# Unit 4: File Systems

ER. RAJAN KARMACHARYA
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
ST. XAVIER'S COLLEGE

**Includes…**

**4.1 Files**

**4.2 Directories**

**4.3 File System Implementation**

Er. Rajan Karmacharya

# Files

- Issues
  - **How to store the large amount of data into the computer?**
  - **What happens when process terminates or is killed using some data?**
  - **How to assign the same data to the multiple process**

- The solution to all these problems is to store information on disks or on other external media called FILES.

Er. Rajan Karmacharya

# Files

- A file is a named collection of related information normally resides on a secondary storage device such as disk or tape.
- Commonly, files represent programs (both source and object forms) and data; data files may be numeric, alphanumeric or binary
- Information stored ***in files must be persistent***, not be effected ***by power failures*** and ***system reboot***
- The files are managed by the OS and part of the OS that ***is responsible to manage files is known as the file system***

Er. Rajan Karmacharya

# File System Issues

- How to create a file?
- How are they named?
- How are they structured?
- What operations are allowed on files?
- How to protect them?
- How are they accessed or used?
- How to implement them?

Er. Rajan Karmacharya

# File Naming

- When a process creates a file, a file name is given which continues to exist and can be accessed by other processes even after the process terminates.
- A file is named, for the convenience of its human users and it is referred to by its name.
- A name is a string of characters which may be digits or special characters
- Some system differentiate between lowercase and uppercase characters such as UNIX and other consider equivalent like MS DOS.
- Normally the string of max 8 characters (MS DOS)are legal file name but recent systems support as long as 255 characters (Windows)

Er. Rajan Karmacharya

# File Naming

- Many OS support two part file names separated by a period.
- The part following the period is known as extension
- The extension name resembles the type of the file
- Some systems may have two or more extensions such as Prog1.c.Z ---- A C source file compressed using Ziv-Lampel Algorithm

Er. Rajan Karmacharya

# Typical File Extensions

| Extension | Meaning |
|---|---|
| file.bak | Backup file |
| file.c | C source program |
| file.gif | Compuserve Graphical Interchange Format image |
| file.hlp | Help file |
| file.html | World Wide Web HyperText Markup Language document |
| file.jpg | Still picture encoded with the JPEG standard |
| file.mp3 | Music encoded in MPEG layer 3 audio format |
| file.mpg | Movie encoded with the MPEG standard |
| file.o | Object file (compiler output, not yet linked) |
| file.pdf | Portable Document Format file |
| file.ps | PostScript file |
| file.tex | Input for the TEX formatting program |
| file.txt | General text file |
| file.zip | Compressed archive |

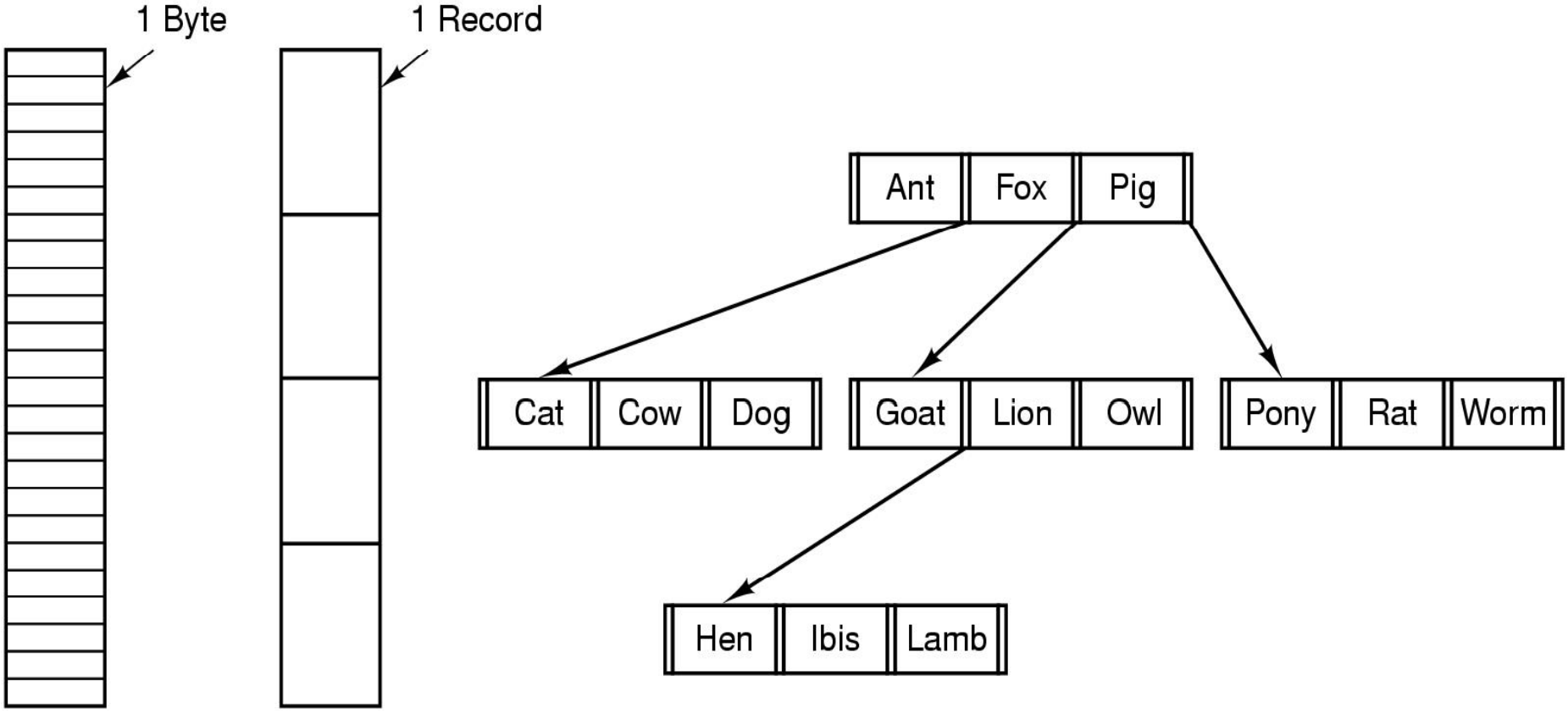
Er. Rajan Karmacharya

# File Structure

- Files must be structured and understood by the OS.
- Files can be structured in many ways and the most general structures are
  - **Unstructured**
  - **Record Structured**
  - **Tree Structured**

Er. Rajan Karmacharya

# File Structure




(a)

**Unstructured**

(b)

**Record Structured**

(c)

**Tree Structured**

Er. Rajan Karmacharya

# File Structures

- ***Unstructured***
  - Consists of unstructured sequence of bytes or words
  - OS does not know or care what is in the file
  - Any meaning must be imposed by the user level program
  - Provides maximum flexibility; user can put anything they want and name them anyway that is convenient
  - Both UNIX and Windows use these approach.

Er. Rajan Karmacharya

# File Structures

- ***Record Structured***
  - A file is a sequence of fixed length records, each with some internal structure.
  - Each read operation returns one records and write operation overwrites or append one record
  - May old mainframe systems use this structure

Er. Rajan Karmacharya

# File Structures

- ***Tree Structured***
  - File consists of tree of records not necessarily all the same length.
  - Each containing a key field in a fixed position in the record sorted on the key to allow rapid searching.
  - The operation is to get the record with the specific key
  - Used in large mainframe for commercial data processing

Er. Rajan Karmacharya

# File Types

- ***Many OS support several types of files***
- ***Regular files:*** contains user information and are generally ACSII or Binary
- ***Directories:*** system files for maintaining the structure of the file system
- ***Character Special Files:*** related to I/O and used to model serial I/O devices such as terminals, printers and networks
- ***Block special files:*** used to model disks

Er. Rajan Karmacharya

# File Types

- ***ASCII Files***
  - Consists of line of text where each line is terminated either by carriage return or by line feed character or both
  - They can be displayed and printed as it is and can be edited with ordinary text editor

- ***Binary Files***
  - Consists of sequence of byte only
  - They have some internal structure known to programs that use them
  - Many OS use extension to identify the file type; but UNIX like OS use a magic number to identify the file type

Er. Rajan Karmacharya

# Access Methods

- ***Sequential Access***
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was magnetic tape

- ***Direct Access***
  - bytes/records read in any order
  - essential for data base systems
  - Used for immediate access to large amount if information
  - read can be …
    - move file marker (seek), then read or …
    - read and then move file marker

Er. Rajan Karmacharya

# File Attributes

- In addition to name and data, all other information about file is termed as file attribute
- The file attributes may vary 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 |

Er. Rajan Karmacharya

# File Operations

- OS provides system calls to perform operations on files. Some common calls are
- ***Create :*** If disk space is available, it creates new file without data.
- ***Delete:*** Deletes files to free up disk space.
- ***Open:*** Before using a file, a process must open it.
- ***Close:*** When all access are finished, the file should be closed to free up the internal table space.
- ***Read :*** Reads data from file
- ***Write:*** Writes data to a file
- ***Append:*** Adds data to the end of a file
- ***Seek :*** Repositions the file pointer to a specific place in the file
- ***Get :*** Returns file attributes for processing
- ***Set :*** To set the user settable attributes when files changed
- ***Rename :*** Rename a file

Er. Rajan Karmacharya

# Directory Structure

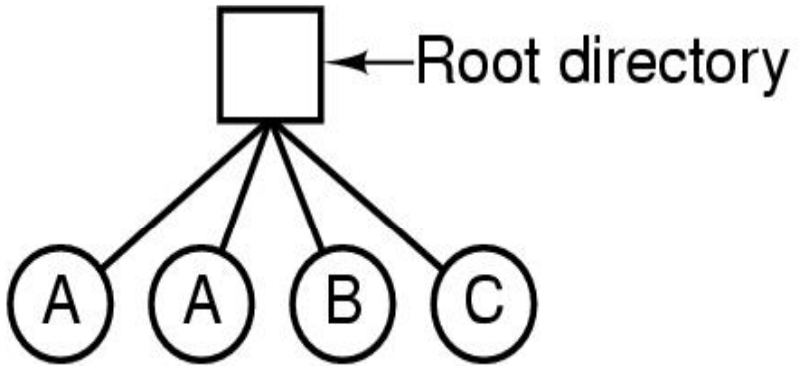
- A directory is a node containing information about files.
- Directories may have different structures

Er. Rajan Karmacharya

# Directory Structure

- ***Single Level Directory***
  - All files are contained in the same directory
  - Easy to support and understand but difficult to manage large amount of files and to manage different users
  - Different users may accidently use the same name for their files where the later overwrites the earlier
  - Advantage of this scheme is the simplicity and ability to locate files quickly; there is only once place to look after all.

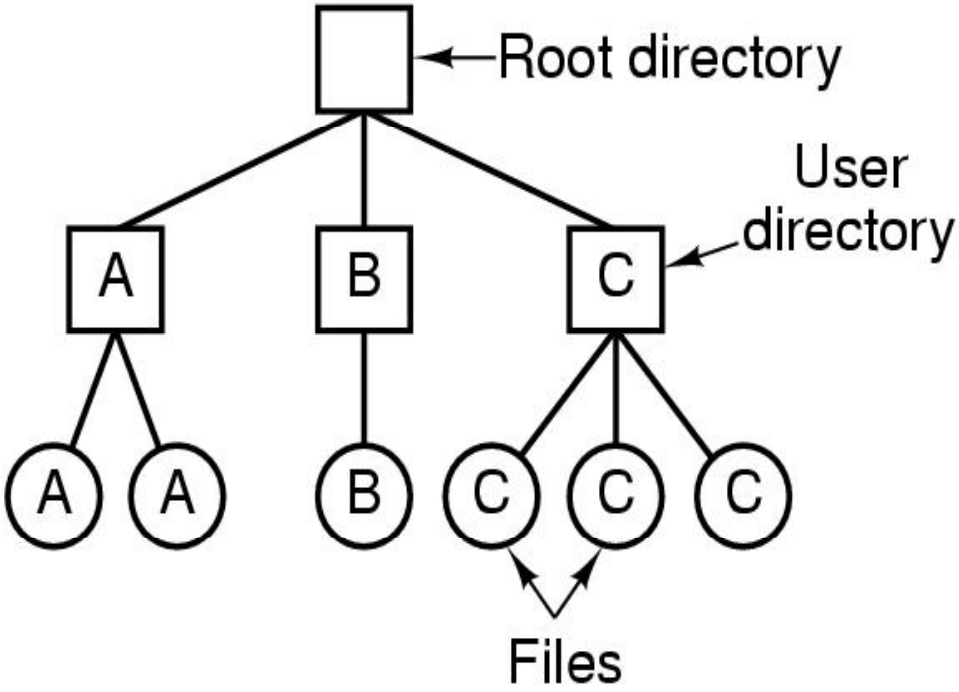
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C



Er. Rajan Karmacharya

# Directory Structure

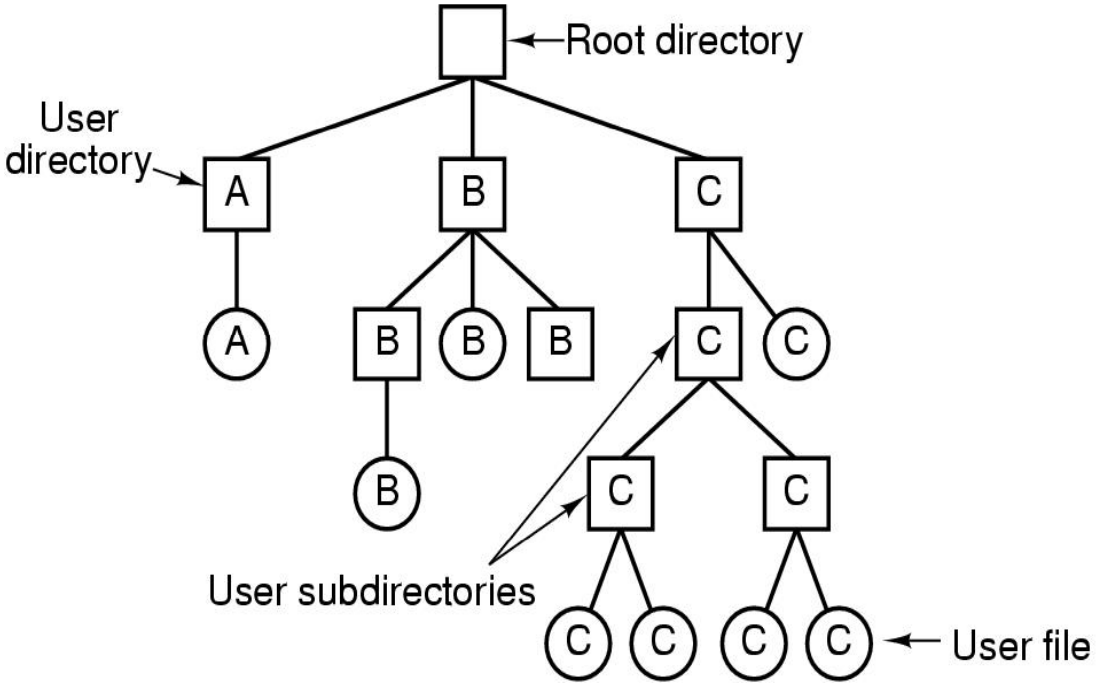
- *Two Level Directory*
  - Separate directory for each user
  - Used on a multiuser computer and on a simple network computer
  - It has problem when users want to cooperate on some task and to access one another's files.
  - It also cause problem when a single user has large number of files.



Er. Rajan Karmacharya

# Directory Structure

- ***Hierarchical Level Directory***
  - Generalization of two level structure to a tree of arbitrary height
  - This allows the user to create their own subdirectories and to organize their files.
  - To allow to share the directory for different user acyclic graph is used
  - Nearly all modern file systems are organized in this manner



Er. Rajan Karmacharya

# Path Names

- ***Absolute Path Names***
  - Path name starting from root directory to the file
  - Path separated by / in UNIX ,\ in Windows and > in MULTICS
  - Windows ---- \usr\ast\mailbox
  - Linux ---- /usr/ast/mailbox
  - MULTICS --- >usr>ast>mailbox
- ***Relative Path Names***
  - Concept of working directory
  - A user can designate one directory as the current working directory in which all path names not beginning at the root directory are taken relative to the working directory.
  - E.g. bin/lab2 is enough to locate same file if current working directory is /usr/user1

- *cp /usr/ast/mailbox /usr/ast/mailbox1*
- *cp mailbox mailbox1*
- *do exactly the same work if the working directory is /ust/ast*

Er. Rajan Karmacharya

# Directory Operations

- Create : A directory is created
- Delete : A directory is deleted
- OpenDir : Directories can be read
- CloseDir : After read, it should be closed to free up internal table space
- Rename : Rename a directory
- Link : allows a file to appear in more than one directory. This system call specifies an existing file and a path name, and creates a link from the existing file to the name specified by the path.
- Unlink : A directory entry is removed

Er. Rajan Karmacharya

# File System Implementation

- How are files and directories are stored?
- How disk space is managed?
- How to make every thing work efficiently and reliably?
- Files can be implemented in two ways
  - **Contiguous Allocation Method**
  - **Linked Allocation Method**

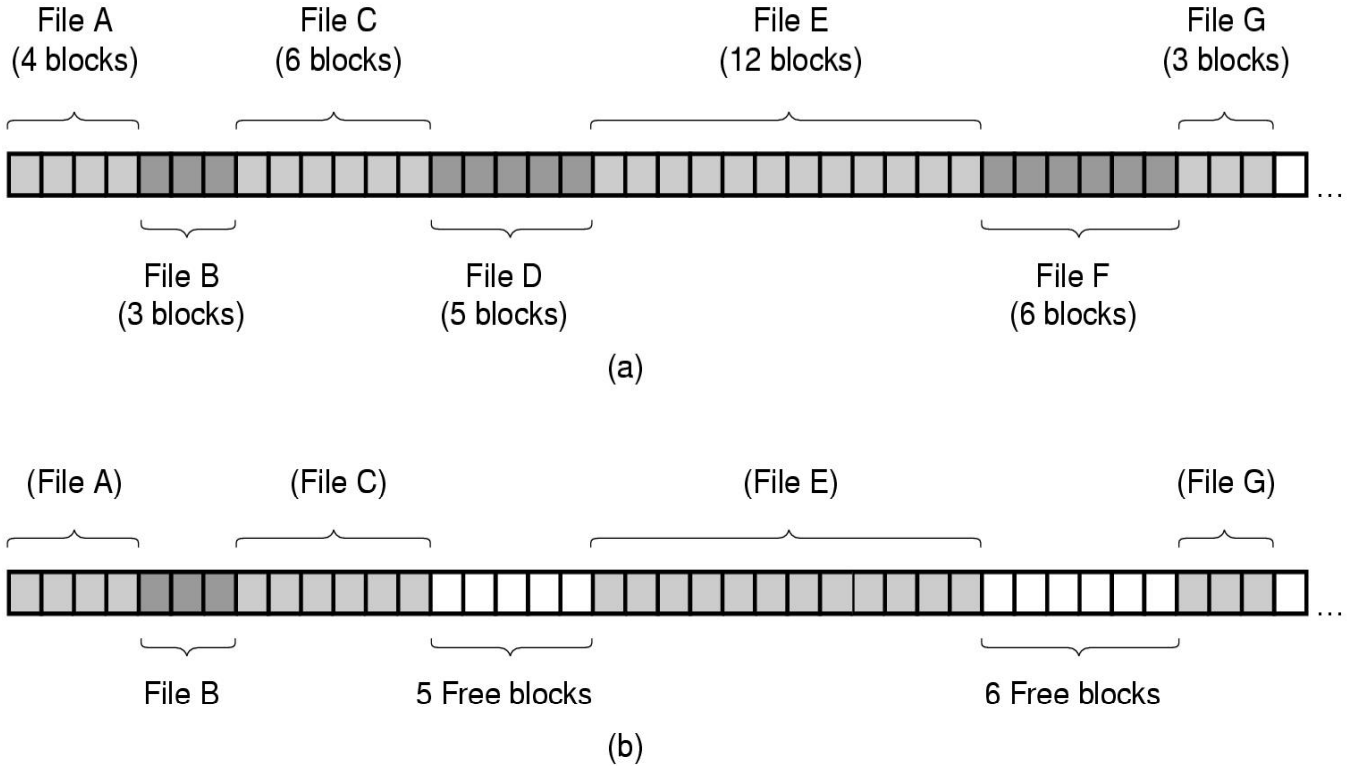
Er. Rajan Karmacharya

# Allocation Method: Contiguous Allocation

- The simplest allocation scheme is to store each file as a contiguous run of disk blocks.
- Each file occupy a set of contiguous block on the disk
- Disk address define a linear ordering on the disk.
- File is defined by the disk address and length in block units
- On a disk with 1 kb blocks, a 50 kb file is allocated 50 consecutive block
- With 2 kb blocks, a 50 kb file would be allocated 25 consecutive blocks
- Both sequential and direct access can be supported by contiguous allocation

Er. Rajan Karmacharya

# Allocation Method: Contiguous Allocation




(a) Contiguous allocation of disk space for 7 files
(b) State of the disk after files *D* and *E* have been removed

Er. Rajan Karmacharya

# Allocation Method: Contiguous Allocation

- Advantages:
  - ***Simple to implement:*** Accessing a file that has been allocated contiguously is easy
  - ***High performance:*** The entire file can be read in single operation i.e. decrease the seek time

- Problems:
  - ***Fragmentation:*** When files are allocated and deleted the free disk space is broken into holes
  - ***Dynamic storage allocation problem:*** Searching of right holes requires pre-information of file size to put it into a hole

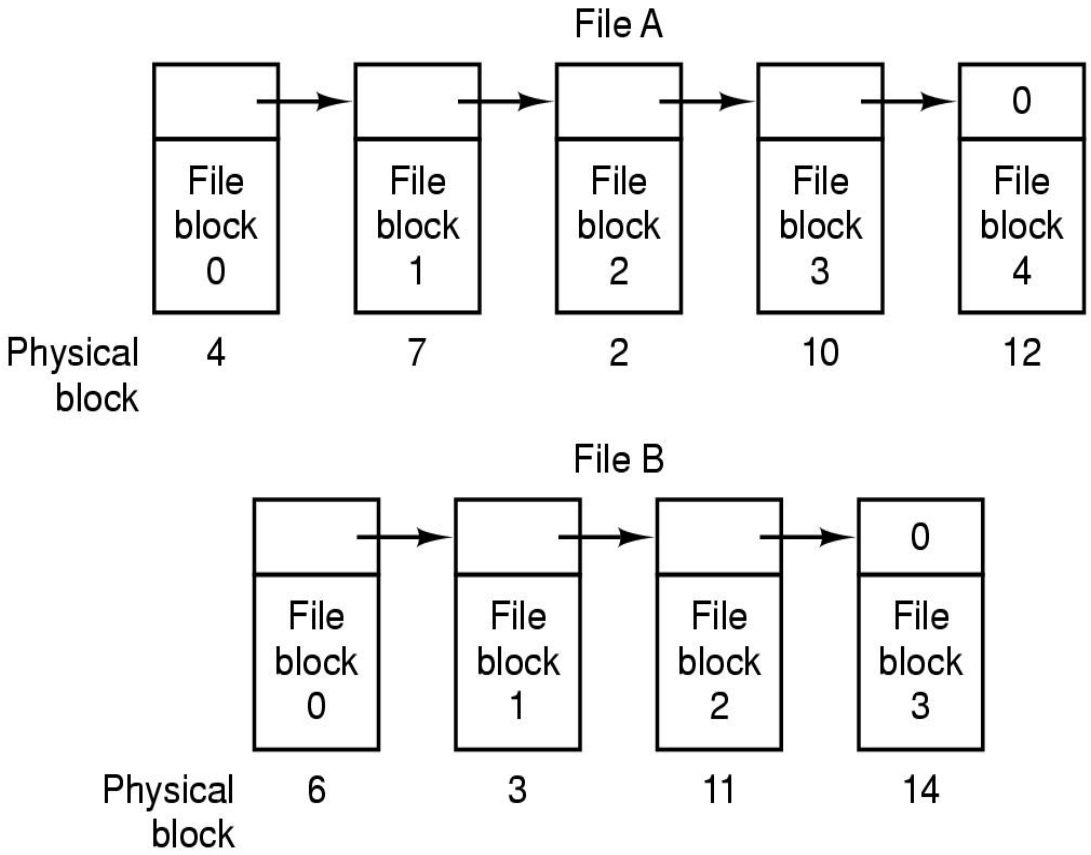
Er. Rajan Karmacharya

# Allocation Method: Linked Allocation

- Each file is a linked list of disk blocks; the disk block may be scattered anywhere on the disk.
- Each block contains a pointer to the next block of same file.
- To create the new file, we simply create a new entry in the directory; with linked allocation, each directory entry has a pointer to the first disk block of the file
- Unlike contiguous allocation, every disk can be used in this method.
- No space is lost to disk fragmentation except for internal fragmentation.

Er. Rajan Karmacharya

# Allocation Method: Linked Allocation




- File A uses disk block 4,7,2,10 and 12 and
  - File B uses disk block 6,3,11 and 14

Er. Rajan Karmacharya

# Allocation Method: Linked Allocation

- It solves all problems of contiguous allocation but it can be used only for sequential access file: random access is extremely slow
- Each block access requires disk seek
- It also requires space for pointer
- Solution : Using File Allocation Table (FAT)
  - The table has one entry for each disk block containing next block number for the file. This resides at the beginning of each disk partition.

Er. Rajan Karmacharya

# Directory Implementation

- The directory entry provides the information needed to find the disk block.
- The file attributes are stored in the directory
- Directories can be implemented in two ways
  - **Linear List**
  - **Hash Table**

Er. Rajan Karmacharya

# Directory Implementation- Linear List

- Use the linear list of the file names with pointer to the data blocks
- It requires linear search to find  a particular entry
- **Advantages:** Simple to implement but time consuming to execute
- **Problem:** Linear search to a file is slow

- *To create a new file, we must first search the directory to be sure that no existing file has the same name. Then we add a new entry at the end of the directory. To delete a file, we search for the named file and release the space allocated to it.*

- *Requires cache and sorting and may use B-Tree*

Er. Rajan Karmacharya

# Directory Implementation – Hash Table

- It consists linear list with has table.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Thus, it greatly decreases the search time.
- If that key is already in use, a linked list is constructed.
- **Advantages:** greatly decreases the file search time.
- **Problem:** It is fixed size and dependence of the hash function on that size.

- *Insertion and deletion are also fairly straight forward although some provision must be made for collisions situations in which two file names has to the same location.*
- *If we make a hash table that holds 64 entries, it converts filenames into integers from 0 to 63*
- *Later if we create a 65$^{th}$ file, we must enlarge the hash table say 128 entries. As a result we need a new hash function that must map files to the range of 0 – 127 and reorganize existing directories*

Er. Rajan Karmacharya

# Free Space Management

- To keep track of free blocks, system maintains the free space list
- The free space list records all free  blocks those not allocated to some file or directory
- To create a file, system searches the free space list for required amount of space, and allocate that space to the new file and removed from free space list.
- When a file is deleted, its disk space is added to the free space list
- The free space list can be implemented in two ways
  - ***Bitmap***
  - ***Linked List***

Er. Rajan Karmacharya

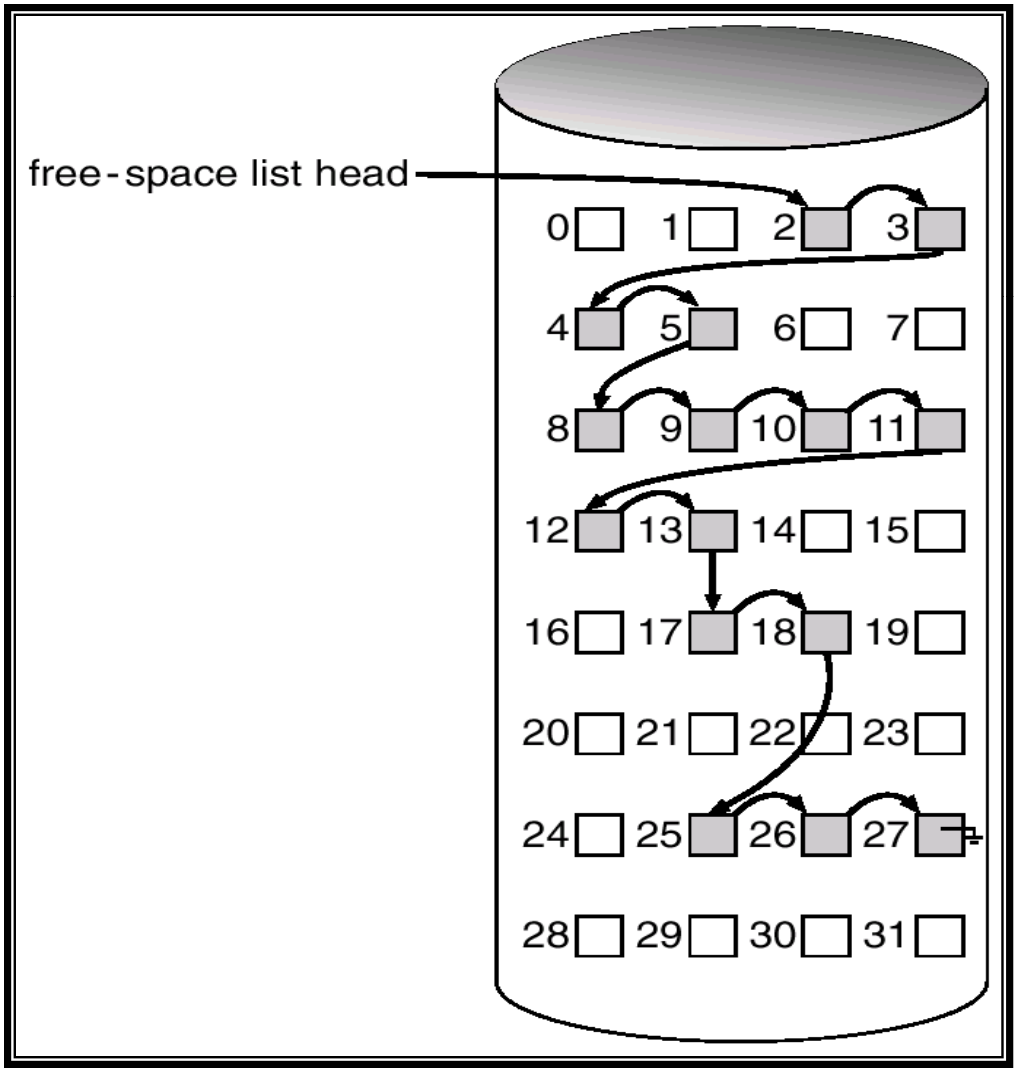
# Free Space Management : Bitmap

- Each block is represented by a bit. If the block is free, the bit is 1; if the block is allocated the bit is 0.
- A disk with n blocks requires a bitmap with n bits.
- *Eg: consider a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18,25,26,27 are free and rest are allocated.*
- *The free space bitmap would be 00111100111111000110000011*
- ***Advantages:***
  - Simple and efficient in finding first free blocks or n consecutive free blocks
- ***Problem:***
  - Inefficient unless the entire bitmap is kept in the main memory
  - Keeping in main memory is possible only for small disk, when disk is large the bitmap would be large

Er. Rajan Karmacharya

# Free Space Management : Linked List

- Link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- The first block contains a pointer to the next free block
- The previous example can be represented in liked list as




- ***Advantage***
  - Only one block is kept in memory
- ***Problem***
  - Not efficient to traverse list, it must read each block

**File Sharing**

In a multiuser system, there is almost always a requirement for files to be shared among a number of users. Two issues arise: access rights and the management of simultaneous access.

**Access Rights**
The file system should provide a flexible tool for allowing extensive file sharing among users. The file system should provide a number of options so that the way in which a particular file is accessed can be controlled. Typically, users or groups of users are granted certain access rights to a file. A wide range of access rights have been used. The following list is representative of access rights that can be assigned to a particular file

**None**: **The user may not even learn of the existence of the file**, much less access it. To enforce this restriction, the user would not be allowed to read the user directory that includes this file.

**Knowledge**: **The user can determine that the file exists and who its owner is**. The user is then able to petition (ask) the owner for additional access rights.

**Execution**: **The user can load and execute a program but cannot copy it**.

**Reading** : **The user can read the file for any purpose, including copying and execution.**

**Appending** : **The user can add data to the file, often only at the end** but **can't modify or delete** any of the file's contents. This right is useful in collecting data from a number of sources.

**Updating**: **The user can modify, delete and add to the file's data.** This normally includes writing the file initially, rewriting it completely or in part and removing all or a portion of the data.

**Changing Protection**: **The user can change the access rights granted to other users**. Typically, this right is held only by the owner and may extend this right to others.

**Deletion**: **The user can delete the file** from the file system.

These rights can be considered to constitute a hierarchy, with each right implying those that precede it.

Access can be provided to different classes of users
**Specific Users**: **Individual users who are designated by user ID**.
**User Groups**: **A set of users who are not individually defined**. The system must have some way of keeping track of the membership of user groups.
**All**: **All users who have access to this system**. These are public files.

**Simultaneous Access**
When access is granted to append or update a file to more than one user, the operating system or file management system must enforce discipline.
**A brute-force approach is to allow a user to lock the entire file when it is to be updated**. A finer grain of control is to lock individual records during update.
**Issues of mutual exclusion and deadlock** must be addressed in designing the access capability.

**File System Reliability**

**Major problem in maintaining a file system**

- **To protect it from being corrupted by both systems failure and errors in system's software.**
- **If a computer's file system is irrecoverably lost, due to hardware or software, restoring all the information will be difficult, time consuming and in many cases, impossible.**

**Solution**

The usual way of ensuring reliability is by **making duplicates copies of files.**

**Backups – a duplicate copy of the files used incase the original damages**

Companies, who well understand the value of back up do it once a day, usually to a tape. Modern tapes hold 100 gigabytes of data and cost pennies per gigabyte.

Backup tapes are generally made to

- **Recover from disaster** – due to disk crash, fire, flood or other natural calamities
- **Recover from stupidity** – accidental removal of files that is required later (may be temporarily in Recycle Bin)

Making a backup takes a longer time and occupies a large amount of space--- doing this efficiently and conveniently in a major issue.

**Periodic dump**

- All files in the file system are copied to another device, usually a magnetic tape. **This is done regularly** and may be **done as often as weekly** if the files are highly volatile.
- Wasteful to back up files that have not changed since the last backup

**Incremental dump**

- **Make a complete dump periodically and make a daily dump of those files that get modified from the last dump.** This is faster to do and usually done more often.
- Makes recovery more complicated because first the most recent full dump is to be restored followed by all the incremental dumps in reverse order.

**Compression**

- With immense amount of data, compression seems to be feasible but with many compression algorithms, a single bad spot on the backup tape can make the entire data unreadable.

**Security Issue**

- Backup introduces non technical problems in an organization
- The best online security system in the world may be useless if the system administrator keeps all the backup tapes in his office and leaves it open and unguarded. So, Good Bye Security.

Two strategies can be used for dumping a disk to a tape: a **physical dump** or a **logical dump**.

**Physical Dump**

**A physical dump starts at block 0 of the disk, writes all the disk blocks onto the output tape in order, and stops when it has copied the last one.** Such a program is so simple that it can probably be made 100% bug free, sometime that can probably not be said about any other useful program.

**Advantages**

- Simplicity and great speed (basically, it can run at the speed of disk).

**Problem**

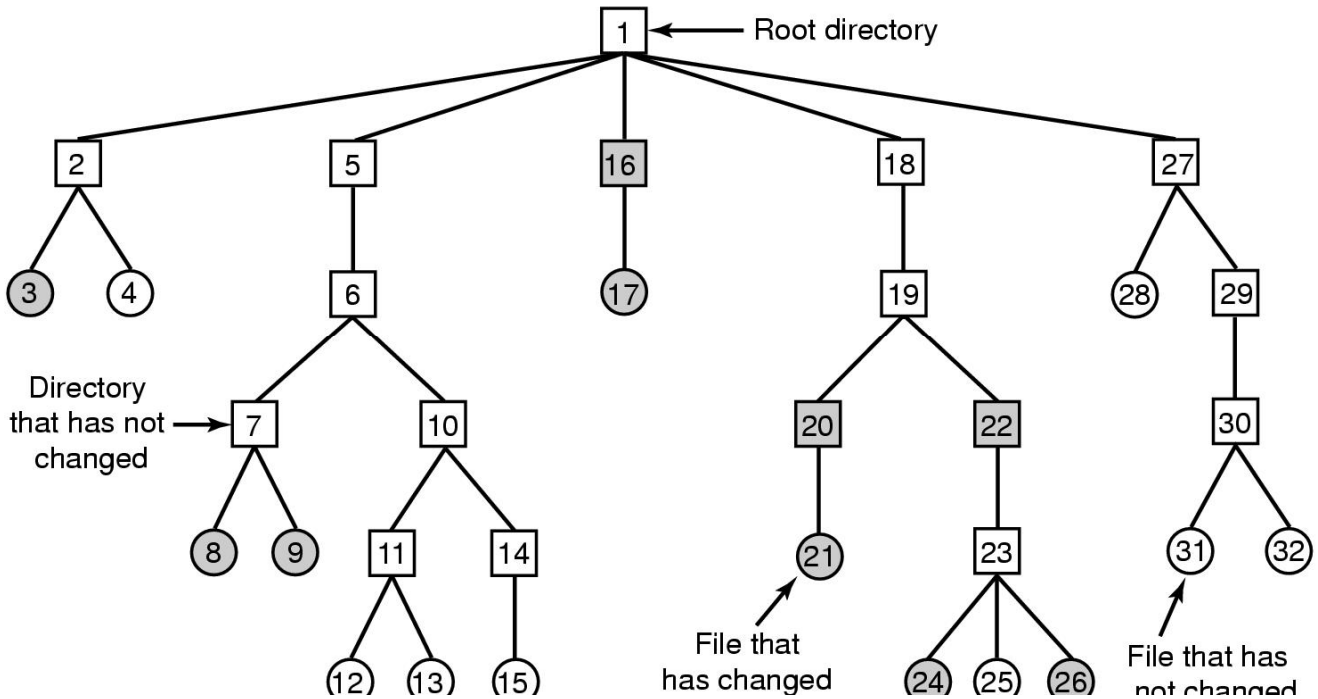
- There is no value in backing up unused disk blocks.
- Dumping bad blocks creates endless disk read errors during the dumping process.
- Unable to skip selected directories and makes incremental dumps.

**Logical Dump**

A logical dump **starts at one or more specified directories and recursively dumps all files and directories found there that have been changed since some given base date** (the last backup for an incremental dump for a full dump).

In a logical dump, the dump tape gets a series of carefully identified directories and files, which makes it easy to restore a specific file or directory upon request.

Logical dumping is the most common form. In the example below, a tree with directories (squares) and files (circles) is shown. The shaded items have been modified since the base date and thus need to be dumped. The unshaded ones do not need to be dumped.



**File System Consistency**

- Another important aspect of a file system is its consistency. **When a system crash occurs, it is possible that a modified block may not have been written yet.** The standard way of checking for consistency is to read all the files and the blocks allocated to the files of the file system and then compare this with the blocks in the free list.
- To deal with the problem of inconsistent file system, most computers have a utility program that checks file system consistency. UNIX has **fsck** and Windows has **scandisk**. These utilities run when the system is booted after a crash.

**File System Performance**

Access to disk is much slower than access to memory. Hence, many file systems have been designed with various optimizations to improve the performance

**Caching**

- The most common technique used to reduce disk access is the **block cache** or the **buffer cache**. A cache is a collection of blocks that logically belong to the disk but are being kept in the memory for performance reasons.
- Check all the requests to see if the needed block is in the cache. If it is, the read request can be satisfied without a disk access and if not, it is first read into the cache and then copied to wherever it is needed. Subsequent requests for the same block can be satisfied from the cache.

**Block Read Ahead**

- Another technique to improve file system performance is to **try to get blocks into the cache before they are needed to increase the hit rate.**
- In particular, many files are read sequentially. When the file system is asked to produce block **k** in a file, it does that, but when it is finished, it makes a sneaky check in the cache to see if block **k+1** is already there. If it is not, it schedules a read for block k+1 in the hope that when it is needed, it will have already arrived in the cache. At the very least, it will be on the way.
- Only helps the sequentially read files.(not in random access method)

**Reducing Disk Arm Motion**

- The disk arm motion can be reduced by putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder. (i.e., blocks lie in the outer most track, say track 1, of platter 1, platter 2 , platter 3 and so on)
- When the blocks lie in the same cylinder, the disk arm does not require moving and only the read write head moves. Hence data access is faster when the disk arm motion is reduced.