UNIT IV FILE SYSTEMS AND I/O SYSTEMS

Mass Storage system – Overview of Mass Storage Structure, Disk Structure, Disk Scheduling and Management, swap space management; File-System Interface – File concept, Access methods, Directory Structure, Directory organization, File system mounting, File Sharing and Protection; File System Implementation- File System Structure, Directory implementation, Allocation Methods, Free Space Management, Efficiency and Performance, Recovery; I/O Systems – I/O Hardware, Application I/O interface, Kernel I/O subsystem, Streams, Performance.

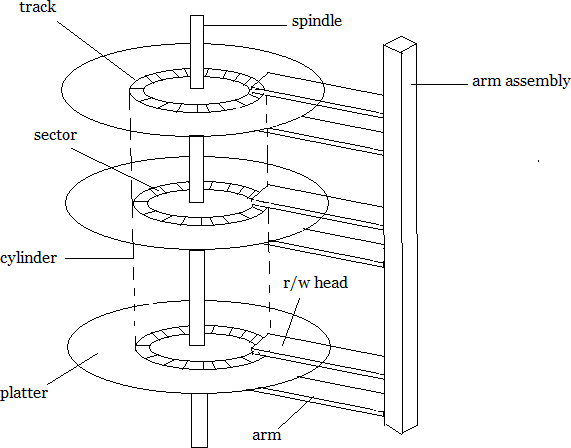
MASS STORAGE STRUCTURE

1. Overview of Mass Storage Structure Magnetic Disks
   * In modern computers, most of the secondary storage is in the form of magnetic disks.
   * A magnetic disk contains several platters. Each platter is divided into circular shaped tracks.
   * The length of the tracks near the centre is less than the length of the tracks farther from the centre.
   * Each track is further divided into sectors.
   * Tracks of the same distance from centre form a cylinder.
   * A read-write head is used to read data from a sector of the magnetic disk.
   * The speed of the disk is measured as two parts:

**Transfer rate**: This is the rate at which the data moves from disk to the computer.

**Random access time**: It is the sum of the seek time and rotational latency.

**Seek time** is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.



Solid-State Disks

SD is non-volatile memory that is used like a hard drive. SSDs have the same characteristics as traditional hard disks but can be more reliable because they have no

moving parts and faster because they have no seek time or latency. In addition, they consume less power.

SSDs have less capacity than the larger hard disks, and may have shorter life spans. use for SSDs is in storage arrays, where they hold file- system metadata that require high performance. Some SSDs are designed to connect directly to the system bus. **Magnetic Tapes**

Magnetic tape was used as an early secondary-storage medium. Although it is relatively permanent and can hold large quantities of data, its access time is slow compared with that of main memory and magnetic disk.

In addition, random access to magnetic tape is about a thousand times slower than random access to magnetic disk, so tapes are not very useful for secondary storage.

1. Disk Structure

In Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer. n The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.

In Sector 0 is the first sector of the first track on the outermost cylinder.

In Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

1. Disk Scheduling

The operating system is responsible for using hardware efficiently.

For the disk drives, this means having a fast access time & disk bandwidth.

Access time has two major components:

Seek time is the time for the disk to move the heads to the cylinder containing the desired sector

Rotational latency time waiting for the disk to rotate the desired sector to the disk head

We like to minimize seek time.

 Disk bandwidth is the total number of bytes transferred divided by the total time between the first request for service and the completion of the last transfer.

Several algorithms exist to schedule the servicing of disk I/O requests.

**We illustrate them with a Request Queue (cylinder range 0-199): 98, 183, 37, 122, 14, 124, 65, 67**

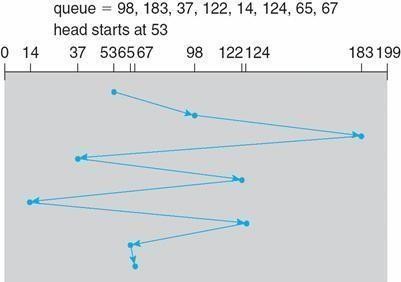
**Head pointer: cylinder 53**

1. **First Come First Serve**

This algorithm performs requests in the same order asked by the system. Let's take an example where the queue has the following requests with cylinder numbers as follows:

98, 183, 37, 122, 14, 124, 65, 67

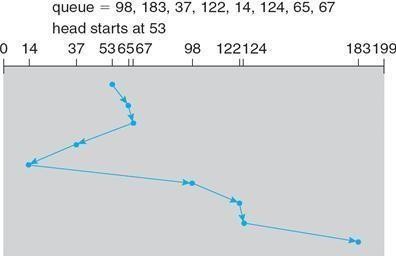
Illustration shows total head movement of 640 cylinders



1. SSTF (Shortest Seek Time First)

Selects the request with the minimum seek time from the current head position.

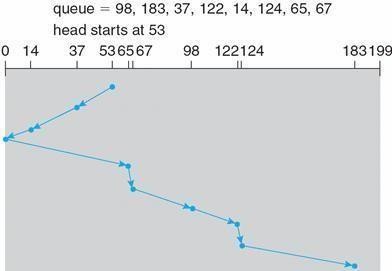
SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Illustration shows total head movement of 236 cylinders.

1. SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

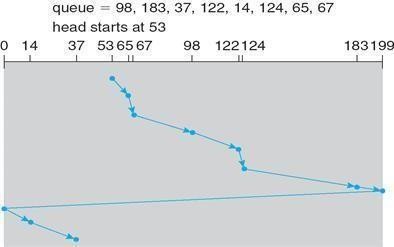
SCAN algorithm sometimes called the elevator algorithm. Illustration shows total head movement of 208 cylinders



1. C-SCAN

Provides a more uniform wait time than SCAN. The head moves from one end of the disk to the other, servicing requests as it goes When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

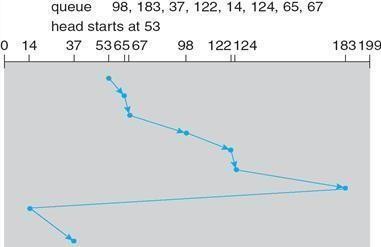
Treats the cylinders as a Circular list that wraps around from the last cylinder to the first one .



1. LOOK

 Version of C-SCAN

 Arm only goes as far as the **last request** in each direction,then reverses direction immediately, without first going all the way to the end of the disk.



1. Disk Management

The operating system is responsible for disk initialization, booting from disk, and bad-block recovery.

Disk Formatting

A new magnetic disk must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting, or physical formatting**. Low- level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytesin size), and a trailer.

The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC).**

This formatting enables the manufacturer to 1. Test the disk and 2. To initialize the mapping from logical block numbers

To use a disk to hold files, the operating system still needs to record its own data structures on the disk.

It does so in two steps.

1. The first step is **Partition** the disk into one or more groups of cylinders.

Among the partitions,one partition can hold a copy of the OS‘s executable code, while another holds user files.

1. The second step is **logical formatting**. The operating system stores the initial file-system data structures onto the disk. These datastructures may include maps of free and allocated space and an initial empty directory.

Boot Block

For a computer to start running-for instance, when it is powered up or rebooted- it needs to have an initial program to run. This initial program is called bootstrap program & it should be simple.

It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.

The bootstrap is stored in read-only memory (ROM). This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And, since ROM is read only, it cannot be infected by a computer virus.

The full bootstrap program is stored in the “**boot blocks**” at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk. The work of boot block as follows

1. Finds the operating system kernel on disk,
2. Loads that kernel into memory, and
3. Jumps to an initial address to begin the operating-system execution.

The full **bootstrap program** is stored in a partition called the boot blocks, at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.

**Bootstrap loader -** load the entire operating system from a non-fixed location on disk, and to start the operating system running.

Bad Blocks

The disk with defected sector is called as bad block. Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: “Handled manually‖

If blocks go bad during normal operation, a **special program** must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

Method 2: “sector sparing or forwarding”

The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

A typical bad-sector transaction might be as follows:

* The operating system tries to read logical block 87.
* The controller calculates the ECC and finds that the sector is bad.
* It reports this finding to the operating system.
* The next time that the system is rebooted, a special command is run to tell the

controller to replace the bad sector with a spare.

* After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

Method 3: “sector slipping”

For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

1. Swap-Space Management

Swap-space — virtual memory uses disk space as an extension of main memory.

Main goal for the design and implementation of swap space is to provide the best throughput for VM system

1. Swap-space use

Swapping –use swap space to hold entire process image Paging –store pages that have been pushed out of memory

Some OS may support multiple swap-space

–Put on separate disks to balance the load

Better to overestimate than underestimate

–If out of swap-space, some processes must be aborted or system crashed

1. Swap-Space Location

Swap-space can be carved out of the normal file system, or in a separate disk partition

A large file within the file system: simple but inefficient

–Navigating the directory structure and the disk-allocation data structure takes time and potentially extra disk accesses

–External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image

–Improvement

Caching block location information in main memory

* Contiguous allocation for the swap file

But, the cost of traversing FS data structure still remains

In a separate partition: raw partition

–Create a swap space during disk partitioning

–A separate swap-space storage manager is used to allocate and de-allocate blocks

–Use algorithms optimized for speed, rather than storage efficiency

–Internal fragment may increase Linux supports both approaches

Swap-space Management: Example

Solaris 1

–Text-segment pages are brought in from the file system and are thrown away if selected for paged out

More efficient to re-read from FS than write it to the swap space

-Swap space: only used as a backing store for pages of anonymous memory

Stack, heap, and uninitialized data

Solaris 2

–Allocates swap space only when a page is forced out of physical memory

Not when the virtual memory page is first created.

**FILE SYSTEM INTERFACE**

### File Concepts

A file is a named collection of related information that is recorded on secondary storage. From user’s perspective a , a file is the smallest

allotment of that logical secondary storage; unless they are within a file.

Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary.

In general, a file is a sequence of bits, bytes, lines, or records, the meaning ofwhich is defined by the file’s creator and user.

A **text file** is a sequence of characters organized into lines (and possibly pages). An **executable** file is a series of code sections that the loader can bring into memory and execute.

1. File Attributes

The information about all files is kept in the directory structure, a directory entry consists of the file’s name and its unique identifier. The identifier in turn locates the other file attributes.

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

1. File Operations

The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

**Creating a file** - First, space in the file system must be found for the file, Second, an entry for the new file must be made in the directory.

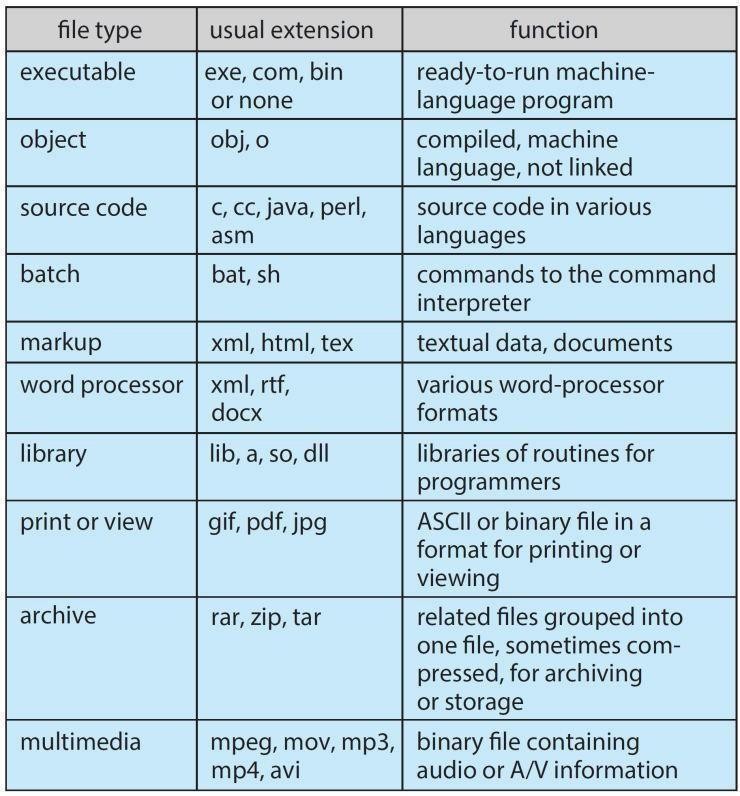
**Writing a file** - System call specifying both the name of the file and the information to be written to the file.

**Reading a file -** we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.

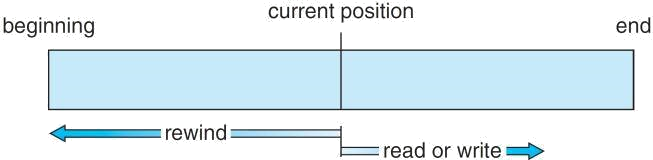
**Repositioning within a file -** The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.

**Deleting a file -** search the directory for the named file. Having found the associated directory entry, we release all file space

**Truncating a file -** this function allows all attributes to remain unchanged— except for file length—but lets the file be reset to length zero and its file space released.

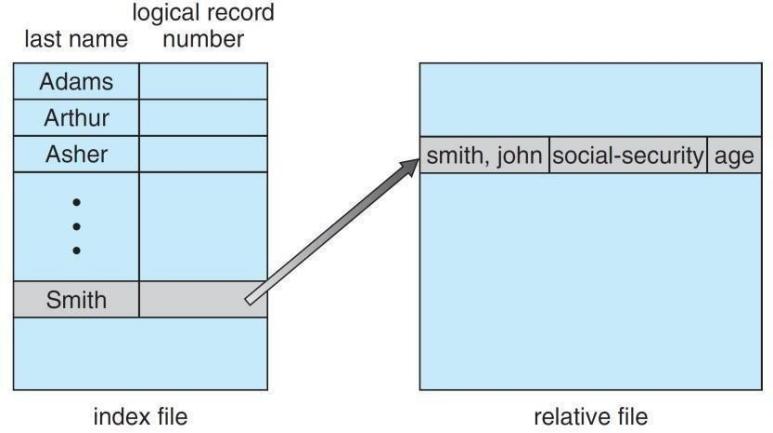
1. **File Types**

### Access Methods

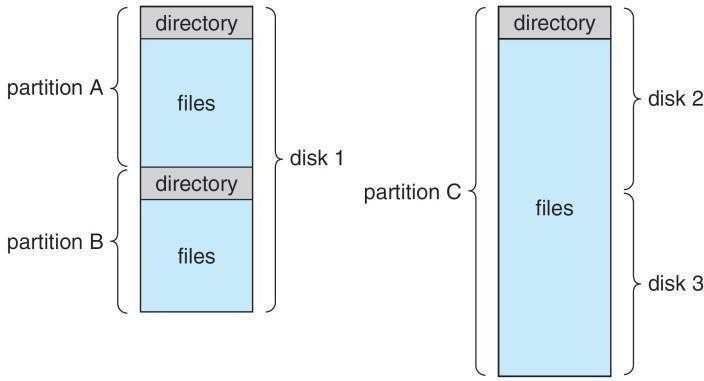
* 1. Sequential Access
     + Data is accessed one record right after another is an order.
     + Read command cause a pointer to be moved ahead by one.
     + Write command allocate space for the record and move the pointer to the new End Of File.
     + Such a method is reasonable for tape.
  2. Direct Access
     + This method is useful for disks.
     + The file is viewed as a numbered sequence of blocks or records.
     + There are no restrictions on which blocks are read/written, it can be dobe in any order.
     + User now says "read n" rather than "read next".
     + "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

**As a simple example**, on an **airline – reservation system**, we might store all the information about a particular flight (for example, flight 713) in the block identified by the flight number.

Thus, the number of available seats for flight 713 is stored in block 713 of the reservation file. To store information about a larger set, such as people, we might compute a hash function on the people’s names, or search a small in-memory index to determine a block to read and search.

* 1. Indexed Access
     + If a file can be sorted on any of the filed then an index can be assigned to a group of certain records.
     + However, A particular record can be accessed by its index.
     + The index is nothing but the address of a record in the file.
     + In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.

1. Directory Structure
   * Directory can be defined as the listing of the related files on the disk.
   * The directory may store some or the entire file attributes.
   * 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.



***Operations that are to be performed on a directory***

**Search for a file**. We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names

may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.

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

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

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

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

**Traverse the file system**. We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals.

Often, we do this by copying all files to magnetic tape. This technique provides a backup copy in case of system failure.

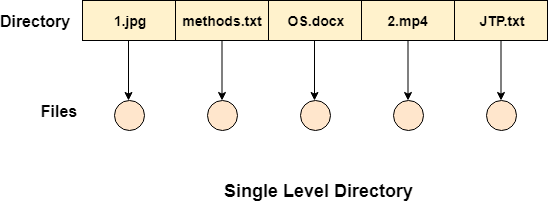
In addition, if a file is no longer in use, the file can be copied to tape and the disk space of that file released for reuse by another file.

Logical Structure (or) Level of Directory

* + - Single-level directory
    - Two-level directory
    - Tree-Structured directory
    - Acyclic Graph directory
    - General Graph directory

Single – Level Directory

* + - * The simplest method is to have one big list of all the files on the disk.
      * The entire system will contain only one directory which is supposed to mention all the files present in the file system.
      * The directory contains one entry per each file present on the file system.

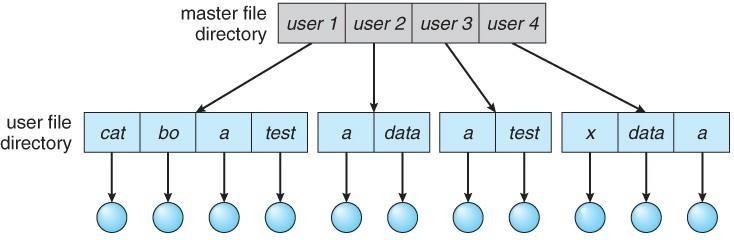


Disadvantages

* + - * 1. We cannot have two files with the same name.
        2. The directory may be very big therefore searching for a file may take so much time.
        3. Protection cannot be implemented for multiple users.
        4. There are no ways to group same kind of files.

Two Level Directory

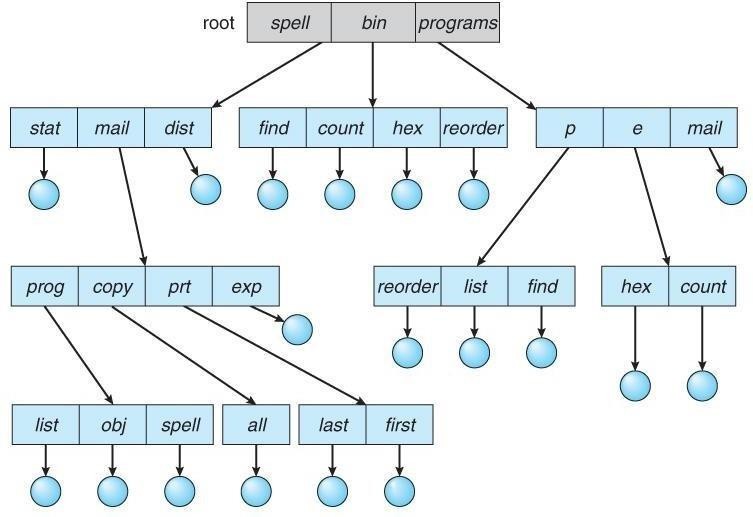
* In two level directory systems, we can create a separate directory for each user.
* There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file.
* The system doesn't let a user to enter in the other user's directory without permission.



Characteristics of two level directory system

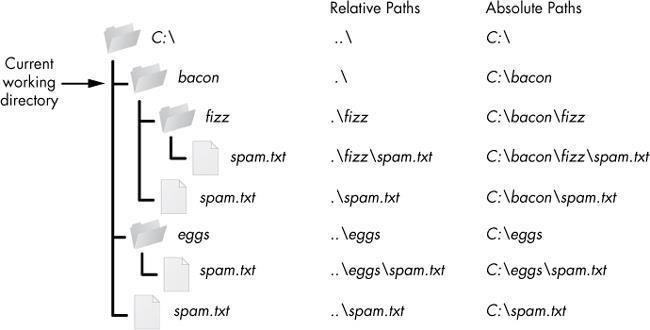
* 1. Each files has a path name as /User-name/directory-name/
  2. Different users can have the same file name.
  3. Searching becomes more efficient as only one user's list needs to be traversed.

Tree Structured Directory

* Tree structured directory system overcomes the drawbacks of two level directory system.
* The similar kind of files can now be grouped in one directory.
* Each user has its own directory and it cannot enter in the other user's directory.
* Searching is more efficient in this directory structure

A file can be accessed by two types of path, either  1.Relative or 2. Absolute.

1. **Absolute path** is the path of the file with respect to the root directory of the system.
2. **Relative path** is the path with respect to the current working directory of the system



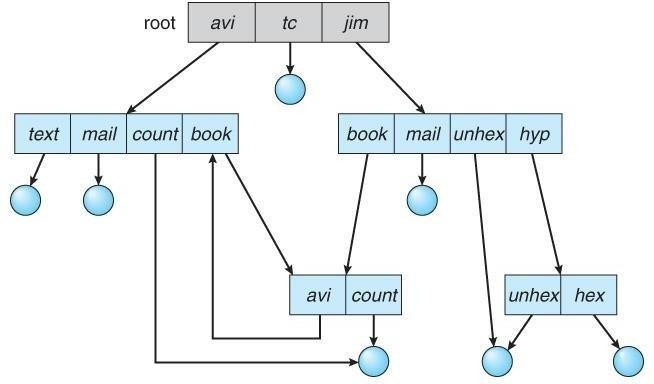
Acyclic-Graph Structured Directories

* + When the **same files need to be accessed** in **more than one place** in the directory structure it can be useful to provide an acyclic-graph structure.
  + In this system two or more directory entry can point to the same file or sub directory. That file or sub directory is shared between the two directory entries.

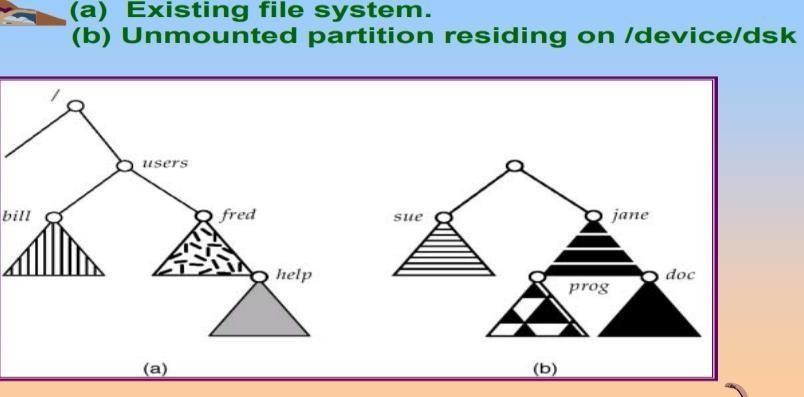
It provides two types of ***links*** for implementing the acyclic-graph structure **Soft link**, the file just gets deleted and we are left with a dangling pointer.

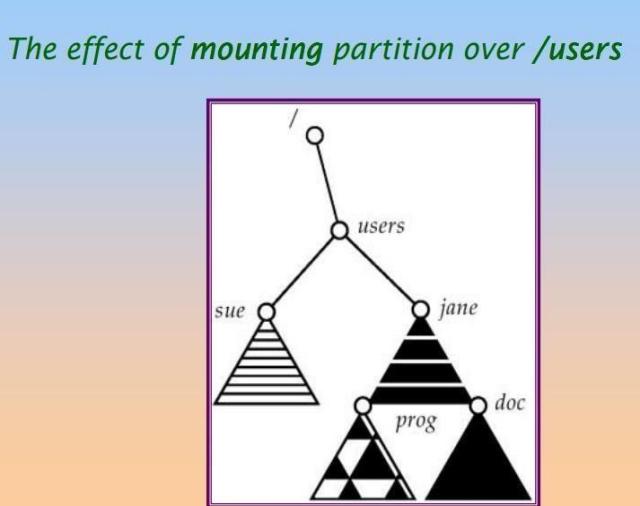
**Hard link**, the actual file will be deleted only if all the references to it gets deleted.

General Graph Directory

* + - 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 total size or space that have been taken by the files and directories.

1. File System Mounting
   * Before you can access the files on a file system, you need to mount the file system.
   * Mounting a file system attaches that file system to a directory (mount point) and makes it available to the system.
   * The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.
   * When you mount a file system, any files or directories in the underlying mount point directory are unavailable as long as the file system is mounted.
   * These files are not permanently affected by the mounting process, and they become available again when the file system is unmounted.
   * However, mount directories are typically empty, because you usually do not want to obscure existing files.





1. **File Sharing**
   * File sharing is the accessing or sharing of files by one or more users.
   * File sharing is performed on computer networks as an easy and quick way to transmit data.

For example, a user may share an instruction document on his computer that is connected to a corporate network allowing all other employees to access and read that document.

1. Multiple Users
   * On a multi-user system, more information needs to be stored for each file:The owner ( user ) who owns the file, and who can control its access.
   * The group of other user IDs that may have some special access to the file.
   * What access rights are afforded to the owner ( User ), the Group, and to the rest of the world.
2. Remote File Systems

The advent of the Internet introduces issues for accessing files stored on remote computers

* + The original method was ftp, allowing individual files to be transported across systems as needed.
  + The Client-Server Model(the system which physically owns the files acts as a *server*, and the system which mounts them is the *client.)*
  + Distributed Information Systems service that runs on a single central location.
  + Failure Modes  When a local disk file is unavailable, the result is generally known immediately, and is generally non-recoverable. The only reasonable response is for the response to fail. Remote access systems allow for blocking or delayed response.

1. Consistency Semantics

*Consistency Semantics* deals with the consistency between the views of shared files on a networked system. When one user changes the file, when do other users see the changes?

1. UNIX Semantics

 Writes to an open file are immediately visible to any other user who has the file open.

1. Session Semantics

AFS uses the following semantics:

Writes to an open file are not immediately visible to other users.

When a file is closed, any changes made become available only to users who open the file at a later time.

1. Immutable-Shared-Files Semantics

 when a file is declared as *shared* by its creator, it becomes immutable and the name cannot be re-used for any other

resource. Hence it becomes read-only, and shared access is simple.

1. **File Protection**
   * Files must be kept safe for reliability ( against accidental damage ), and protection( against deliberate malicious access. ) The former is usually managed with backup copies. This section discusses the latter.
   * One simple protection scheme is to remove all access to a file. However this makes the file unusable, so some sort of controlled access must be arranged.

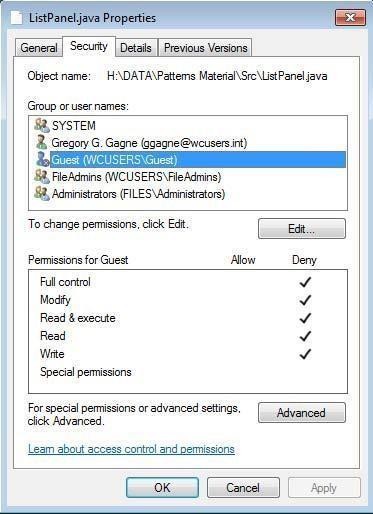
Types of Access

* 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

* One approach is to have complicated ***Access Control Lists, ACL,*** which specify exactly what access is allowed or denied for specific users or groups.
  + The AFS uses this system for distributed access.
  + Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown. ( AFS allows some wild cards, so for example all users on a certain remote system may be trusted, or a given username may be trusted when accessing from any remote system. )
* UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others. ( See "man chmod" for full details. ) The RWX bits control the following privileges for ordinary files and directories:

|  |  |  |
| --- | --- | --- |
| **bit** | **Files** | **Directories** |
| **R** | **Read ( view ) file contents.** | **Read directory contents. Required to get a listing of the directory.** |
| **W** | **Write**  **( change ) file contents.** | **Change directory contents. Required to create or delete files.** |
| **X** | **Execute file contents as a program.** | **Access detailed directory information. Required toget a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access.** |



FILE SYSTEM IMPLEMENTATION

1. File System Structure
   * File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way.
   * A file System must be able to store the file, locate the file and retrieve the file.
   * Most of the Operating Systems use layering approach for every task including file systems.
   * Every layer of the file system is responsible for some activities.

Logical file system

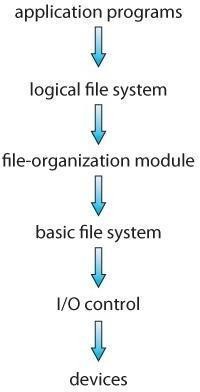
✄ **Provides** users the view of a contiguous sequence of words, bytes stored somewhere.

✄ Uses a directory structure, symbolic name

✄Provides protection and security

✄OS/user interface

☎E.g., to create a new file the API provides a call that calls the logical file system



The file organization module

✄Knows about files and their logical blocks (say 1,..N)

✄Files are organized in blocks of 32 bytes to 4K bytes

✄Translates logical blocks into physical

✄Knows location of file, file allocation type

✄Includes a free space manages that tracks unallocated blocks

Basic file system

✄Issues commands to the device driver (layer of software that directly controls disk hardware) to read and write physical blocks on the disk,

✄Each physical block identified by a disk address (e.g., drive 2, cylinder 34, track 2, sector 11)

IO control

✄The lowest level in the file system

✄Consists of device drivers and interrupt handlers to transfer information betweenthe memory and the disk

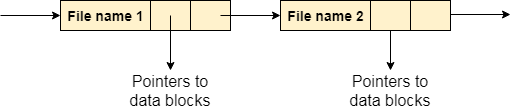
✄A device driver translates commands such as “get me block 111” into hardwarespecific ISA used by hardware controller. This is accomplished by writing specificbits into IO registers

##### Directory Implementation

* Directories need to be fast to search, insert, and delete, with a minimum of wasted disk space.

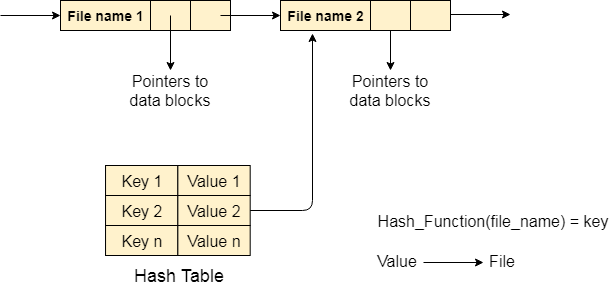
Linear List

* + A linear list is the simplest and easiest directory structure to set up, but it does have some drawbacks.
  + Finding a file ( or verifying one does not already exist upon creation ) requires a linear search.
  + Deletions can be done by moving all entries, flagging an entry as deleted, or by moving the last entry into the newly vacant position.
  + Sorting the list makes searches faster, at the expense of more complex insertions and deletions.
  + A linked list makes insertions and deletions into a sorted list easier, with overhead for the links.
  + More complex data structures, such as B-trees, could also be considered.



Hash Table

* + A hash table can also be used to speed up searches.
  + Hash tables are generally implemented in addition to a linear or other structure.
  + A key-value pair for each file in the directory gets generated and stored in the hash table.
  + The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.
  + **Searching** Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



1. Allocation Methods
   * There are various

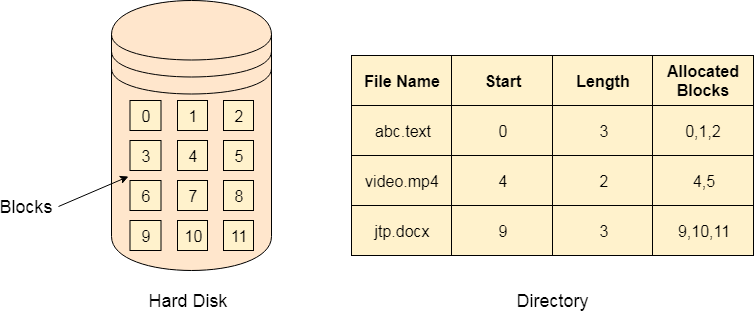
methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system.

* + Allocation method

provides a way in which the disk will be utilized and the files will be accessed.

Contiguous Allocation

* If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.
* In the image shown below, there are three files in the directory.
* The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



* + All these algorithms suffer from the problem of external fragmentation.
  + As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks.
  + It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.
  + This scheme effectively **compacts** all free space into one contiguous space, solving the fragmentation problem.

Advantages

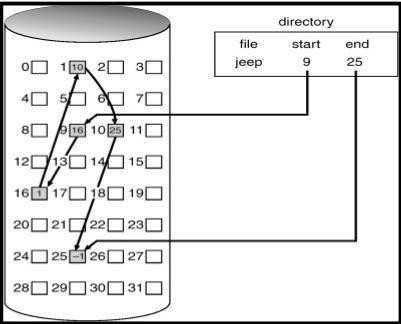
* + It is simple to implement.
  + We will get Excellent read performance.
  + Supports Random Access into files.

Disadvantages

* + The disk will become fragmented.
  + It may be difficult to have a file grow.

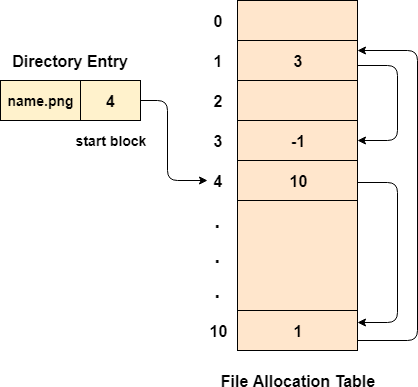
Linked List Allocation

* Linked List allocation solves all problems of contiguous allocation.
* In linked list allocation, each file is considered as the linked list of disk blocks.
* However, the disks blocks allocated to a particular file need not to be contiguous on the disk.
* Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.
* For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25 (See Figure). Each block contains a pointerto the next block. These pointers are not made available to the user. Thus, if each block is512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes.



File Allocation Table

* The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks.
* File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number.
* File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block.



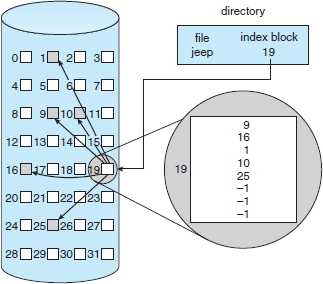
Advantages

* There is no external fragmentation with linked allocation.
* Any free block can be utilized in order to satisfy the file block requests.
* File can continue to grow as long as the free blocks are available.
* Directory entry will only contain the starting block address.

Disadvantages

* Random Access is not provided.
* Pointers require some space in the disk blocks.
* Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
* Need to traverse each block.

###### Indexed Allocation

* Indexed allocation solves this problem by bringing all the pointers together into one location: **the 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.
* To find and read the *ith* block, we use the pointer in the *ith* index-block entry. This scheme is similar to the paging scheme.
* When the file is created, all pointers in the index block are set to null.
* When the *ith* block is first written, a block is obtained from the free-space manager, and its address is put in the *i*th index-block entry.
* Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.

Advantages

1. Supports direct access
2. A bad data block causes the lost of only that block.

Disadvantages

1. A bad index block could cause the lost of entire file.
2. Size of a file depends upon the number of pointers, a index block can hold.
3. Having an index block for a small file is totally wastage.

4. More pointer overhead

1. Free Space Management

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks – those not allocated to some file or directory.

To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

* 1. Bit Vector

The free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit.

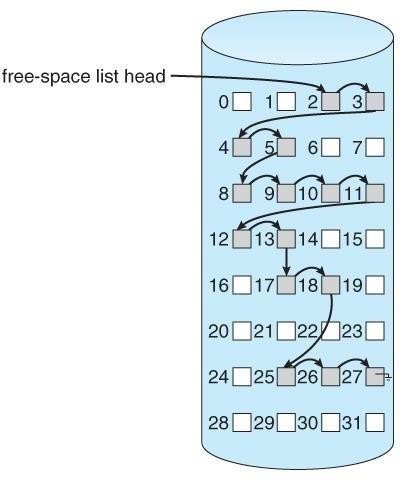
If the block is free, the bit is 1; if the block is allocated, the bit is 0. For example, Consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be 001111001111110001100000011100000 …

The main advantage **of** this approach is its relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

* 1. Linked List

Another approach to free-space management is to 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. This first block contains a pointer to the next free disk block, and so on.

In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on. However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time. The FAT method incorporates free-block accounting data structure. No separate method is needed.

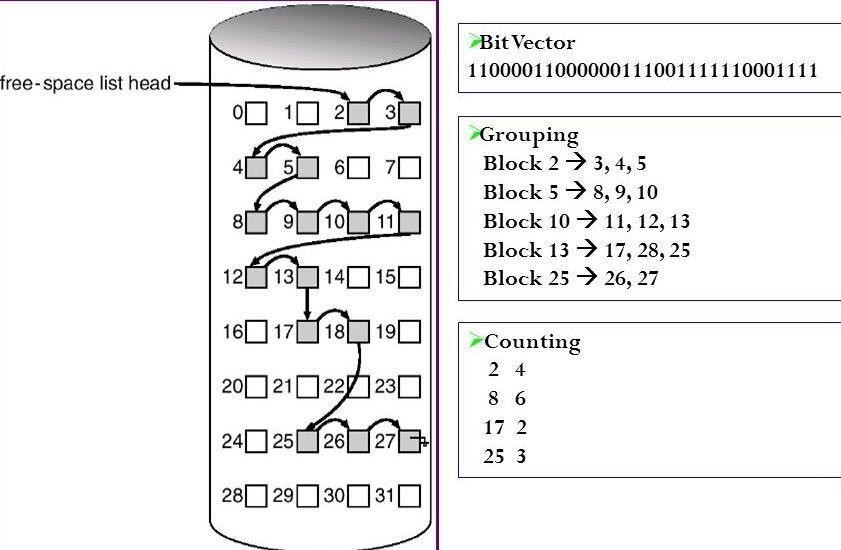


* 1. Grouping

A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The first n-1 of these blocks are actually free. The last block contains the addresses of another n free blocks, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

* 1. Counting

We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than1.



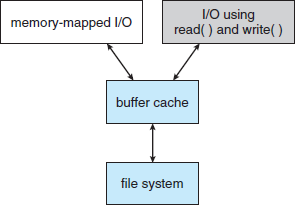
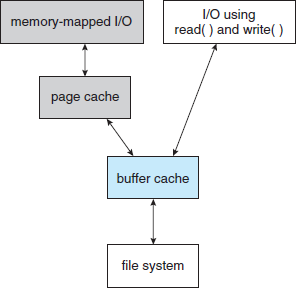
* 1. Efficiency and Performance

**Efficiency**

* + - The efficient use of disk space depends heavily on the disk-allocation and directory algorithms in use.
    - Let’s reconsider the clustering scheme, which improves file-seek and file-transfer performance at the cost of internal fragmentation. To reduce this fragmentation, BSD UNIX varies the cluster size as a file grows. Large clusters are used where they can be filled, and small clusters are used for small files and the last cluster of a file. This
    - The types of data normally kept in a file’s directory (or inode) entry also require consideration. Commonly, a “last write date” is recorded to supply information to the user and to determine whether the file needs to be backed up. Some systems also keep a “last access date,” so that a user can determine when the file was last read.
    - The result of keeping this information is that, whenever the file is read, a field in the directory structure must be written to. That means the block must be read into memory, a section changed, and the block written back out to disk, because operations on disks occur only in block (or cluster) chunks. So any time a file is opened for reading, its directory entry must be read and written as well.
    - Generally, every data item associated with a file needs to be considered for its effect on efficiency and performance.

Performance

* + - Some systems maintain a separate section of main memory for a **buffer cache**, where blocks are kept under the assumption that they will be used again shortly. Other systems cache file data using a **page cache**.
    - The **page cache** uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks.
    - Caching file data using virtual addresses is far more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the filesystem.
    - Several systems—including Solaris, Linux, and Windows —use page caching to cache both process pages and file data. This is known as **unified virtual memory**.



* + - * The two alternatives for opening and accessing a file. One approach is to use memory mapping the second is to use the standard system calls **read()** and **write()**.
    - Here, the read() and write() system calls go through the buffer cache.
    - The memory-mapping call, however, requires using two caches— the **page cache** and the **buffer cache.**
    - A memory mapping proceeds by reading in disk blocks from the file system and storing them in the buffer cache. Because the virtual memory system does not interface with the buffer cache, the contents of the file in the buffer cache must be copied into the page cache. This situation, known as **double caching**, requirescaching file-system data twice.
  1. Recovery

**Consistency Checking**

* + - The storing of certain data structures ( e.g. directories and inodes ) in memory and the caching of disk operations can speed up performance, but what happens in the result of a system crash? All volatile memory structures are lost, and the information stored on the hard drive may be left in an inconsistent state.
    - A Consistency Checker ( fsck in UNIX, chkdsk or scandisk in Windows ) is often run at boot time or mount time, particularly if a filesystem was not closed down properly. Some of the problems that these tools look for include:
* Disk blocks allocated to files and also listed on the free list.
* Disk blocks neither allocated to files nor on the free list.
* Disk blocks allocated to more than one file.
* The number of disk blocks allocated to a file inconsistent with the file's stated size.
* Properly allocated files / inodes which do not appear in any directory entry.
* Link counts for an inode not matching the number of references to that inode in the directory structure.
* Two or more identical file names in the same directory.
* Illegally linked directories, e.g. cyclical relationships where those are not allowed, or files/directories that are not accessible from the root of the directory tree.
* Consistency checkers will often collect questionable disk blocks into new files with names such as chk00001.dat. These files may contain valuable information that would otherwise be lost, but in most cases they can be safely deleted, ( returning those disk blocks to the free list. )

UNIX caches directory information for reads, but any changes that affect space allocation or metadata changes are written synchronously, before any of the corresponding data blocks are written to.

Log-Structured File Systems

* Log-based transaction-oriented ( a.k.a. journaling ) filesystems borrow techniques developed for databases, guaranteeing that any given transaction either completes successfully or can be rolled back to a safe state before the transaction commenced:
  + All metadata changes are written sequentially to a log.
  + A set of changes for performing a specific task ( e.g. moving a file

) is a transaction.

* + As changes are written to the log they are said to be committed, allowing the system to return to its work.
  + In the meantime, the changes from the log are carried out on the actual filesystem, and a pointer keeps track of which changes in the log have been completed and which have not yet been completed.
  + When all changes corresponding to a particular transaction have been completed, that transaction can be safely removed from the log.
  + At any given time, the log will contain information pertaining to uncompleted transactions only, e.g. actions that were committed but for which the entire transaction has not yet been completed.
  + From the log, the remaining transactions can be completed, or if the transaction was aborted, then the partially completed changes can be undone.

**Backup and Restore**

* In order to recover lost data in the event of a disk crash, it is important to conduct backups regularly.
* Files should be copied to some removable medium, such as magnetic tapes, CDs, DVDs, or external removable hard drives.
* A full backup copies every file on a file system.Incremental backups copy only files which have changed since some previous time.
* A combination of full and incremental backups can offer a compromise between full recoverability, the number and size of backup tapes needed, and the number of tapes that need to be used to do a full restore.
* A typical backup schedule may then be as follows:

Day 1. Copy to a backup medium all files from the disk. This is called a full backup.

Day 2. Copy to another medium all files changed since day 1. This is an incremental backup.

Day 3. Copy to another medium all files changed since day 2.

I/O SYSTEMS

1. **I/O Hardware**

Day N. Copy to another medium all files changed since day N−1. Then go back to day 1.

The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices. A device communicates with a computer system by sending signals over a cable or even through the air.

**Port:** The device communicates with the machine via a connection point (or port), for example, a serial port.

**Bus**: If one or more devices use a common set of wires, the connection is called a bus.

**Daisy chain**: Device ‗A ‘has a cable that plugs into device ‗B ‘, and device

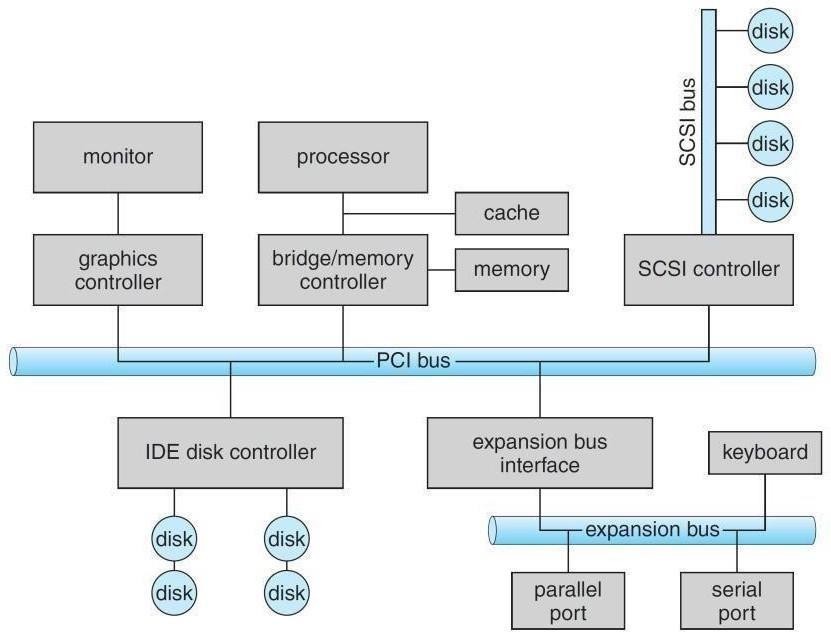
‗B ‘has a cable that plugs into device ‗C ‘, and device ‗C ‘plugs into a port on the computer, this arrangement is called a daisy chain. A daisy chain usually operates as a bus.

PC bus structure

A PCI bus that connects the processor-memory subsystem to the fast devices, and an expansion bus that connects relatively slow devices such as the keyboard and serial and parallel ports. In the upper- right portion of the figure, four disks are connected together on a SCSI bus plugged into a SCSI controller.

A **controller or host adapter** is a collection of electronics that can operate a port, a bus, or a device. A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port. By contrast, a SCSI bus controller is not simple.

Because the SCSI protocol is complex, the SCSI bus controller is often implemented as a separate circuit board. It typically contains a processor, microcode, and some private memory. Some devices have their own built- in controllers.



How can the processor give commands and data to a controller to accomplish an I/O transfer?

* Direct I/O instructions
* Memory-mapped I/O

###### Direct I/O instructions

Use special I/O instructions that specify the transfer of a byte or word to an I/O port address. The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register

###### Memory-mapped I/O

The device-control registers are mapped into the address space of the processor. The CPU executes I/O requests using the standard data-transfer instructions to read and write the device- control registers.

|  |  |
| --- | --- |
| **Status register** | Read by the host to indicate states such as whether the current command |
| has completed, whether a byte is available to be read from the data-in |
| register, and whether there has been a device error. |
| **Control register** | Written by the host to start a command or to change the mode of a device. |
| **data-in register** | Read by the host to get input |
| **data-out register** | Written by the host to send output |

* An I/O port typically consists of four registers: status, control, data-in, and data-out registers.

1. Polling

**Interaction between the host and a controller**

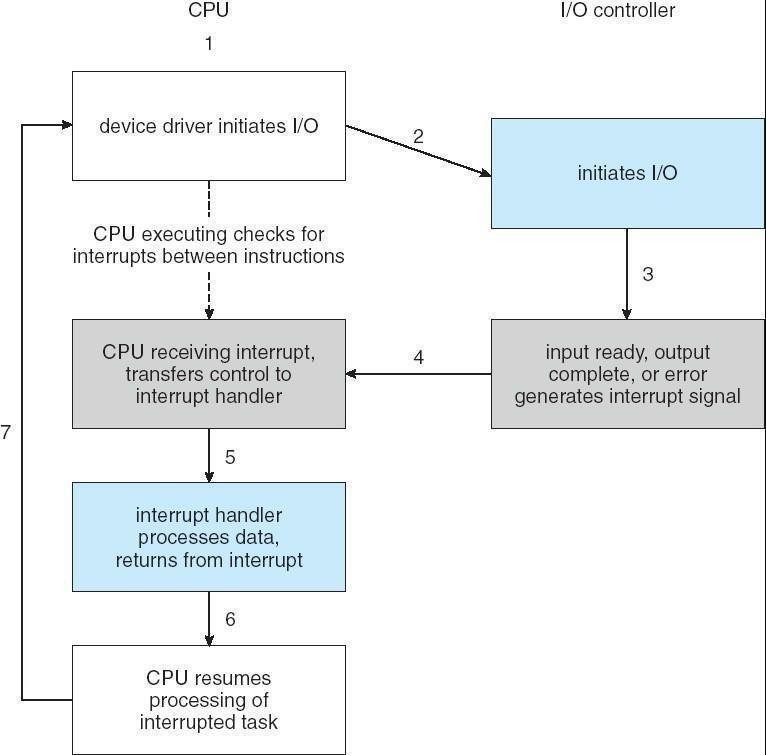
* + The controller sets the busy bit when it is busy working, and clears the busy bit when it is ready to accept the next command.
  + The host sets the command ready bit when a command is available for the controller to execute.

Coordination between the host & the controller is done by handshaking as follows:

1. The host repeatedly reads the busy bit until that bit becomes clear.
2. The host sets the write bit in the command register and writes a byte into the data-out register.
3. The host sets the command-ready bit.
4. When the controller notices that the command-ready bit is set, it sets the busy bit.
5. The controller reads the command register and sees the write command. It reads the data-out register to get the byte, and does the I/O to the device.
6. The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.
7. In step 1, the host is ―**busy-waiting or polling**‖: It is in a loop, reading the status register over and over until the busy bit becomes clear.
8. Interrupts

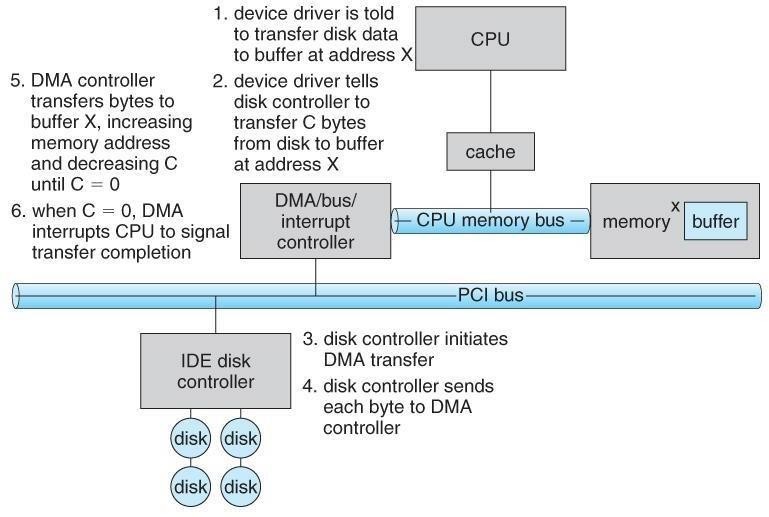
The CPU hardware has a wire called the ―interrupt-request line‖. The basic interrupt mechanism works as follows;

1. Device controller raises an interrupt by asserting a signal on the interrupt request line.
2. The CPU catches the interrupt and dispatches to the interrupt handler and
3. The handler clears the interrupt by servicing the device.
   * **Nonmaskable interrupt**: which is reserved for events such as unrecoverable memory errors?
   * **Maskable interrupt**: Used by device controllers to request service



1. Direct Memory Access (DMA)

In general it is tough for the CPU to do the large transfers between the memory buffer & disk; because it is already equipped with some other tasks ,then this will create overhead. So a special-purpose processor called a direct memory-access **(DMA)** controller is used.



1. Application I/O Interface

I/O system calls encapsulate device behaviours in generic classes. Device-driver layer hides differences among I/O controllers from kernel

Devices vary on many dimensions, as illustrated in

* + **Character-stream or block**. A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.
  + **Sequential or random access**. A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.
  + **Synchronous or asynchronous**. A synchronous device performs data transfers with predictable response times, in coordination with other aspects of the system. An asynchronous device exhibits irregular or unpredictable response times not

coordinated with other computer events.

* + **Sharable or dedicated**. A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.

**Speed of operation**. Device speeds range from a few bytes per second to a few gigabytes per second.

* + **Read–write, read only, or write only**. Some devices perform both input and output, but others support only one data transfer direction.

1. Block and Character Devices

**Block-device:** The block-device interface captures all the aspects necessary for accessing disk drives and other block-oriented devices. The device should understand the commands such as read () & write (), and if it is a random access device, it has a seek() command to specify which block to transfer next.

**Character Devices:** A keyboard is an example of a device that is accessed through a character stream interface. The basic system calls in this interface enable an application to get() or put() one character.

1. Network Devices

Because the performance and addressing characteristics of network I/O differ

significantly from those of disk I/O, most operating systems provide a network I/O interface that is different from the read0

-write() -seek() interface used for disks.

* + Windows NT provides one interface to the network interface card, and a second interface to the network protocols.
  + In UNIX, we find half-duplex pipes, full-duplex FIFOs, full- duplex STREAMS, message queues and sockets.

1. Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

* + Give the current time
  + Give the elapsed time
  + Set a timer to trigger operation X at time T

Programmable interval timer: The hardware to measure elapsed time and to trigger operations is called a programmable interval timer. It can be set to wait a certain amount of time and then to generate an interrupt. To generate periodic interrupts, it can be set todo this operation once or to repeat.

Uses of Programmable interval timer:

|  |  |
| --- | --- |
| Scheduler | To generate an interrupt that will pre-empt a process at the end of its |
| time slice. |
| Disk I/O subsystem | To invoke the flushing of dirty cache buffers to disk periodically |
| Network subsystem | To cancel operations those are proceeding too slowly because of |
| network congestion or failures. |

When the timer interrupts, the kernel signals the requester, and reloads the timer with the next earliest time.

**Counter**: The hardware clock is constructed from a high frequency counter.

In some computers, the value of this counter can be read from a device register, in which the counter can be considered to be a high-resolution clock.

1. ***Blocking and Non-blocking I/O (or) synchronous & asynchronous: Blocking I/O:*** When an application issues a blocking system call;
   * The execution of the application is suspended.
   * The application is moved from the operating system's run queue to a wait queue.
   * After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution, at which time it will receive the values returned by the system call.

**Non-blocking, I/O:** Some user-level processes need non-blocking

I/O***Examples:***

User interface that receives keyboard and mouse input while processing and displaying data on the screen.

Video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display.

1. Kernel I/O Subsystem

Kernels provide many services related to I/O.

* One way that the I/O subsystem improves the efficiency of the computer is by scheduling I/O operations.
* Another way is by using storage space in main memory or on disk, via techniques called buffering, caching, and spooling.

1. I/O Scheduling:

To determine a good order in which to execute the set of I/O requests. Uses:

* + It can improve overall system performance,
  + It can share device access fairly among processes, and
  + It can reduce the average waiting time for 1/0 to complete.

Implementation: OS developers implement scheduling by maintaining a

―queue of requests for each device.

* + When an application issues a blocking I/O system call,
  + The request is placed on the queue for that device.
  + The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

1. Buffering:

**Buffer**: A memory area that stores data while they are transferred between two devices or between a device and an application.

Reasons for buffering:

* + To cope with a speed mismatch between the producer and consumer of a data stream.
  + To adapt between devices that have different data-transfer sizes.
  + To support copy semantics for application I/O.

**Copy semantics** Suppose that an application has a buffer of data that it wishes to write to disk. It calls the write () system call, providing a pointer to the buffer and an integer specifying the number of bytes to write.

1. ***Caching***

A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original

**Cache vs buffer**: A buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere.

When the kernel receives a file I/O request,

1. The kernel first accesses the buffer cache to see whether that region of the file is already available in main memory.
2. If so, a physical disk I/O can be avoided or deferred. Also, disk writes are accumulated in the buffer cache for several seconds, so that large transfers are gathered to allow efficient write schedules.
3. ***Spooling and Device Reservation:***

Spool: A buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.

A printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together

The OS provides a control interface that enables users and system administrators ;

* + To display the queue,
  + To remove unwanted jobs before those jobs print,
  + To suspend printing while the printer is serviced, and so on.

Device reservation - provides exclusive access to a device

* + System calls for allocation and de-allocation
  + Watch out for deadlock

1. Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors. OS can recover from disk read, device unavailable, transient write failures Most return an error number or code when I/O request fails System error logs hold problem reports

STREAMS

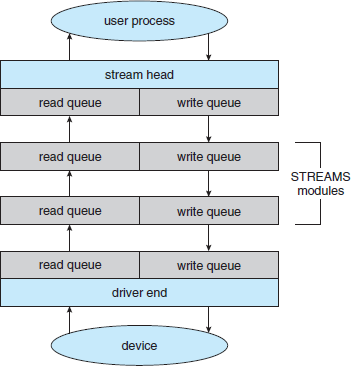
Stream is a full-duplex communication channel between a user-level process

and a device in Unix System V and beyond A STREAM consists of:

* + STREAM head interfaces with the user process
  + Driver end interfaces with the device
  + Zero or more STREAM modules between them.

Each module contains a read queue and a write queue. Message passing is used to communicate between queues. Modules provide the functionality of STREAMS processing and they are pushed onto a stream using the ioct () system call.

Flow control: Because messages are exchanged between queues in adjacent modules, a queue in one module may overflow an adjacent queue. To prevent this from occurring, a queue may support flow control.



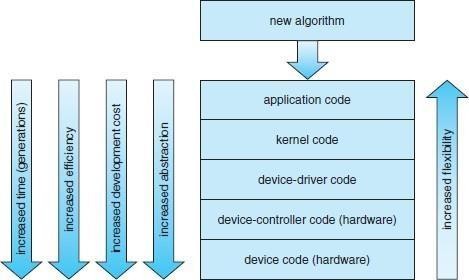
PERFORMANCE

I/O a major factor in system performance:

* + Heavy demands on CPU to execute device driver, kernel I/O code. So context switches occur due to interrupts.
  + Interrupt handling is a relatively expensive task: Each interrupt causes the system to perform a state change, to execute the interrupt handler & then to restore state
  + Network traffic especially stressful.
  + Systems use separate ―front-end processors” for terminal I/O, to reduce the interrupt burden on the main CPU.

We can employ several principles to improve the efficiency of I/O:

* + Reduce the number of context switches.
  + Reduce the number of times that data must be copied in memory while passing between device and application.
  + Reduce the frequency of interrupts by using large transfers, smart controllers & polling.
  + Increase concurrency by using DMA-knowledgeable controllers or channels to offload simple data copying from the CPU.
  + Move processing primitives into hardware, to allow their operation in device controllers concurrent with the CPU and bus operation.
  + Balance CPU, memory subsystem, bus, and I/O performance, because an overload in any one area will cause idleness in others.





* + 1. **An application-level implementation**: Implement experimental I/O algorithms at the application level, because application code is flexible, and application bugs are unlikely to cause system crashes.

It can be inefficient;

* + Because of the overhead of context switches and
  + Because the application cannot take advantage of internal kernel data structures and kernel functionality

1. **In-kernel implementation**: Re-implement application-level algorithm in the kernel. This can improve the performance, but the development effort is more challenging, because an operating-system kernel is a large, complex software system. Moreover, an in-kernel implementation must be thoroughly debugged to avoid data corruption and system crashes.
2. **A hardware implementation**: The highest performance may be obtained by a specialized implementation in hardware, either in the device or in the controller.
   * Difficult and expense of making further improvements or of fixing bugs, (-) Increased development time
   * Decreased flexibility.