2.
- When a process makes a successful request to open a file, the kernel returns a file descriptor which points to an entry in the kernel's global file table.
- The file table entry contains information such as the inode of the file, byte offset, and the access restrictions for that data stream (read-only, write-only, etc.).

Directories

- Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.
- To get the benefit of different file systems on the different operating systems, A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.
- Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file.



- A directory can be viewed as a file which contains the Meta data of the bunch of files

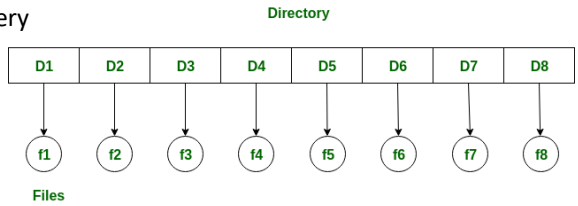
Directories

Single level directory

- The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand. A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have a unique name. if two users call their dataset test, then the unique name rule violated.

Advantages:

- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.



Disadvantages:

- There may chance of name collision because two files can have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

Directories

Two level directory

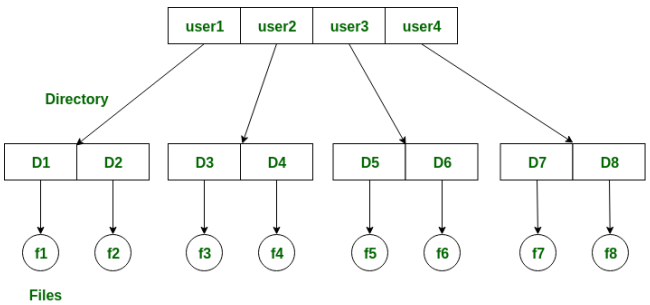
- As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user. In the two-level directory structure, each user has their own *user files directory (UFD)*. The UFDs have similar structures, but each lists only the files of a single user. system's *master file directory (MFD)* is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

Advantages:

- We can give full path like /User-name/directory-name/.
- Different users can have the same directory as well as the file name.
- Searching of files becomes easier due to pathname and user-grouping.

Disadvantages:

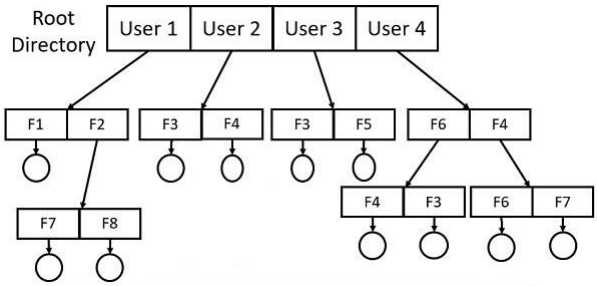
- A user is not allowed to share files with other users.
- Still, it not very scalable, two files of the same type cannot be grouped together in the same user.



Directories

Hierarchical Directory Structure

- In Hierarchical directory structure, the users can create directories under the root directory and can also create sub-directories under this structure. As the user is free to create many sub-directories, it can create different sub-directories for different file types.



- Here, the files are accessed by their location using the path. There are two types of paths to locate the file in this directory structure:
- **Absolute Path:** Here, the path for the desired file is described by considering the root directory as the base directory.
- **Relative Path:** Here, either user's directory is considered as the base directory or the desired file directory is considered as the base directory.

Directories

Tree-structured Directory

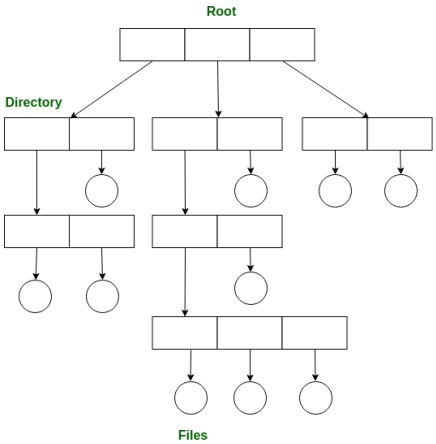
- Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height. This generalization allows the user to create their own subdirectories and to organize their files accordingly.
- A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

Advantages:

- Very general, since full pathname can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute paths as well as relative.

Disadvantages:

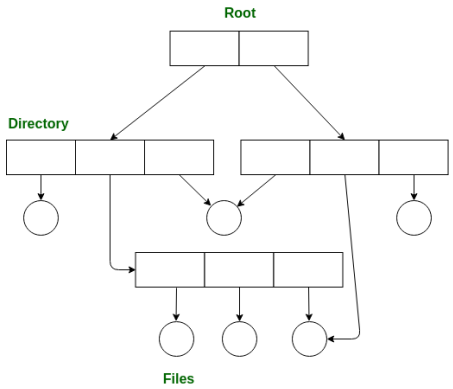
- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.



Directories

Acyclic graph Directory

- An acyclic graph is a graph with no cycle and allows us to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory. It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.
- It is the point to note that the shared file is not the same as the copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



Advantages:

- We can share files.
- Searching is easy due to different-different paths.

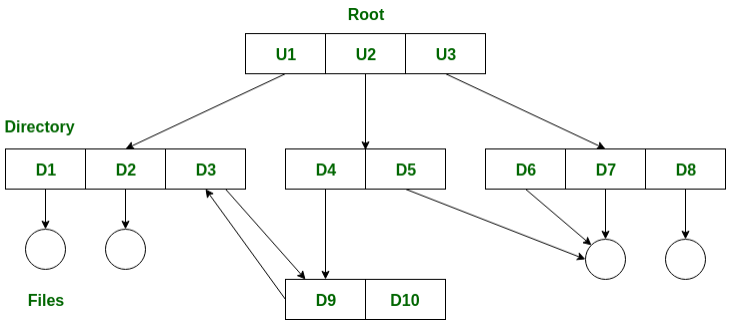
Disadvantages:

- We share the files via linking, in case deleting it may create the problem,
- If the link is a soft link then after deleting the file we left with a dangling pointer.
- In the case of a hard link, to delete a file we have to delete all the references associated with it.

Directories

General Graph Directory Structure

- In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory. The main problem with this kind of directory structure is to calculate the total size or space that has been taken by the files and directories.



Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

Path Names

- A sequence of symbols and names that identifies a file.
- Every file has a name, called a filename, so the simplest type of pathname is just a filename.
- If we specify a filename as the pathname, the operating system looks for the file in our current working directory.
- If the file resides in a different directory, we must tell the operating system how to find that directory. We perform this by specifying a path that the operating system must follow.
- The pathname always starts from our working directory or from the root directory.
- Each operating system has its own rules for specifying paths. In DOS systems, the root directory is named \ and each subdirectory is separated by an additional backslash.
- In UNIX, the root directory is named /, and each subdirectory is followed by a slash.
- In Macintosh environments, directories are separated by a colon.

Operations on Directory

- A directory contains the entries of all related files. For organizing the directory in the better way the user must be able to insert, delete, search and list entries in the directory.

1. **Searching**

- A directory can be **searched** for a particular file or for another directory. It can also be searched to list all the files with the same name.

2. **Creating**

- A new file can be created and inserted to the directory or new directory can be created keeping in mind that its name must be unique under that particular directory.

3. **Deleting**

- If a file is no longer needed by the user, it can be deleted from the directory. The entire directory can also be deleted if it is not needed. An empty directory can also be deleted. When a directory is empty it is resembled by dot and dot dot.

4. **List a directory**

- List of all the files in the directory can be retrieved and also the contents of the directory entry, for each file in a list. To read the list of all the files in the directory, it must be opened and after reading the directory must be closed to free up the internal tablespace.

Operations on Directory

5. Renaming

- The name of the file or a directory represents the content it holds and its use. The file or directory can be renamed in case, the content inside or the use of file get changed. Renaming the file or directory also changes its position inside the directory.

6. Link

- The file can be allowed to appear in more than one directory. Here, the system call creates a link between the file and the name specified by the path where the file is to appear.

7. Unlink

- If the file is unlinked and is only present in one directory its directory entry is removed. If the file appears in multiple directories, only the link is removed.

Protection

- File system often contain information that is highly valuable to their users. Protecting these information against unauthorized usage is the major concern of all file system.

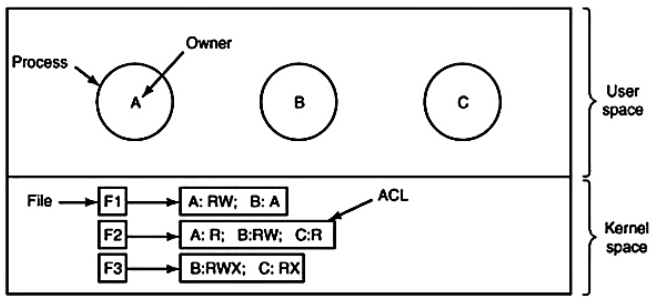
Defense Mechanism:

- To protect the system, we must take security at four levels.
- **Physical:** Computer system must be physically secured.
- **Human:** User authentication by secure password.
- **Network:** Setting firewall
- **Operating System:** Checking its file system regular interval of time.

Types of Access

- The following low-level operations are often controlled:
  - Read - View the contents of the file
  - Write - Change the contents of the file.
  - Execute - Load the file onto the CPU and follow the instructions contained therein.
  - Append - Add to the end of an existing file.
  - Delete - Remove a file from the system.
  - List -View the name and other attributes of files on the system.
- Higher-level operations, such as copy, can generally be performed through combinations of the above.
- Access control is a security technique that can be used to regulate who or what can view or use resources in a computing environment.
- There are two main types of access control: Physical and Logical
- Physical access control limits access to campuses, buildings, rooms and physical IT assets. Logical access limits connections to computer network, system files and data.
- Access controls are necessary to protect the confidentiality, integrity and availability of objects (and by extension, their information and data).
- The term access control is used to describe a broad range of controls, from forcing a user to provide a valid username and password to log on to preventing users from gaining access to a resource outside of their sphere of access.

Access Control List



- An ACL consists of a set of ACL entries. An ACL entry specifies the access permissions on the associated object for an individual user or a group of users as a combination of read, write and execute permissions.
- Each file has an ACL associated with it. File F1 has two entries in its ACL (separated by a semicolon). The first entry says that any process owned by user A may read and write the file. The second entry says that any process owned by user B may read the file. All other accesses by these users and all accesses by other users are forbidden. Note that the rights are granted by user, not by process. As far as the protection system goes, any process owned by user A can read and write file F1. It does not matter if there is one such process or 100 of them. It is the owner, not the process ID, that matters. File F2 has three entries in its ACL: A, B, and C can all read the file, and in addition B can also write it. No other accesses are allowed. File F3 is apparently an executable program, since B and C can both read and execute it. B can also write it.

Access Control Matrix

- The access controls provided with an operating system typically authenticate principals using some mechanism such as passwords or Kerberos, then mediate their access to files, communications ports, and other system resources. Their effect can often be modelled by a matrix of access permissions, with columns for files and rows for users. We'll write **r for permission to read, w for permission to write, x for permission to execute a program, and (–) for no access at all**, as shown in Figure below.

	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	rw	r
Alice	x	x	rw	–
Bob	rx	r	r	r

- In this simplified example, Sam is the system administrator, and has universal access (except to the audit trail, which even he should only be able to read). Alice, the manager, needs to execute the operating system and application, but only through the approved interfaces—she mustn't have the ability to tamper with them. She also needs to read and write the data. Bob, the auditor, can read everything. Access control matrices (whether in two or three dimensions) can be used to implement protection mechanisms, as well as just model them. But they do not scale well. For instance, a bank with 50,000 staff and 300 applications would have an access control matrix of 15 million entries. This is inconveniently large. It might not only impose a performance problem but also be vulnerable to administrators' mistakes. We will usually need a more compact way of storing and managing this information. The two main ways of doing this are to use groups or roles to manage the privileges of large sets of users simultaneously, or to store the access control matrix either by columns (access control lists) or rows (capabilities, sometimes known as "tickets") or certificates .

File Allocation methods

Contiguous allocation:

- It requires each file to occupy a set of contiguous addresses on a disk. It sore each file as a contiguous run of disk blocks. Thus on a disk with 1-KB blocks, a 50-KB file would be allocated 50 consecutive blocks. Both sequential and direct access is supported by the contiguous allocation method. Contiguous disk space allocation has two significant advantages.

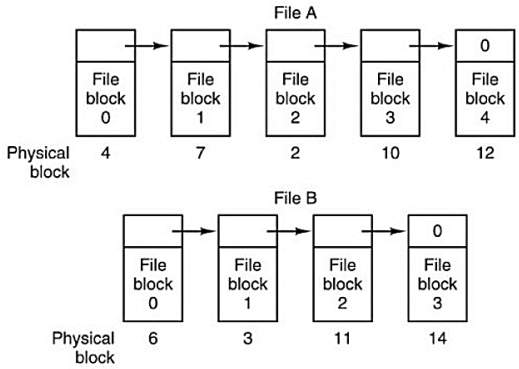
1. First, it is simple to implement because keeping track of where a file's blocks are is reduced to remembering two numbers: the disk address of the first block and the number of blocks in the file. Given the number of the first block, the number of any other block can be found by a simple addition.
2. Second, the read performance is excellent because the entire file can be read from the disk in a single operation. Only one seek is needed (to the first block). After that, no more seeks or rotational delays are needed so data come in at the full bandwidth of the disk.

- Thus contiguous allocation is simple to implement and has high performance. Unfortunately, contiguous allocation also has a major drawback: in time, the disk becomes fragmented, consisting of files and holes. It needs compaction to avoid this.
- Example of contiguous allocation: CD and DVD ROMs

File Allocation methods

Linked List Allocation:

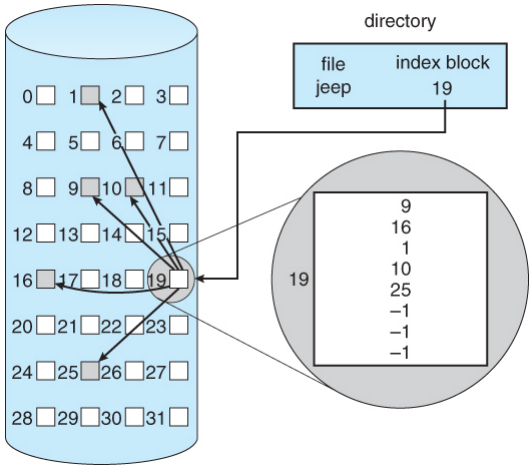
- keep each file as a linked list of disk blocks as shown in the fig. The first word of each block is used as a pointer to the next one. The rest of the block is for data.
- Unlike contiguous allocation, every disk block can be used in this method. No space is lost to disk fragmentation. The major problem with linked allocation is that it can be used only for sequential access files. To find the i^{th} block of a file, we must start at the beginning of that file, and follow the pointers until we get the i^{th} block. It is inefficient to support direct access capability for linked allocation of files.
- Another problem of linked list allocation is reliability. Since the files are linked together with the pointer scattered all over the disk. Consider what will happen if a pointer is lost or damaged.



File Allocation methods

Indexed allocation (I-Nodes):

- It solves the external fragmentation and size declaration problems of contiguous allocation. In this allocation all pointers are brought together into one location called Index block. Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block.



File Allocation methods

- To read the ith block, we use the pointer in the ith index block entry to find and read the desired block. This scheme is similar to the paging scheme.