UNIT IV STORAGE MANAGEMENT

Mass Storage System - Disk Structure - Disk Scheduling and 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; I/O Systems - I/O Hardware, Application I/O interface, Kernel I/O subsystem.

MASS STORAGE SYSTEM

Mass storage refers to various techniques and devices for storing large amounts of data.

The earliest storage devices were punched paper cards, which were used as early as 1804 to control silk-weaving looms.

Modern mass storage devices include all types of disk drives and tape drives.

Mass storage is distinct from memory, which refers to temporary storage areas within the computer. Unlike main memory, mass storage devices retain data even when the computer is turned off.

Examples of Mass Storage Devices (MSD)

Common types of mass storage include the following:

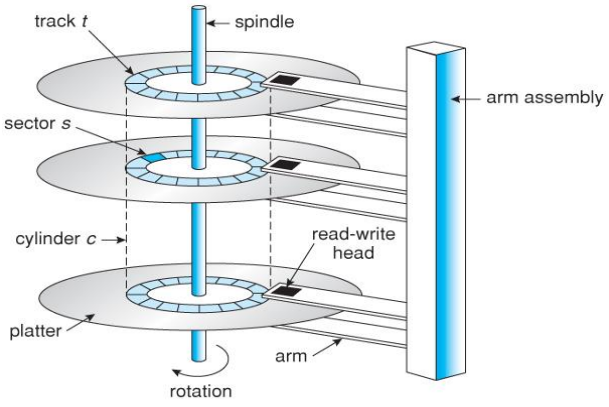
- solid-state drives (SSD)
- hard drives
- external hard drives
- optical drives
- tape drives
- RAID storage
- USB storage
- flash memory cards

Mass storage is sometimes called **auxiliary storage.**

OVERVIEW OF MASS-STORAGE STRUCTURE

Magnetic Disks

Each modern disk contains concentric tracks and each track is divided into multiple sectors. The disks are usually arranged as a one dimensional array of blocks, where blocks are the smallest storage unit. Blocks can also be called as sectors. For each surface of the disk, there is a read/write desk available. The same tracks on all the surfaces are known as a cylinder.



Moving-head disk mechanism

Solid-State Disks

- SSDs or solid state 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.
- But they consume less power and are more expensive per megabyte, have less capacity, and may have shorter life spans than hard disks, so their uses are somewhat limited.
- One use for SSDs is in storage arrays, where they hold file-system meta data that require high performance.
- SSDs are also used in some laptop computers to make them smaller, faster, and more energy-efficient.
- Because SSDs can be much faster than magnetic disk drives, standard bus interfaces can cause a major limit on throughput.
- Some SSDs are designed to connect directly to the system bus (PCI, for example).
- SSDs are changing other traditional aspects of computer design as well.
- Some systems use them as a direct replacement for disk drives, while others use them as a new cache tier, moving data between magnetic disks, SSDs, and memory to optimize performance.

Magnetic Tapes

- Magnetic tape was used as an early secondary-storage medium.
- Tapes are used mainly for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another.
- A tape is kept in a spool and is wound or rewound past a read-write head.
- Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
- Tape capacities vary greatly, depending on the particular kind of tape drive, with current capacities exceeding several terabytes.
- Some tapes have built-in compressions that can more than double the effective storage.

DISK SCHEDULING

Some important terms related to disk scheduling:

Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

Transfer Time

It is the time taken to transfer the data.

Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

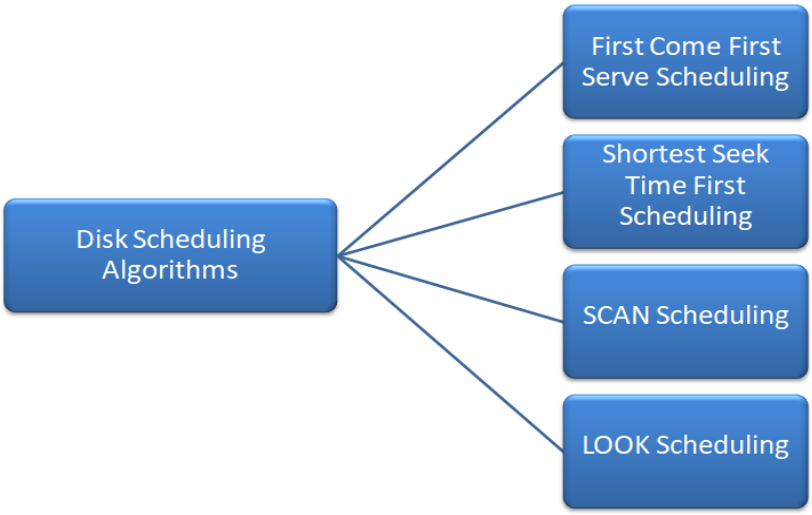
It is the average of time spent by each request waiting for the IO operation.

Disk Scheduling

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more requests may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

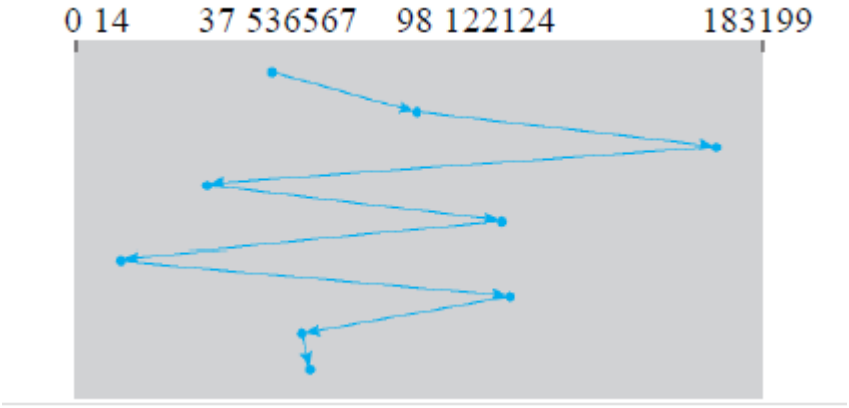
There are many disk scheduling algorithms that provide the total head movement for various requests to the disk. Here are the types of disk scheduling algorithms



FCFS Scheduling

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Example: A disk queue with requests for I/O to blocks on cylinders in that order 98, 183, 37, 122, 14, 124, 65, 67, and head starts at 53



FCFS disk scheduling

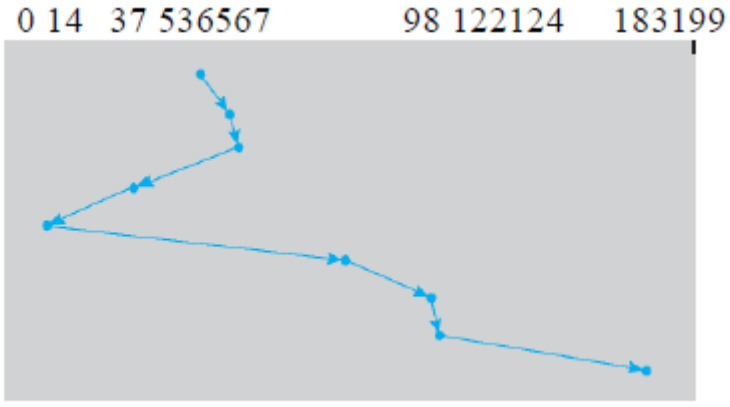
- If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders.
- The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule.
- If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

SSTF Scheduling

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It allows the head to move to the closest track in the service queue.

Example: Consider the above request queue, the closest request to the initial head position (53) is at cylinder 65 and head starts at 53.



SSTF disk scheduling

- Once we are at cylinder 65, the next closest request is at cylinder 67.
- From there, the request at cylinder 37 is closer than the one at 98, so37 is served next.
- Continuing, it services the request at cylinder 14, then 98, 122, 124, and finally 183. This scheduling method results in a total head movement of only 236 cylinders-little more than one-third of the distance needed for FCFS scheduling of this request queue.
- This algorithm gives a substantial improvement in performance.
- Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal.

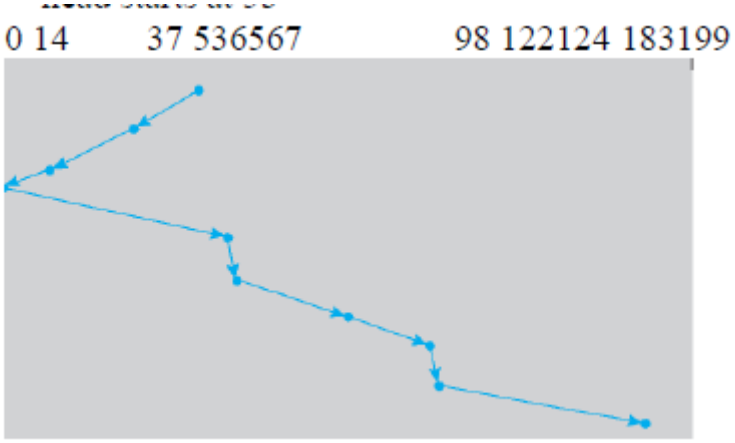
SCAN Scheduling

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns backward moves in the reverse direction satisfying requests coming in its path.

It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

- Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next service 37 and then 14.
- At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183.
- If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

Queue: 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



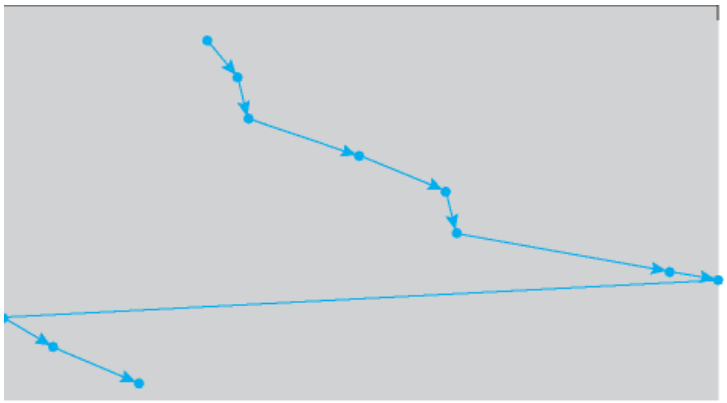
SCAN disk scheduling

C-SCAN Scheduling

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Queue: 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

0 14 37 536567 98 122124 183199



C- SCAN disk scheduling

LOOK and C-LOOK Scheduling

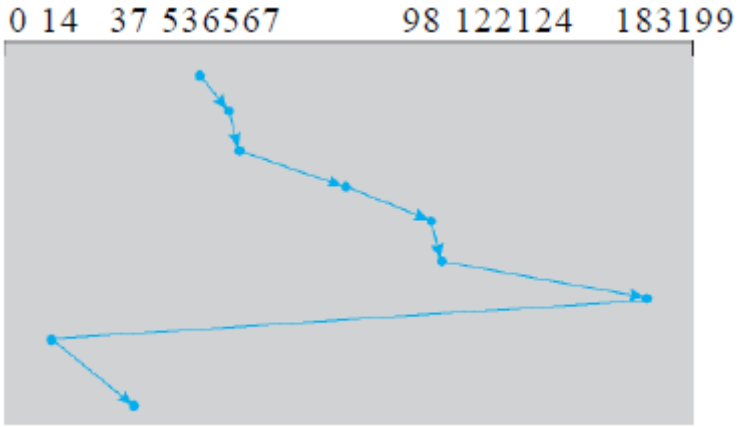
LOOK Scheduling

It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

C Look Scheduling

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

Queue: 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



C-LOOK disk scheduling

Selection of Disk Scheduling Algorithm

Deterministic − In deterministic disk scheduling, the algorithm is selected based on the characteristics of the workload. For example, if the workload consists of short requests, the SSTF algorithm may be the best choice.

Dynamic − In dynamic disk scheduling, the algorithm is selected based on the current state of the system. For example, if the disk is heavily loaded, the operating system may switch to a more efficient algorithm to handle the increased workload. Dynamic disk scheduling can adapt to changes in the workload and improve system performance.

The selection of the disk scheduling algorithm depends on several factors, including the characteristics of the workload, system performance requirements, and the available resources. The operating system must evaluate the different disk scheduling algorithms based on the evaluation criteria and select the algorithm that best meets the system's requirements.

Advantages

Improved performance − Disk scheduling algorithms ensure that data is accessed in the most efficient manner possible, which improves the performance of the system.

Fairness − Disk scheduling algorithms ensure that all requests for data access are treated fairly and given a chance to be processed.

Reduced disk fragmentation − Disk scheduling algorithms can help to reduce disk fragmentation by accessing data in a more organized manner.

Disadvantages

Overhead − Disk scheduling algorithms can create overhead and delay in processing data access requests, which can reduce overall system performance.

Complexity − Some disk scheduling algorithms can be complex and difficult to understand, which may make it difficult to optimize system performance.

Risk of starvation − Disk scheduling algorithms can result in starvation of certain requests, which can lead to inefficiencies and reduced system performance.

DISK MANAGEMENT

The operating system is also responsible for several other aspects of disk management.

Disk Formatting

- Before a disk can be used, it has to be **low-level formatted,** which means laying down all of the headers and trailers demarking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and **error-correcting codes, ECC** which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered (depending on the extent of the damage). Sector sizes are traditionally 512 bytes, but may be larger, particularly in larger drives.
- ECC calculation is performed with every disk read or write, and if damage is detected but the data is recoverable, then a **soft error** has occurred. Soft errors are generally handled by the on-board disk controller, and never seen by the OS.
- Once the disk is low-level formatted, the next step is to partition the drive into one or more separate partitions. This step must be completed even if the disk is to be used as a single large partition, so that the partition table can be written to the beginning of the disk.
- After partitioning, then the filesystems must be **logically formatted**, which involves laying down the master directory information (FAT table or inode structure), initializing free lists, and creating at least the root directory of the filesystem. (Disk partitions which are to be used as raw devices are not logically formatted. This saves the overhead and disk space of the filesystem structure, but requires that the application program manage its own disk storage requirements).

Boot Block

- Computer ROM contains a **bootstrap program** (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it. (The ROM bootstrap program may look in floppy and/or CD drives before accessing the hard drive, and is smart enough to recognize whether it has found valid boot code or not).
- The first sector on the hard drive is known as the **Master Boot Record, MBR** and contains a very small amount of code in addition to the partition table. The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the **active or boot partition.**
- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way.
- In a **dual-boot** (or larger multi-boot) system, the user may be given a choice of which operating system to boot, with a default action to be taken in the event of no response within some time frame.
- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS. The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, sched, init, etc.), and finally providing one or more login prompts. Boot options at this stage may include **single-user** a.k.a. **maintenance** or **safe modes**, in which very few system services are started - These modes are designed for system administrators to repair problems or otherwise maintain the system.

Bad Blocks

- No disk can be manufactured to 100% perfection, and all physical objects wear out over time. For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time. If a large number of blocks go bad then the entire disk will need to be replaced, but a few here and there can be handled through other means.

- In the old days, bad blocks had to be checked for manually. Formatting of the disk or running certain disk-analysis tools would identify bad blocks, and attempt to read the data off of them one last time through repeated tries. Then the bad blocks would be mapped out and taken out of future service. Sometimes the data could be recovered, and sometimes it was lost forever. (Disk analysis tools could be either destructive or non-destructive).
- Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered. (Recall that blocks are tested with every write as well as with every read, so often errors can be detected before the write operation is complete and the data simply written to a different sector instead).
- Note that re-mapping of sectors from their normal linear progression can throw off the disk scheduling optimization of the OS, especially if the replacement sector is physically far away from the sector it is replacing. For this reason most disks normally keep a few spare sectors on each cylinder, as well as at least one spare cylinder. Whenever possible a bad sector will be mapped to another sector on the same cylinder, or at least a cylinder as close as possible. **Sector slipping** may also be performed, in which all sectors between the bad sector and the replacement sector are moved down by one, so that the linear progression of sector numbers can be maintained.
- If the data on a bad block cannot be recovered, then a **hard error** has occurred, which requires replacing the file(s) from backups, or rebuilding them from scratch.

SWAP SPACE MANAGEMENT

Swapping

When our main memory (RAM) is not enough to temporarily store multiple programs then we take some program from RAM and store them into the hard disk by a mechanism called swap out. Similarly when RAM is free, then we again swap in the programs from hard disk to RAM.

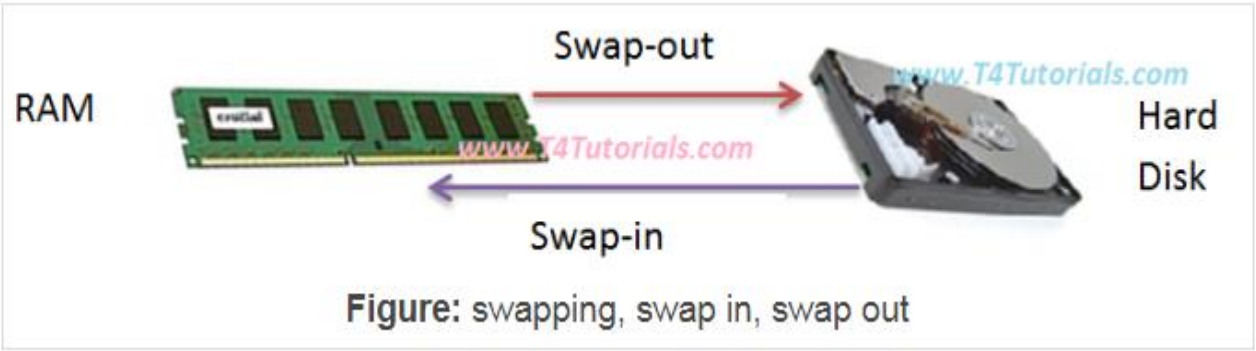


Figure: swapping, swap in, swap out

Swap out and Swap in

Swap out means to take the program from RAM and to bring them in Hard disk.
Swap in means to take the program from Hard disk and again bring them to the RAM.

Advantages:

- With the help of swapping we can manage many processes within the same RAM.
- Swapping helps to create the virtual memory.
- Swapping is economical.

FILE SYSTEM INTERFACE

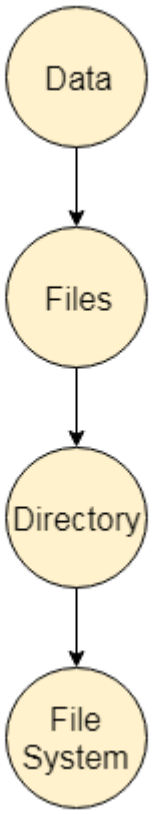
FILE CONCEPT

A **file** is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

A file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

The collection of files is known as **Directory.**

The collection of directories at the different levels is known as **File System.**



File Structure

A file has a certain defined structure according to its type.

- A **text file** is a sequence of characters organized into lines - txt, doc
- A **source file** is a sequence of procedures and functions - c, cpp, java
- An **object file** is a sequence of bytes organized into blocks that are understandable by the machine – obj, o
- An **Executable file** ready to run machine language program - exe, com, bin
- **Batch file** commands to the command interpreter - bat, sh
- **Word Processor file** various word processor formats - wp, tex, rrf, doc
- **Archive file** related files grouped into one compressed file - arc, zip, tar
- **Multimedia file** containing audio/video information - mpeg, mov, rm, mp3,avi

File Attributes

A file's attributes vary from one operating system to another but typically consist of these:

- **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 systems that support different types of files.
- **Location -** This information is a pointer to a device and to the location of the file on that device.
- **Size -** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection** - Access-control information 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.

File Operations

A file is an abstract data type. The operations that can be performed on files are:

- **Create** Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.
- **Write** Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which, the data needs to be written.
- **Read** Every file is opened in three different modes : Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.
- **Re-position** Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as seeking.
- **Delete** Deleting the file will not only delete all the data stored inside the file, It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.
- **Truncate** Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

FILE ACCESS METHODS

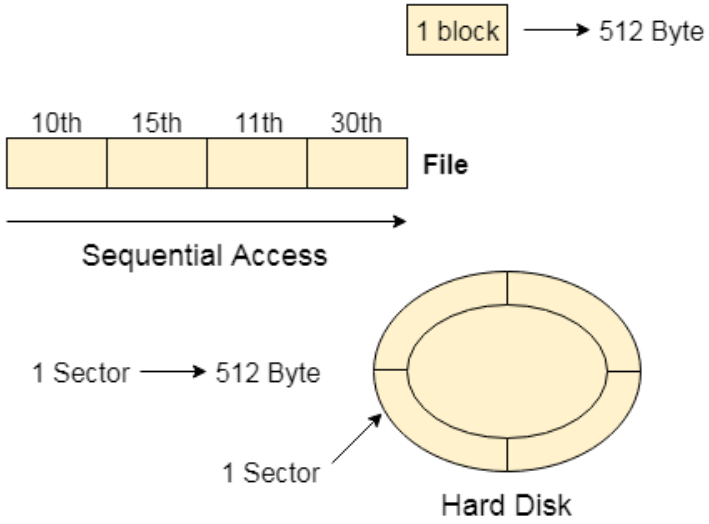
Files stores information. When it is used; information must be accessed.

Various ways to access files stored in secondary memory.

Sequential Access

Most of the operating systems access the file sequentially.

In sequential access, the OS read the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.

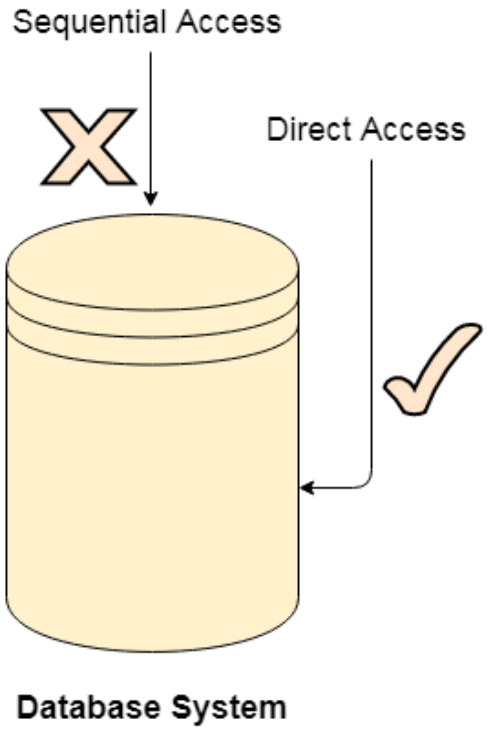


Direct Access

The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.

Suppose every block of the storage stores 4 records and we know that the record we needed is stored in 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record.

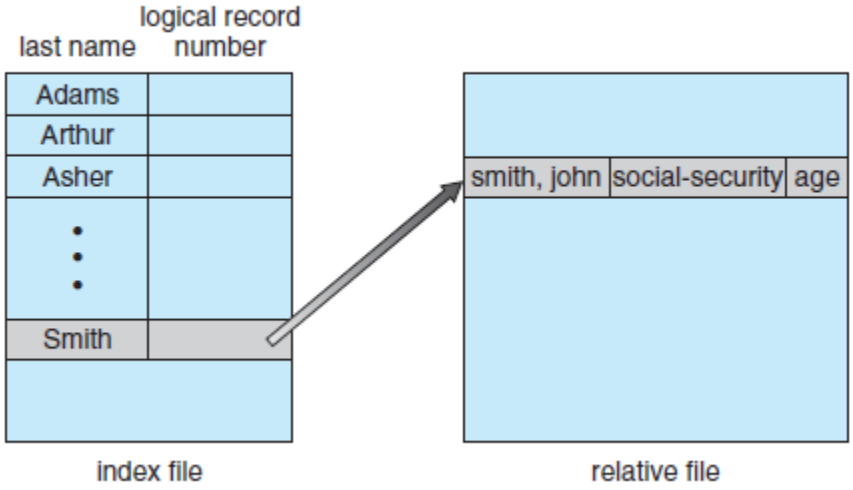
Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.



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.

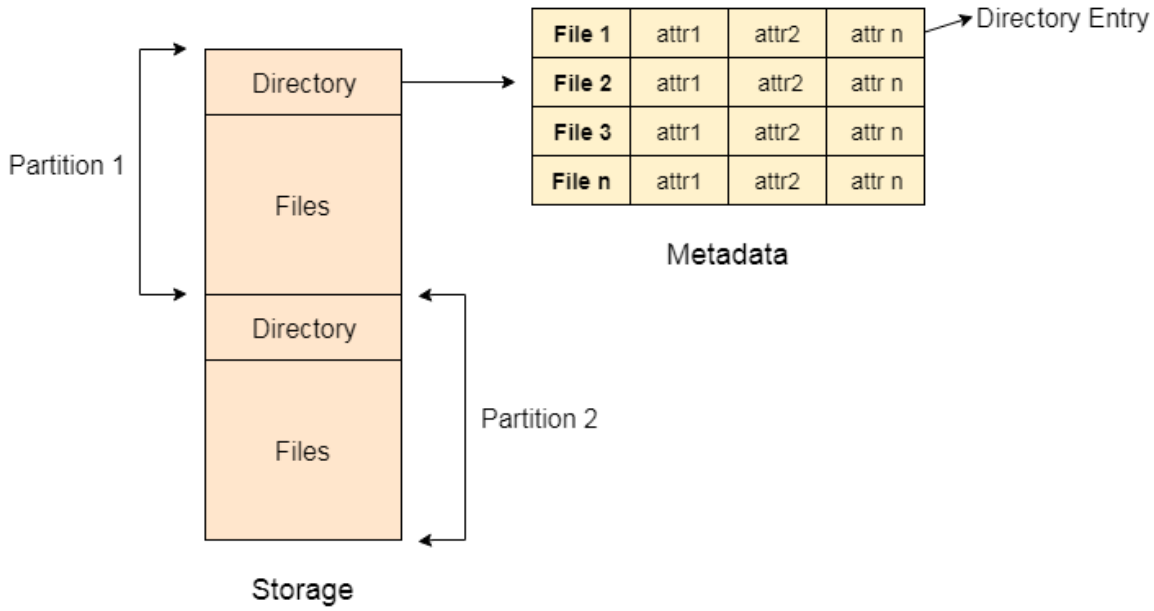


Example of index and relative files

DIRECTORY STRUCTURE

Collection of files is a file is called **Directory.**

Directory can be defined as the listing of the related files on the disk. The directory contains information about the files, including attributes, location and ownership.



A hard disk can be divided into the number of partitions of different sizes. The partitions are also called **volumes or mini disks.**

Each partition must have at least one directory in which, all the files of the partition can be listed. A directory entry is maintained for each file in the directory which stores all the information related to that file. A directory can be viewed as a file which contains the Meta data of the bunch of files.

Directory operations:

File Creation New files need to be created and added to the directory

Search for the file A particular file in a directory by their names.

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

Renaming the file 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.

Traversing Files 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.

Listing of files 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.

Advantages:

Efficiency: A file can be located more quickly.

Naming: It becomes convenient for users as two users can have same name for different files or may have different name for same file.

Grouping: Logical grouping of files can be done by properties e.g. all java programs, all games etc.

DIRECTORY ORGANIZATION

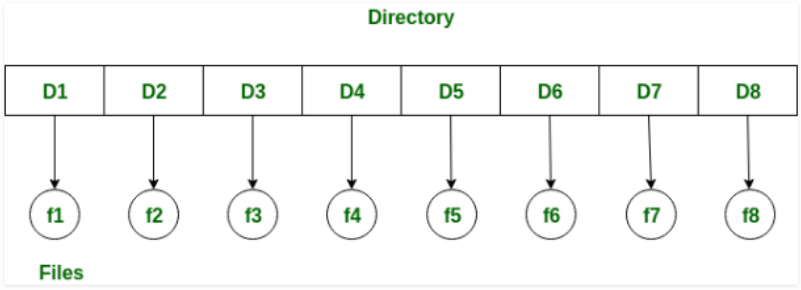
Structures of Directory

A directory is a container that is used to contain folders and file. It organises files and folders into hierarchical manner.

Single-Level Directory

Single level directory is simplest disectory structure.In it all files are contained in same directory which make it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name. if two users call there dataset test, then the unique name rule violated.



Advantages:

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

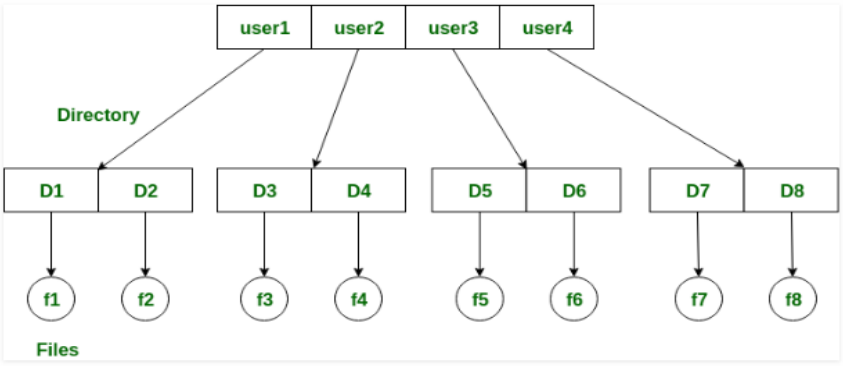
Disadvantages:

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

Two-level directory

As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user.

In the two-level directory structure, each user has there own user files directory (UFD). The UFDs has similar structures, but each lists only the files of a single user. system's master file directory (MFD) is searches whenever a new user id is logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



Advantages:

- We can give full path like /User-name/directory-name/.
- Diffrent users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

Disadvantages:

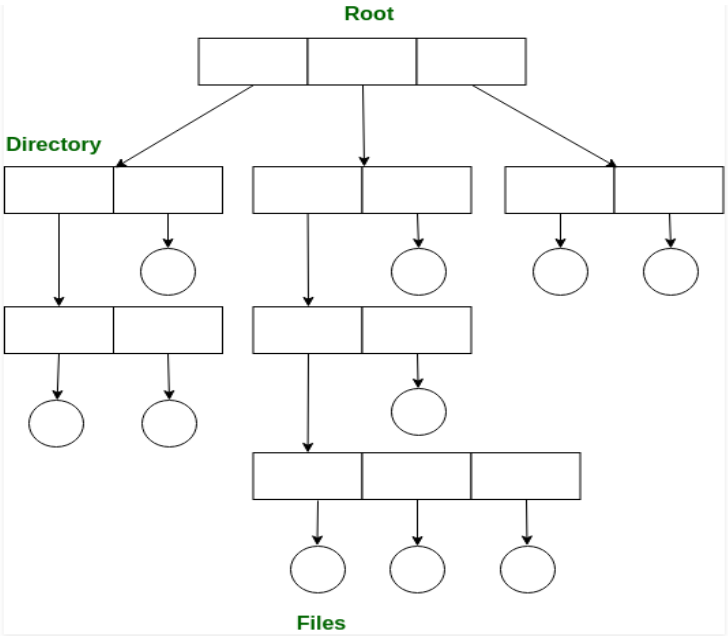
- A user is not allowed to share files with other users.
- Still it not very scalable, two files of the same type cannot be grouped together in the same user.

Tree-structured directory

Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.

This generalization allows the user to create there own subdirectories and to organise on their files accordingly.

A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path.



Path names can be of two types: absolute and relative.

An **absolute path name** begins at the root and follows a path down to the specified file, giving the directory names on the path.

A **relative path name** defines a path from the current directory.

Advantages:

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

Disadvantages:

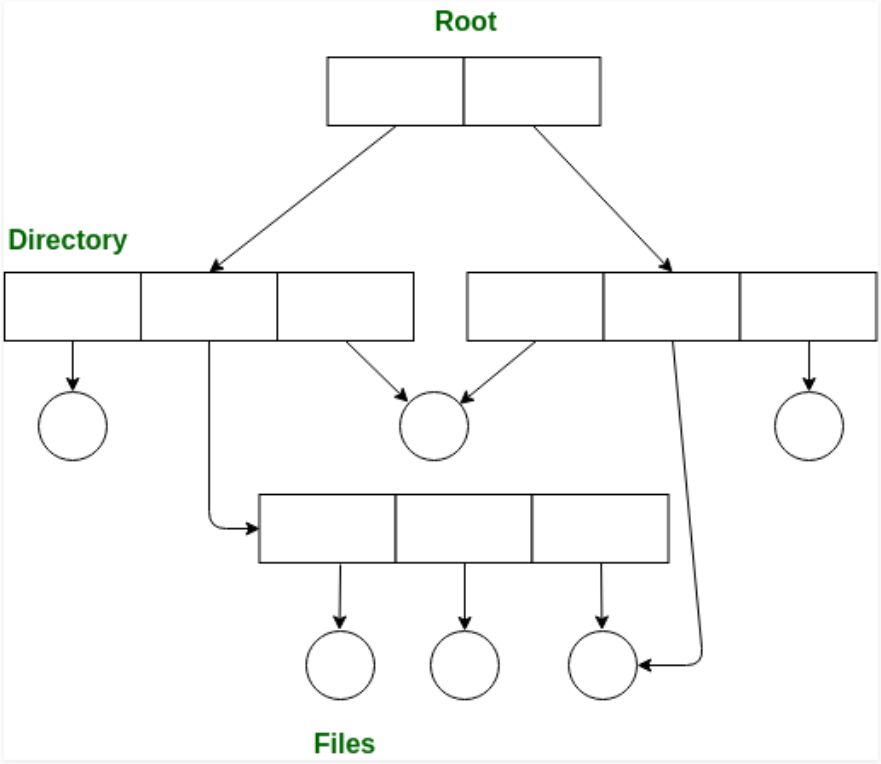
- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.

Acyclic graph directory

An acyclic graph is a graph with no cycle and allows to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.

It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separated them from other projects and files of other programmers since they are working on a joint project so they want to the subdirectories into there own directories. The common subdirectories should be shared. So here we use Acyclic directories.

It is the point to note that shared file is not the same as copy file if any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



Advantages:

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

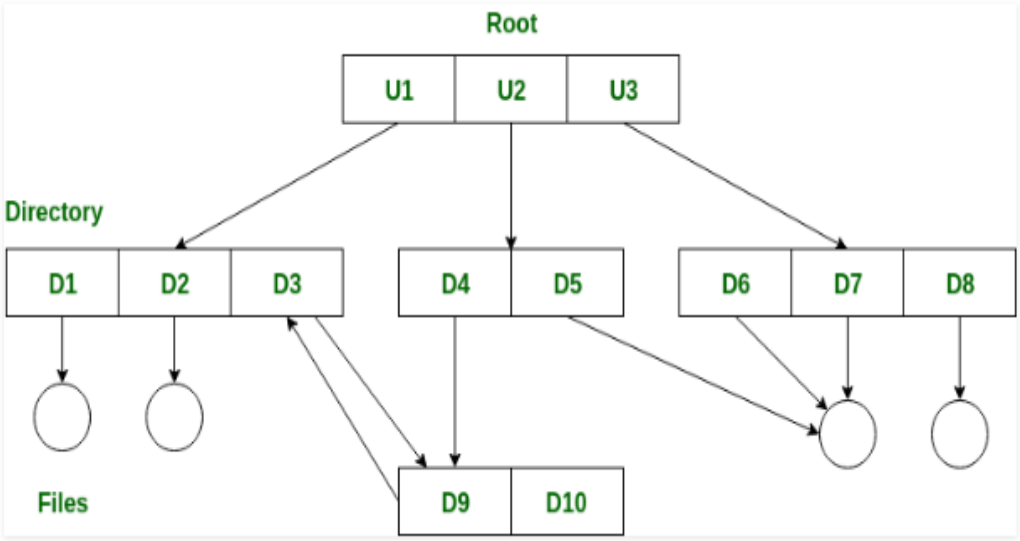
Disadvantages:

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

General graph directory structure

In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.

The main problem with this kind of directory structure is to calculate total size or space that have been taken by the files and directories.



Advantages:

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

Disadvantages:

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

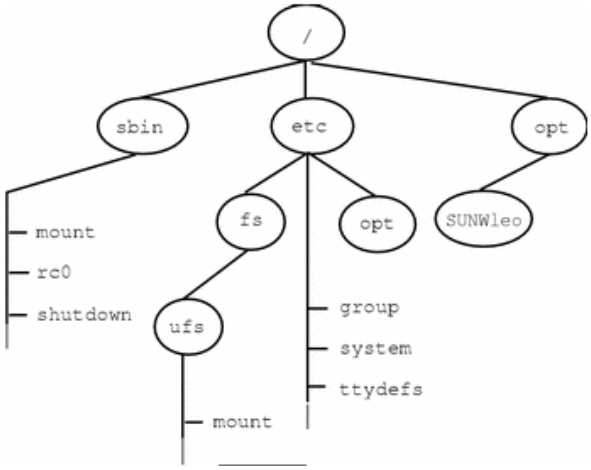
FILE SYSTEM MOUNTING

Mounting and Unmounting File Systems

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.

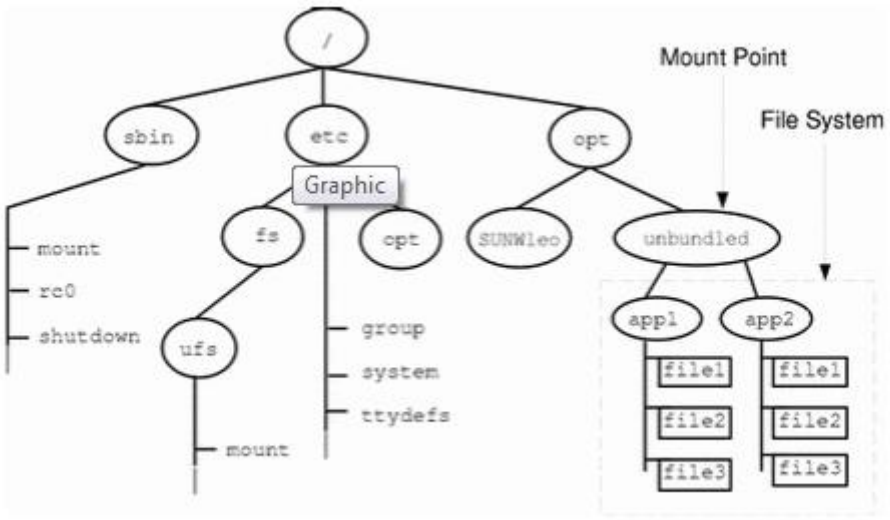
For example, the figure below shows a local file system, starting with a root (/) file system and subdirectories sbin, etc, and opt.



Sample root (/) File System

Now, say you wanted to access a local file system from the /opt file system that contains a set of unbundled products.

First, you must create a directory to use as a mount point for the file system you want to mount, for example, /opt/unbundled. Once the mount point is created, you can mount the file system (by using the mount command), which makes all of the files and directories in /opt/unbundled available, as shown in the figure below. See Chapter 36, Mounting and Unmounting File Systems (Tasks) for detailed instructions on how to perform these tasks.



Mounting a File System

The Mounted File System Table

Whenever you mount or unmount a file system, the /etc/mnttab (mount table) file is modified with the list of currently mounted file systems. You can display the contents of this file with the cat or more commands, but you cannot edit it. Here is an example of an /etc/mnttab file:

```
$ more /etc/mnttab
/dev/dsk/c0t0d0s0  /  ufs rw,intr,largefiles,onerror=panic,suid,dev=2200000 938557523
/proc   /proc   proc    dev=3180000     938557522
fd      /dev/fd fd      rw,suid,dev=3240000     938557524
mnttab  /etc/mnttab     mntfs   dev=3340000     938557526
swap    /var/run        tmpfs   dev=1   938557526
swap    /tmp    tmpfs   dev=2   938557529
/dev/dsk/c0t0d0s7 /export/home ufs rw,intr,largefiles,onerror=panic,suid,dev=2200007 938557529
$
```

FILE SHARING AND PROTECTION

FILE SHARING

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 (the universe, a.k.a. Others.)
- Some systems have more complicated access control, allowing or denying specific accesses to specifically named users or groups.

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. Ftp can be either account and password controlled, or anonymous, not requiring any user name or password.
- Various forms of distributed file systems allow remote file systems to be mounted onto a local directory structure, and accessed using normal file access commands. (The actual files are still transported across the network as needed, possibly using ftp as the underlying transport mechanism.)
- The WWW has made it easy once again to access files on remote systems without mounting their filesystems, generally using (anonymous) ftp as the underlying file transport mechanism.

The Client-Server Model

When one computer system remotely mounts a filesystem that is physically located on another system, the system which physically owns the files acts as a server, and the system which mounts them is the client.

User IDs and group IDs must be consistent across both systems for the system to work properly. (I.e. this is most applicable across multiple computers managed by the same organization, shared by a common group of users.)

The same computer can be both a client and a server. (E.g. cross-linked file systems.)

There are a number of security concerns involved in this model:

- Servers commonly restrict mount permission to certain trusted systems only. Spoofing (a computer pretending to be a different computer) is a potential security risk.
- Servers may restrict remote access to read-only.
- Servers restrict which filesystems may be remotely mounted. Generally the information within those subsystems is limited, relatively public, and protected by frequent backups.

The NFS (Network File System) is a classic example of such a system.

Distributed Information Systems

The Domain Name System, DNS, provides for a unique naming system across all of the Internet.

Domain names are maintained by the Network Information System, NIS, which unfortunately has several security issues. NIS+ is a more secure version, but has not yet gained the same widespread acceptance as NIS.

Microsoft's Common Internet File System, CIFS, establishes a network login for each user on a networked system with shared file access. Older Windows systems used domains, and newer systems (XP, 2000), use active directories. User names must match across the network for this system to be valid.

A newer approach is the Lightweight Directory-Access Protocol, LDAP, which provides a secure single sign-on for all users to access all resources on a network. This is a secure system which is gaining in popularity, and which has the maintenance advantage of combining authorization information in one 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.
- However when a remote file is unavailable, there are many possible reasons, and whether or not it is unrecoverable is not readily apparent. Hence most remote access systems allow for blocking or delayed response, in the hopes that the remote system (or the network) will come back up eventually.

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?

At first glance this appears to have all of the synchronization issues discussed in Chapter 6. Unfortunately the long delays involved in network operations prohibit the use of atomic operations as discussed in that chapter.

UNIX Semantics

The UNIX file system uses the following semantics:

- Writes to an open file are immediately visible to any other user who has the file open.
- One implementation uses a shared location pointer, which is adjusted for all sharing users.

The file is associated with a single exclusive physical resource, which may delay some accesses.

Session Semantics

The Andrew File System, 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.

According to these semantics, a file can be associated with multiple (possibly different) views. Almost no constraints are imposed on scheduling accesses. No user is delayed in reading or writing their personal copy of the file.

Immutable-Shared-Files Semantics

Under this system, 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.

FILE PROTECTION

When information is stored in a computer system, we want to keep it safe from physical damage (the issue of reliability) and improper access (the issue of protection). Reliability is generally provided by duplicate copies of files.

Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.

File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism.

Files may be deleted accidentally.

Protection can be provided in many ways. For a single-user laptop system, we might provide protection by locking the computer in a desk drawer or file cabinet.

In a larger multiuser system, however, other mechanisms are needed.

Types of Access

Protection mechanisms provide controlled access by limiting the types of file access that can be made.

Access is permitted or denied depending on several factors, one of which is the type of access requested.

Several different types of operations may be controlled:

Read: Read from the file.

Write: Write or rewrite the file.

Execute: Load the file into memory and execute it.

Append: Write new information at the end of the file.

Delete: Delete the file and free its space for possible reuse.

List: List the name and attributes of the file.

Access Control

The most general scheme to implement identity-dependent access is to associate with each file and directory an **access-control list** (ACL) specifying user names and the types of access allowed for each user.

When a user requests access to a particular file, the operating system checks the access list associated with that file.

If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

Three classifications of users

Owner: The user who created the file is the owner. Its value is 7.

Read	Write	Execute
1	1	1

Group: A set of users who are sharing the file and need similar access is a group. Its value is 6.

Read	Write	Execute
1	1	0

Universe: All other users in the system. Its value is 1.

Read	Write	Execute
0	0	1

Other Protection Approaches

Another approach to the protection problem is to associate a password with each file. If the passwords are chosen randomly and changed often, this scheme may be effective in limiting access to a file.

The use of passwords has a few disadvantages:

- First, the number of passwords that a user needs to remember may become large, making the scheme impractical.
- Second, if only one password is used for all the files, then once it is discovered, all files are accessible.

FILE SYSTEM IMPLEMENTATION

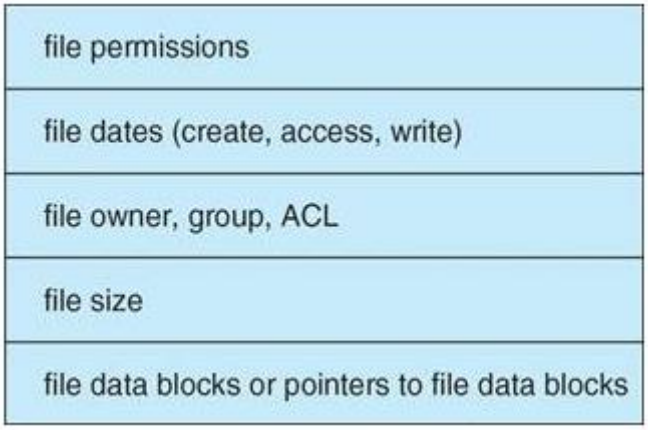
File systems store several important data structures on the disk:

A boot-control block, (per volume) a.k.a. the boot block in UNIX or the partition boot sector in Windows contains information about how to boot the system off of this disk. This will generally be the first sector of the volume if there is a bootable system loaded on that volume, or the block will be left vacant otherwise.

A volume control block, (per volume) a.k.a. the master file table in UNIX or the superblock in Windows, which contains information such as the partition table, number of blocks on each filesystem, and pointers to free blocks and free FCB blocks.

A directory structure (per file system), containing file names and pointers to corresponding FCBs. UNIX uses inode numbers, and NTFS uses a master file table.

The File Control Block, FCB, (per file) containing details about ownership, size, permissions, dates, etc. UNIX stores this information in inodes, and NTFS in the master file table as a relational database structure.



File Control Block

There are also several key data structures stored in memory:

- An in-memory mount table.
- An in-memory directory cache of recently accessed directory information.

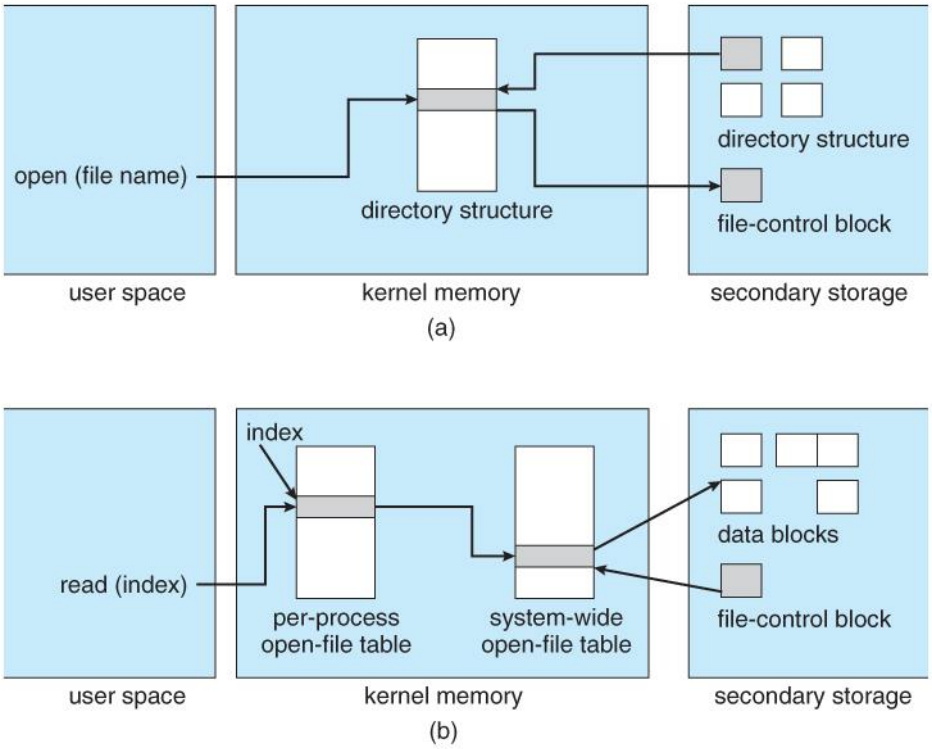
A **system-wide open file table,** containing a copy of the FCB for every currently open file in the system, as well as some other related information.

A **per-process open file table**, containing a pointer to the system open file table as well as some other information.

When a new file is created, a new FCB is allocated and filled out with important information regarding the new file. The appropriate directory is modified with the new file name and FCB information.

When a file is accessed during a program, the open( ) system call reads in the FCB information from disk, and stores it in the system-wide open file table. An entry is added to the per-process open file table referencing the system-wide table, and an index into the per-process table is returned by the open( ) system call. UNIX refers to this index as a file descriptor, and Windows refers to it as a file handle.

If another process already has a file open when a new request comes in for the same file, and it is sharable, then a counter in the system-wide table is incremented and the per-process table is adjusted to point to the existing entry in the system-wide table.
When a file is closed, the per-process table entry is freed, and the counter in the system-wide table is decremented. If that counter reaches zero, then the system wide table is also freed. Any data currently stored in memory cache for this file is written out to disk if necessary.



In-memory file system structures. (a) File open (b) File read

Partitions and Mounting

Physical disks are commonly divided into smaller units called **partitions.** They can also be combined into larger units, but that is most commonly done for RAID installations.
Partitions can either be used as raw devices (with no structure imposed upon them), or they can be formatted to hold a filesystem (i.e. populated with FCBs and initial directory structures as appropriate).
Raw partitions are generally used for swap space, and may also be used for certain programs such as databases that choose to manage their own disk storage system. Partitions containing filesystems can generally only be accessed using the file system structure by ordinary users, but can often be accessed as a raw device also by root.

The **boot block** is accessed as part of a raw partition, by the boot program prior to any operating system being loaded. Modern boot programs understand multiple OSes and filesystem formats, and can give the user a choice of which of several available systems to boot.

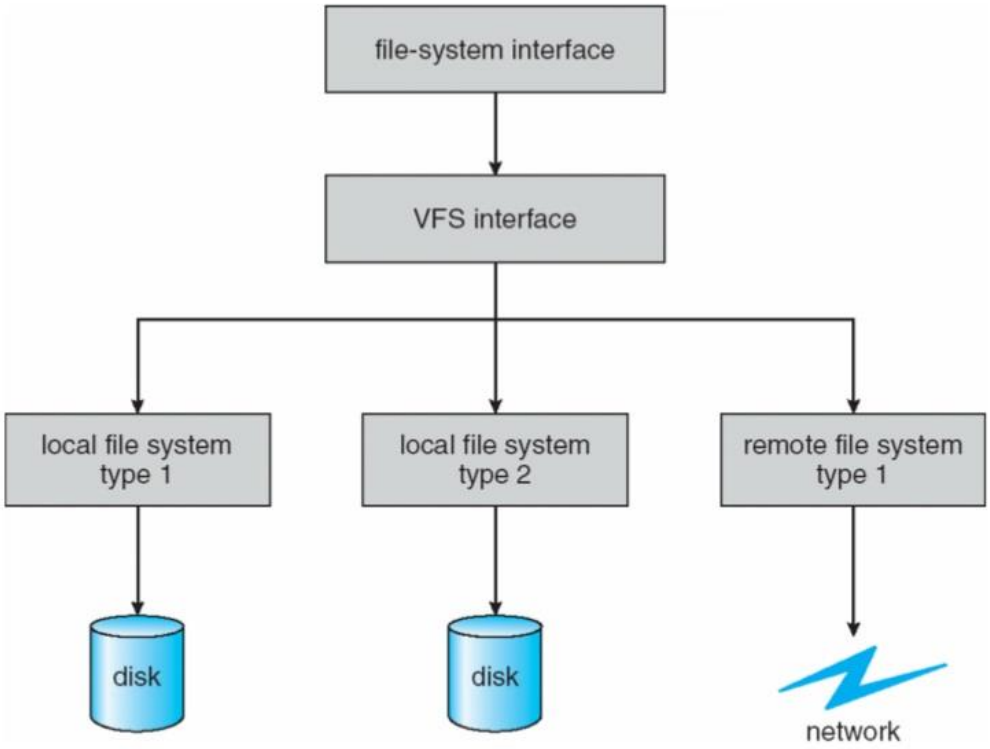
The **root partition** contains the OS kernel and at least the key portions of the OS needed to complete the boot process. At boot time the root partition is mounted, and control is transferred from the boot program to the kernel found there. (Older systems required that the root partition lie completely within the first 1024 cylinders of the disk, because that was as far as the boot program could reach. Once the kernel had control, then it could access partitions beyond the 1024 cylinder boundary).

Continuing with the boot process, additional filesystems get mounted, adding their information into the appropriate **mount table structure.** As a part of the mounting process the file systems may be checked for errors or inconsistencies, either because they are flagged as not having been closed properly the last time they were used, or just for general principals. Filesystems may be mounted either automatically or manually.

Virtual File Systems

A virtual file system (VFS) is programming that forms an interface between an operating system's kernel and a more concrete file system.

VFS, provide a common interface to multiple different filesystem types. In addition, it provides for a unique identifier (vnode) for files across the entire space, including across all filesystems of different types. (UNIX inodes are unique only across a single filesystem, and certainly do not carry across networked file systems).



View of virtual file system

The VFS in Linux is based upon four key object types:

- The **inode object,** representing an individual file
- The **file object,** representing an open file.
- The **superblock object,** representing a filesystem.
- The **dentry object,** representing a directory entry.

Linux VFS provides a set of common functionalities for each filesystem, using function pointers accessed through a table. The same functionality is accessed through the same table position for all filesystem types, though the actual functions pointed to by the pointers may be filesystem-specific.

Example: Some of the operations for the file object includes:

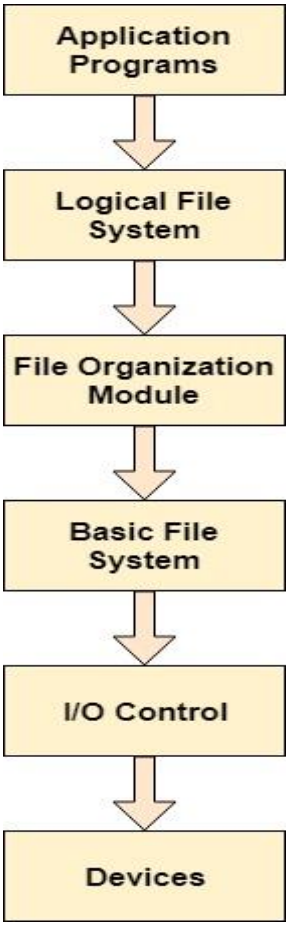
int open(...)	-	*Open a file.*
int close(...)	-	*Close an already-open file.*
ssize t read(...)	-	*Read from a file.*
ssize t write(...)	-	*Write to a file.*
int mmap(...)	-	*Memory-map a file.*

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.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



When an **application program** asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.

Generally, files are divided into various **logical blocks**. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.

Once **File organization module** decided which physical block the application program needs, it passes this information to basic file system. The **basic file system** is responsible for issuing the commands to I/O control in order to fetch those blocks.

I/O controls contain the codes by using which it can access hard disk. These codes are known as **device drivers**. I/O controls are also responsible for handling interrupts.

DIRECTORY IMPLEMENTATION

There is the number of algorithms by using which, the directories can be implemented. However, the selection of an appropriate directory implementation algorithm may significantly affect the performance of the system.

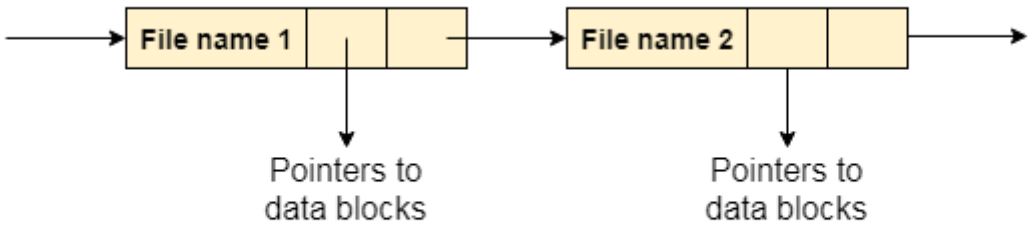
Linear List

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.

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 the directory for the named file and then release the space allocated to it.

To **reuse the directory entry,** we can do one of several things. We can mark the entry as unused (by assigning it a special name, such as an all-blank name, or by including a used– unused bit in each entry), or we can attach it to a list of free directory entries.



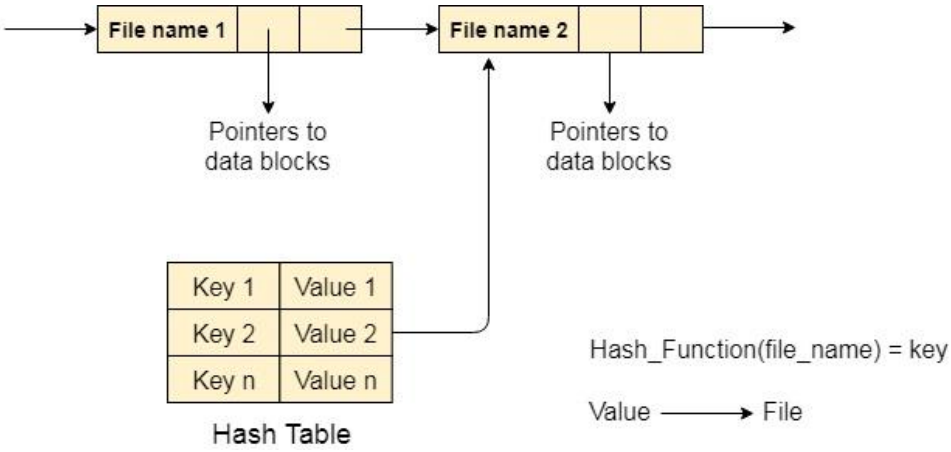
Linear List

Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

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.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.

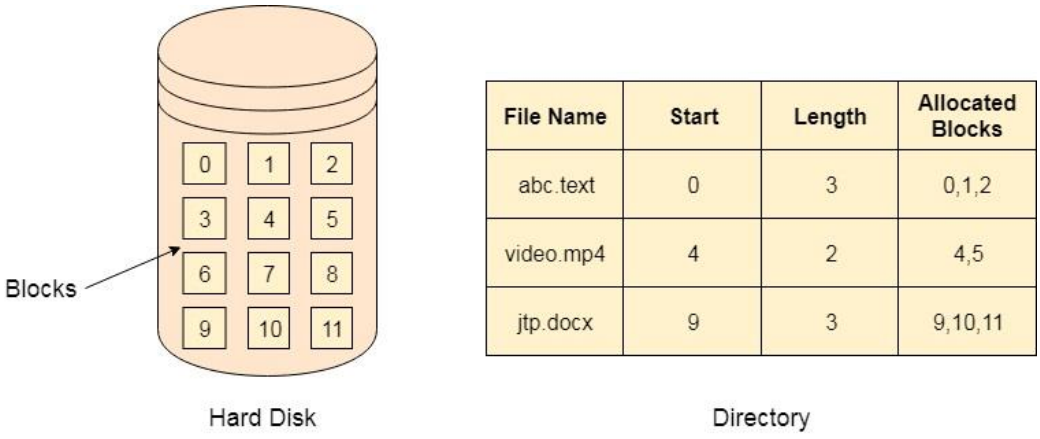


ALLOCATION METHODS

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.
Three major methods of allocating disk space are in wide use: **contiguous, linked, and indexed.**

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.



File Name	Start	Length	Allocated Blocks
abc.text	0	3	0,1,2
video.mp4	4	2	4,5
jtp.docx	9	3	9,10,11

Directory

Contiguous Allocation

Advantages

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

Disadvantages

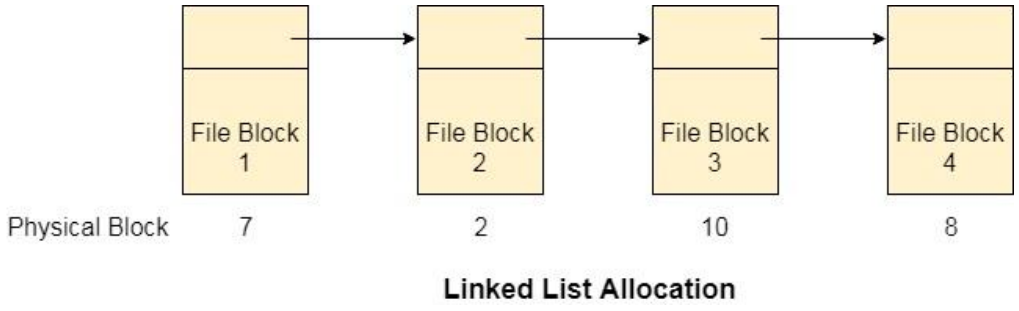
- The disk will become fragmented.
- It may be difficult to have a file grow.

Linked 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.

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.



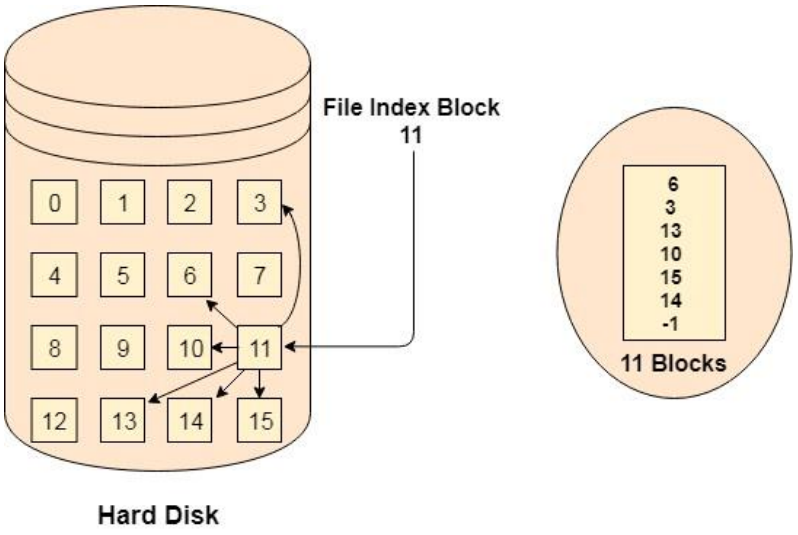
Linked List Allocation

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

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.



Advantages

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

Disadvantages

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

FREE-SPACE MANAGEMENT

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.

The free space list can be implemented mainly as:

- Bitmap
- Linked list
- Grouping
- Counting

Bit Vector

Frequently, 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.

Example:

Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated.

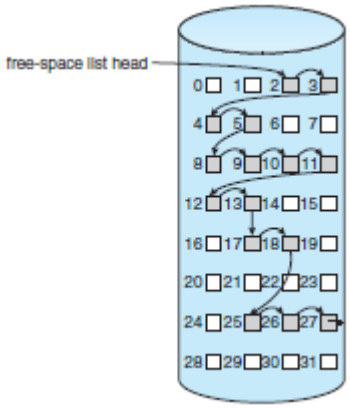
The free-space bit map would be **00111100111111000110000001110000 0...**

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.
The first non-0 word is scanned for the first 1 bit, which is the location of the first free block.
The calculation of the block number is

(number of bits per word) × (number of 0-value words) + offset of first 1 bit.

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.



Linked free-space list on disk

This first block contains a pointer to the next free disk block, and so on.

In our example, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated. In this situation, 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.

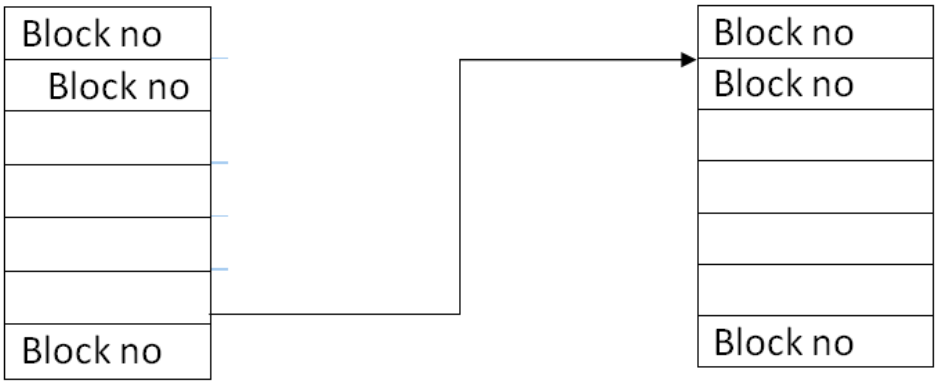
Grouping

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.



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.

EFFICIENCY AND PERFORMANCE

Efficiency

UNIX pre-allocates inodes, which occupies space even before any files are created.

UNIX also distributes inodes across the disk, and tries to store data files near their inode, to reduce the distance of disk seeks between the inodes and the data.

Some systems use variable size clusters depending on the file size.

The more data that is stored in a directory (e.g. last access time), the more often the directory blocks have to be re-written.

As technology advances, addressing schemes have had to grow as well.

Kernel table sizes used to be fixed, and could only be changed by rebuilding the kernels. Modern tables are dynamically allocated, but that requires more complicated algorithms for accessing them.

Performance

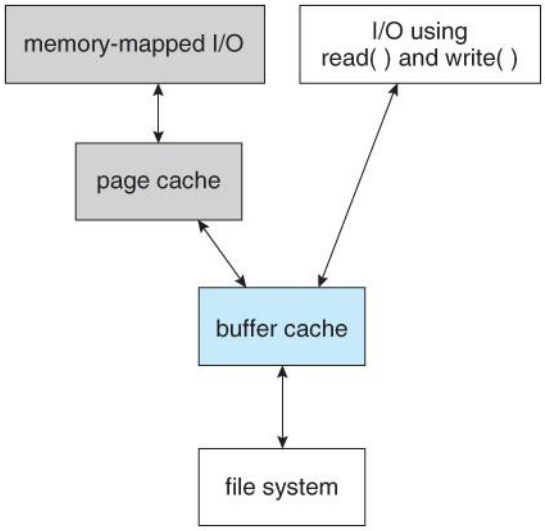
Disk controllers generally include on-board caching. When a seek is requested, the heads are moved into place, and then an entire track is read, starting from whatever sector is currently under the heads (reducing latency). The requested sector is returned and the unrequested portion of the track is cached in the disk's electronics.

Some OSes cache disk blocks they expect to need again in a **buffer cache.**

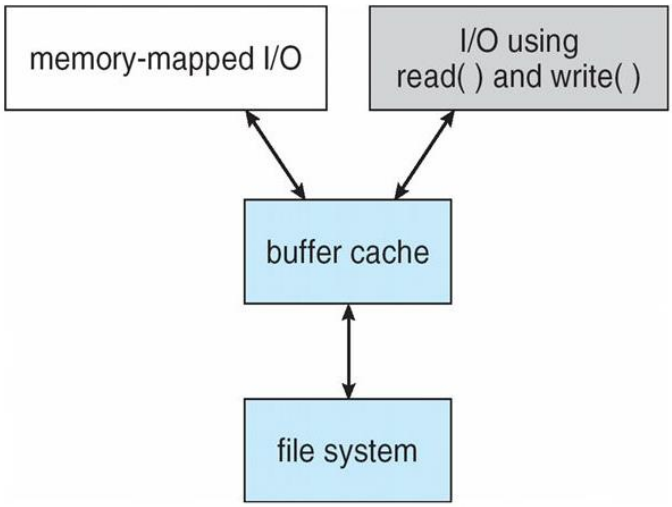
A **page cache** connected to the virtual memory system is actually more efficient as memory addresses do not need to be converted to disk block addresses and back again.

Some systems (Solaris, Linux, Windows 2000, NT, XP) use page caching for both process pages and file data in a **unified virtual memory.**

Diagram shows the advantages of the **unified buffer cache** found in some versions of UNIX and Linux - Data does not need to be stored twice, and problems of inconsistent buffer information are avoided.



I/O Without a Unified Buffer Cache



I/O Using a Unified Buffer Cache

Page replacement strategies can be complicated with a unified cache, as one need to decide whether to replace process or file pages, and how many pages to guarantee to each category of pages. Solaris, for example, has gone through many variations, resulting in **priority paging** giving process pages priority over file I/O pages, and setting limits so that neither can knock the other completely out of memory.

Another issue affecting performance is the question of whether to implement synchronous writes or asynchronous writes.

Synchronous writes occur in the order in which the disk subsystem receives them, without caching. **Asynchronous writes** are cached, allowing the disk subsystem to schedule writes in a more efficient order. Metadata writes are often done synchronously. Some systems support flags to the open call requiring that writes be synchronous, for example for the benefit of database systems that require their writes be performed in a required order.

The type of file access can also have an impact on optimal page replacement policies. Sequential access files often take advantage of two special policies:

- **Free-behind** frees up a page as soon as the next page in the file is requested, with the assumption that we are now done with the old page and won't need it again for a long time.
- **Read-ahead** reads the requested page and several subsequent pages at the same time, with the assumption that those pages will be needed in the near future.

The caching system and asynchronous writes speed up disk writes considerably, because the disk subsystem can schedule physical writes to the disk to minimize head movement and disk seek times.

RECOVERY

Files and directories are kept both in main memory and on disk, and care must be taken to ensure that system failure does not result in loss of data or in data inconsistency.

Consistency Checking

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.

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 filesystem.

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. For example, one strategy might be:

- At the beginning of the month do a full backup.
- At the end of the first and again at the end of the second week, backup all files which have changed since the beginning of the month.
- At the end of the third week, backup all files that have changed since the end of the second week.
- Every day of the month not listed above, do an incremental backup of all files that have changed since the most recent of the weekly backups described above.

Backup tapes are often reused, particularly for daily backups, but there are limits to how many times the same tape can be used.

Every so often a full backup should be made that is kept "forever" and not overwritten.

Backup tapes should be tested, to ensure that they are readable!

For optimal security, backup tapes should be kept off-premises, so that a fire or burglary cannot destroy both the system and the backups. There are companies (e.g. Iron Mountain) that specialize in the secure off-site storage of critical backup information.

Keep your backup tapes secure - The easiest way for a thief to steal all your data is to simply pocket your backup tapes!

Storing important files on more than one computer can be an alternate though less reliable form of backup.

Note that incremental backups can also help users to get back a previous version of a file that they have since changed in some way.

Beware that backups can help forensic investigators recover e-mails and other files that users had though they had deleted!

I/O SYSTEMS

The three major jobs of a computer are Input, Output, and Processing. In a lot of cases, the most important job is Input / Output, and the processing is simply incidental.

For example, when you browse a web page or edit any file, our immediate attention is to read or enter some information, not for computing an answer.

The primary role of the operating system in computer Input / Output is to manage and organize I/O operations and all I/O devices.

Overview

The controlling of various devices that are connected to the computer is a key concern of operating-system designers. This is because I/O devices vary so widely in their functionality and speed (for example a mouse, a hard disk and a CD-ROM), varied methods are required for controlling them. These methods form the I/O sub-system of the kernel of OS that separates the rest of the kernel from the complications of managing I/O devices.

I/O Hardware

I/O devices can be roughly categorized as storage, communications, user-interface, and other

Devices communicate with the computer via signals sent over wires or through the air.

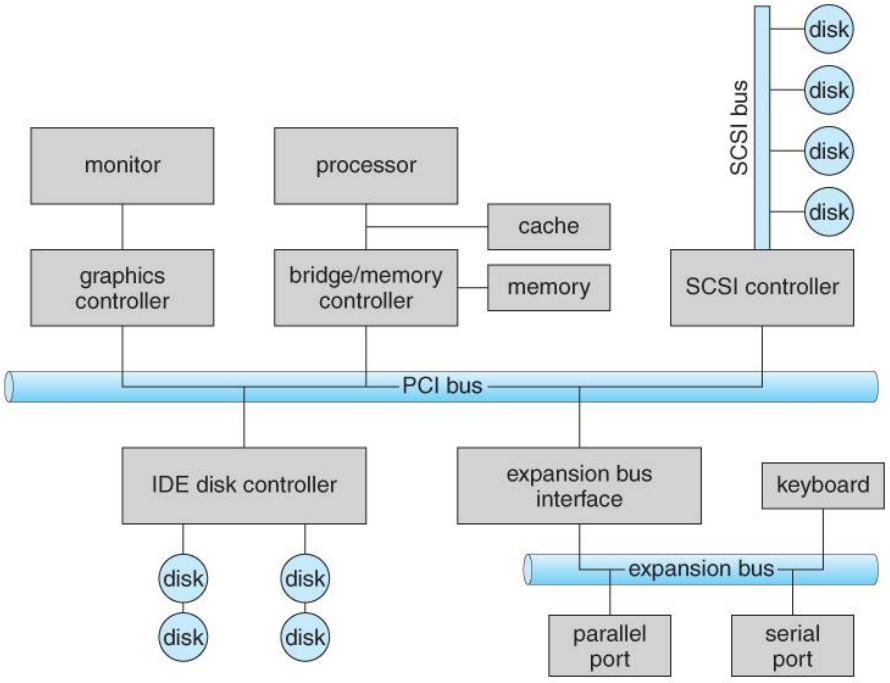
Devices connect with the computer via ports, e.g. a serial or parallel port.

A common set of wires connecting multiple devices is termed a **bus.**

Buses include rigid protocols for the types of messages that can be sent across the bus and the procedures for resolving contention issues.

There are four bus types commonly found in a modern PC:

- The **PCI bus** connects high-speed high-bandwidth devices to the memory subsystem (and the CPU).
- The **expansion bus** connects slower low-bandwidth devices, which typically deliver data one character at a time (with buffering).
- The **SCSI bus** connects a number of SCSI devices to a common SCSI controller.
- A **daisy-chain bus,** (not shown) is when a string of devices is connected to each other like beads on a chain, and only one of the devices is directly connected to the host.



A typical PC bus structure

One way of communicating with devices is through registers associated with each port. Registers may be one to four bytes in size, and may typically include (a subset of) the following four:

- The **data-in register** is read by the host to get input from the device.
- The **data-out register** is written by the host to send output.
- The **status register** has bits read by the host to ascertain the status of the device, such as idle, ready for input, busy, error, transaction complete, etc.
- The **control register** has bits written by the host to issue commands or to change settings of the device such as parity checking, word length, or full- versus half-duplex operation.

The most common I/O port address ranges.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Another technique for communicating with devices is memory-mapped I/O.

- In this case a certain portion of the processor's address space is mapped to the device, and communications occur by reading and writing directly to/from those memory areas.
- Memory-mapped I/O is suitable for devices which must move large quantities of data quickly, such as graphics cards.
- Memory-mapped I/O can be used either instead of or more often in combination with traditional registers. For example, graphics cards still use registers for control information such as setting the video mode.
- A potential problem exists with memory-mapped I/O, if a process is allowed to write directly to the address space used by a memory-mapped I/O device.

Polling

One simple means of device **handshaking** involves polling:

- The host repeatedly checks the **busy bit** on the device until it becomes clear.
- The host writes a byte of data into the data-out register, and sets the **write bit** in the command register (in either order.)
- The host sets the **command ready bit** in the command register to notify the device of the pending command.
- When the device controller sees the command-ready bit set, it first sets the busy bit.
- Then the device controller reads the command register, sees the write bit set, reads the byte of data from the data-out register, and outputs the byte of data.
- The device controller then clears the **error bit** in the status register, the command-ready bit, and finally clears the busy bit, signaling the completion of the operation.

Polling can be very fast and efficient, if both the device and the controller are fast and if there is significant data to transfer. It becomes inefficient, however, if the host must wait a long time in the busy loop waiting for the device, or if frequent checks need to be made for data that is infrequently there.

Interrupts

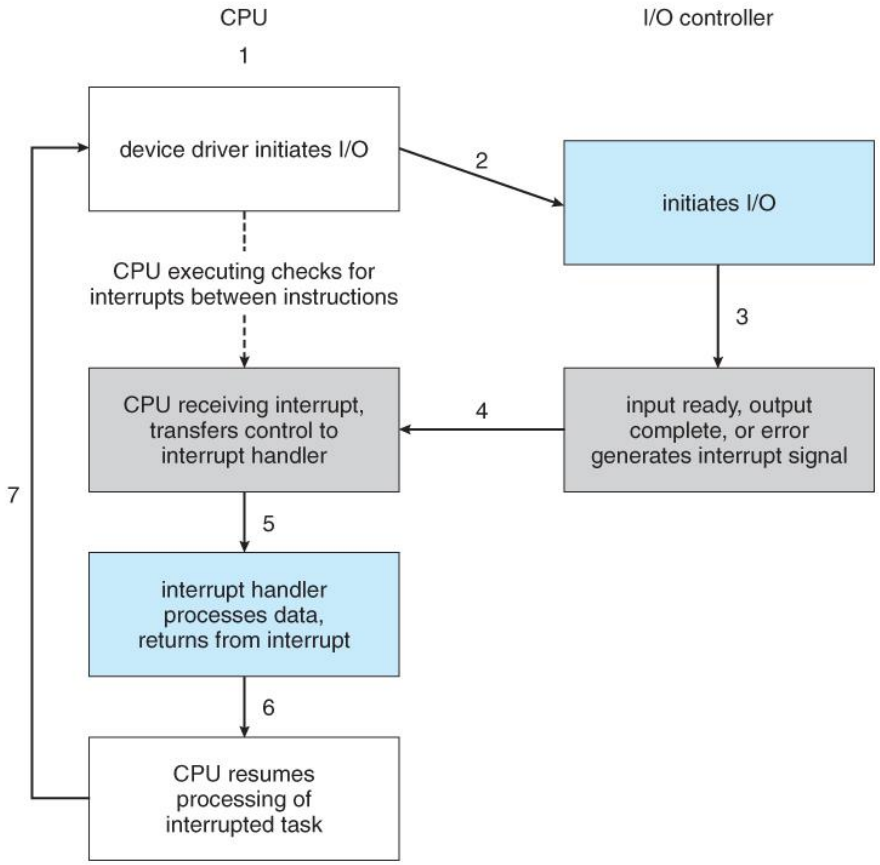
Interrupts allow devices to notify the CPU when they have data to transfer or when an operation is complete, allowing the CPU to perform other duties when no I/O transfers need its immediate attention.

The CPU has an **interrupt-request line** that is sensed after every instruction.

A device's controller **raises** an interrupt by asserting a signal on the interrupt request line.

The CPU then performs a state save, and transfers control to the **interrupt handler** routine at a fixed address in memory. (The CPU catches the interrupt and dispatches the interrupt handler).

The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a **return from interrupt** instruction to return control to the CPU. (The interrupt handler clears the interrupt by servicing the device).



Interrupt-driven I/O cycle

The above description is adequate for simple interrupt-driven I/O, but there are three needs in modern computing which complicate the picture:

- The need to defer interrupt handling during critical processing,
- The need to determine which interrupt handler to invoke, without having to poll all devices to see which one needs attention, and
- The need for multi-level interrupts, so the system can differentiate between high- and low-priority interrupts for proper response.

These issues are handled in modern computer architectures with **interrupt-controller** hardware.

- Most CPUs now have two interrupt-request lines: One that is **non-maskable** for critical error conditions and one that is **maskable,** that the CPU can temporarily ignore during critical processing.
- The interrupt mechanism accepts an **address,** which is usually one of a small set of numbers for an offset into a table called the **interrupt vector.** This table (usually located at physical address zero ?) holds the addresses of routines prepared to process specific interrupts.
- The number of possible interrupt handlers still exceeds the range of defined interrupt numbers, so multiple handlers can be **interrupt chained.** Effectively the addresses held in the interrupt vectors are the head pointers for linked-lists of interrupt handlers.

Direct Memory Access

For devices that transfer large quantities of data (such as disk controllers), it is wasteful to tie up the CPU transferring data in and out of registers one byte at a time. Instead this work can be off-loaded to a special processor, known as the **Direct Memory Access, DMA, Controller.**

The host issues a command to the DMA controller, indicating the location where the data is located, the location where the data is to be transferred to, and the number of bytes of data to transfer. The DMA controller handles the data transfer, and then interrupts the CPU when the transfer is complete.

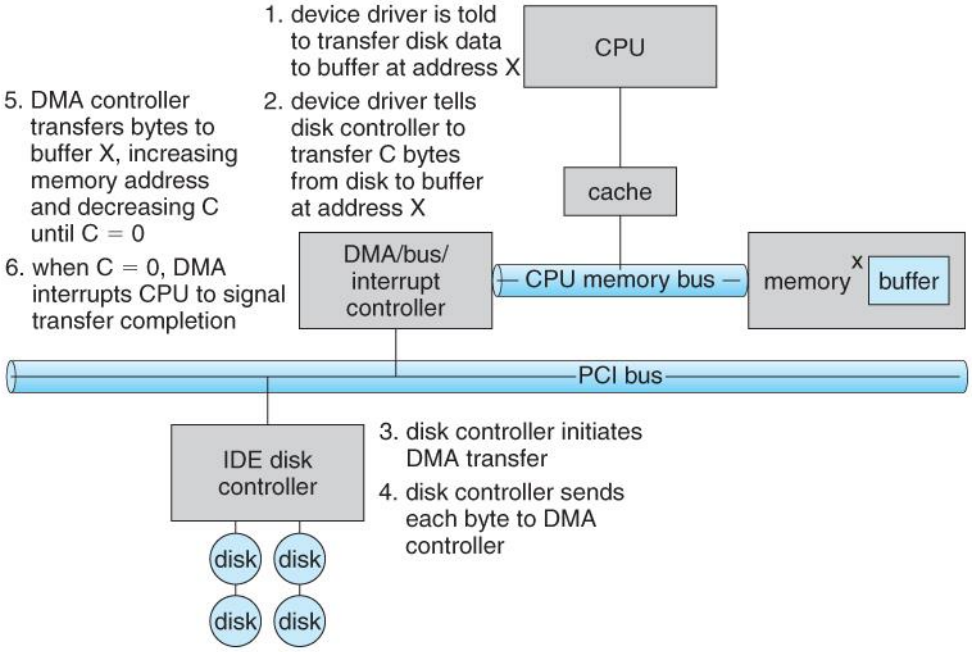
A simple DMA controller is a standard component in modern PCs, and many **bus-mastering** I/O cards contain their own DMA hardware.

Handshaking between DMA controllers and their devices is accomplished through two wires called the DMA-request and DMA-acknowledge wires.

While the DMA transfer is going on the CPU does not have access to the PCI bus (including main memory), but it does have access to its internal registers and primary and secondary caches.

DMA can be done in terms of either physical addresses or virtual addresses that are mapped to physical addresses. The latter approach is known as **Direct Virtual Memory Access, DVMA,** and allows direct data transfer from one memory-mapped device to another without using the main memory chips.

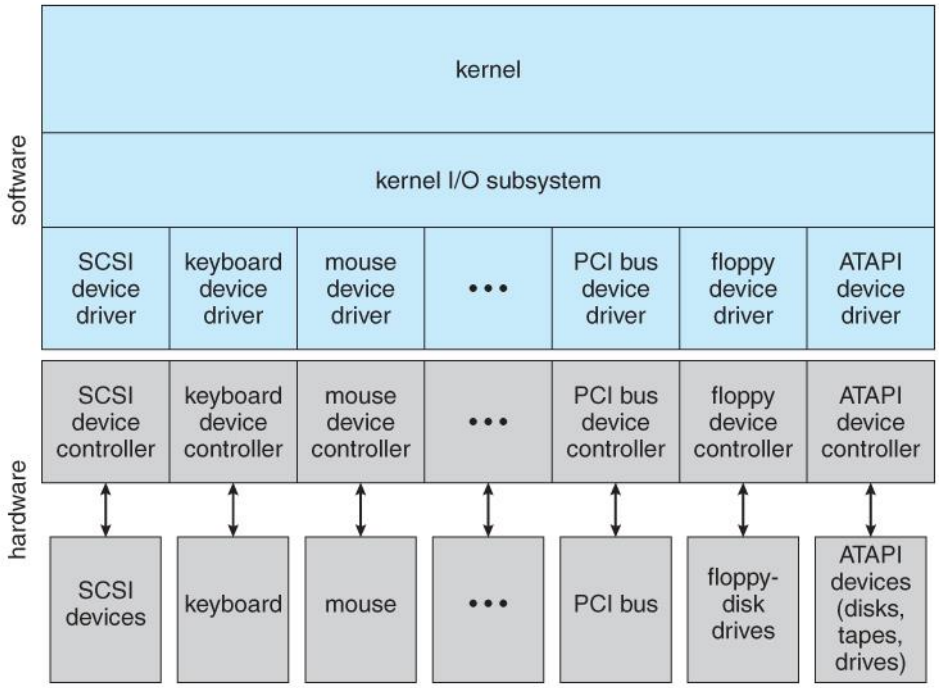
Direct DMA access by user processes can speed up operations, but is generally forbidden by modern systems for security and protection reasons. (i.e. DMA is a kernel-mode operation).



Steps in a DMA transfer

APPLICATION I/O INTERFACE

User application access to a wide variety of different devices is accomplished through layering, and through encapsulating all of the device-specific code into device drivers, while application layers are presented with a common interface for all (or at least large general categories of) devices.



A Kernel I/O Structure

Devices differ on many different dimensions,

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Characteristics of I/O devices

Most devices can be characterized as either block I/O, character I/O, memory mapped file access, or network sockets. A few devices are special, such as time-of-day clock and the system timer.

Most OSes also have an escape, or back door, which allows applications to send commands directly to device drivers if needed. In UNIX this is the ioctl() system call (I/O Control). Ioctl() takes three arguments - The file descriptor for the device driver being accessed, an integer indicating the desired function to be performed, and an address used for communicating or transferring additional information.

Block and Character Devices

Block devices are accessed a block at a time, and are indicated by a "b" as the first character in a long listing on UNIX systems. Operations supported include read( ), write( ), and seek( ).

- Accessing blocks on a hard drive directly (without going through the filesystem structure) is called **raw I/O,** and can speed up certain operations by bypassing the buffering and locking normally conducted by the OS. (It then becomes the application's responsibility to manage those issues).
- A new alternative is **direct I/O,** which uses the normal filesystem access, but which disables buffering and locking operations.

Memory-mapped file I/O can be layered on top of block-device drivers.

- Rather than reading in the entire file, it is mapped to a range of memory addresses, and then paged into memory as needed using the virtual memory system.
- Access to the file is then accomplished through normal memory accesses, rather than through read( ) and write( ) system calls. This approach is commonly used for executable program code.

Character devices are accessed one byte at a time, and are indicated by a "c" in UNIX long listings. Supported operations include get( ) and put( ), with more advanced functionality such as reading an entire line supported by higher-level library routines.

Network Devices

Because network access is inherently different from local disk access, most systems provide a separate interface for network devices.

One common and popular interface is the **socket** interface, which acts like a cable or pipeline connecting two networked entities. Data can be put into the socket at one end, and read out sequentially at the other end. Sockets are normally full-duplex, allowing for bi-directional data transfer.

The select( ) system call allows servers (or other applications) to identify sockets which have data waiting, without having to poll all available sockets.

Clocks and Timers

Three types of time services are commonly needed in modern systems:

- Get the current time of day.
- Get the elapsed time (system or wall clock) since a previous event.
- Set a timer to trigger event X at time T.

Unfortunately time operations are not standard across all systems.

A **programmable interrupt timer, PIT** can be used to trigger operations and to measure elapsed time. It can be set to trigger an interrupt at a specific future time, or to trigger interrupts periodically on a regular basis.

- The scheduler uses a PIT to trigger interrupts for ending time slices.
- The disk system may use a PIT to schedule periodic maintenance cleanup, such as flushing buffers to disk.
- Networks use PIT to abort or repeat operations that are taking too long to complete. I.e. resending packets if an acknowledgement is not received before the timer goes off.
- More timers than actually exist can be simulated by maintaining an ordered list of timer events, and setting the physical timer to go off when the next scheduled event should occur.

On most systems the system clock is implemented by counting interrupts generated by the PIT. Unfortunately this is limited in its resolution to the interrupt frequency of the PIT, and may be subject to some drift over time. An alternate approach is to provide direct access to a high frequency hardware counter, which provides much higher resolution and accuracy, but which does not support interrupts.

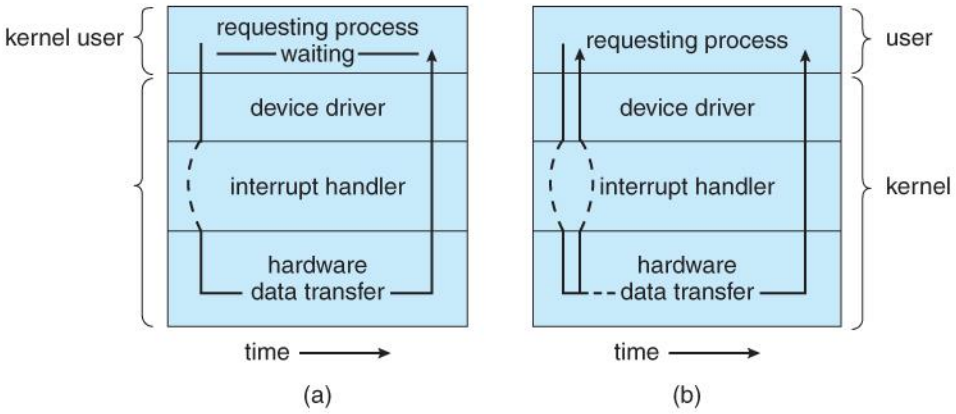
Blocking and Non-blocking I/O

With **blocking I/O** a process is moved to the wait queue when an I/O request is made, and moved back to the ready queue when the request completes, allowing other processes to run in the meantime.

With **non-blocking I/O** the I/O request returns immediately, whether the requested I/O operation has (completely) occurred or not. This allows the process to check for available data without getting hung completely if it is not there.

One approach for programmers to implement non-blocking I/O is to have a multi-threaded application, in which one thread makes blocking I/O calls (say to read a keyboard or mouse), while other threads continue to update the screen or perform other tasks.

A subtle variation of the non-blocking I/O is the **asynchronous I/O,** in which the I/O request returns immediately allowing the process to continue on with other tasks, and then the process is notified (via changing a process variable, or a software interrupt, or a callback function) when the I/O operation has completed and the data is available for use. (The regular non-blocking I/O returns immediately with whatever results are available, but does not complete the operation and notify the process later).



Two I/O methods: (a) synchronous and (b) asynchronous.

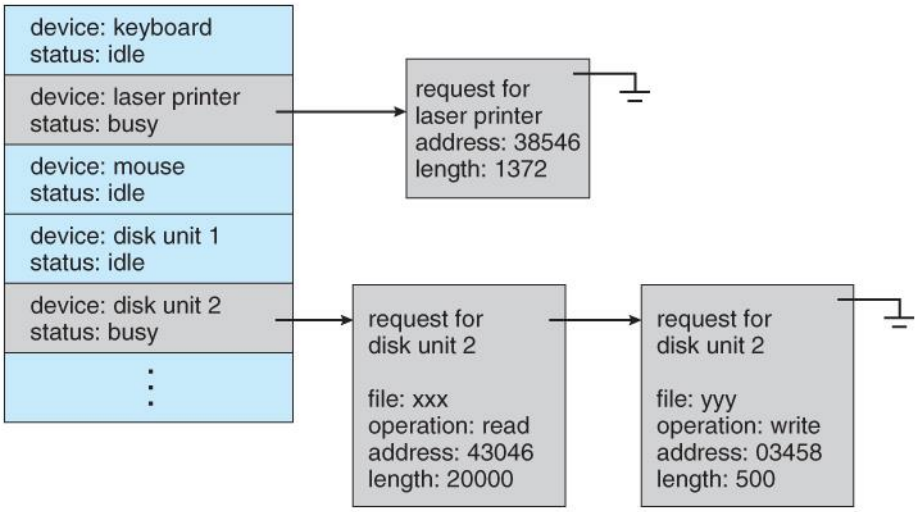
KERNEL I/O SUBSYSTEM

I/O Scheduling

Scheduling I/O requests can greatly improve overall efficiency. Priorities can also play a part in request scheduling.

Buffering and caching can also help, and can allow for more flexible scheduling options.

On systems with many devices, separate request queues are often kept for each device:



Device-status table

Buffering

Buffering of I/O is performed for (at least) 3 major reasons:

- Speed differences between two devices. A slow device may write data into a buffer, and when the buffer is full, the entire buffer is sent to the fast device all at once. So that the slow device still has somewhere to write while this is going on, a second buffer is used, and the two buffers alternate as each becomes full. This is known as **double buffering.** (Double buffering is often used in (animated) graphics, so that one screen image can be generated in a buffer while the other (completed) buffer is displayed on the screen. This prevents the user from ever seeing any half-finished screen images).
- Data transfer size differences. Buffers are used in particular in networking systems to break messages up into smaller packets for transfer, and then for re-assembly at the receiving side.
- To support **copy semantics.** For example, when an application makes a request for a disk write, the data is copied from the user's memory area into a kernel buffer. Now the application can change their copy of the data, but the data which eventually gets written out to disk is the version of the data at the time the write request was made.

Caching

Caching involves keeping a **copy** of data in a faster-access location than where the data is normally stored.

Buffering and caching are very similar, except that a buffer may hold the only copy of a given data item, whereas a cache is just a duplicate copy of some other data stored elsewhere.

Buffering and caching go hand-in-hand, and often the same storage space may be used for both purposes. For example, after a buffer is written to disk, then the copy in memory can be used as a cached copy, (until that buffer is needed for other purposes).

Spooling and Device Reservation

A spool **(Simultaneous Peripheral Operations On-Line)** buffers data for (peripheral) devices such as printers that cannot support interleaved data streams.

If multiple processes want to print at the same time, they each send their print data to files stored in the spool directory. When each file is closed, then the application sees that print job as complete, and the print scheduler sends each file to the appropriate printer one at a time.

Support is provided for viewing the spool queues, removing jobs from the queues, moving jobs from one queue to another queue, and in some cases changing the priorities of jobs in the queues.

Spool queues can be general (any laser printer) or specific (printer number 42).

OSes can also provide support for processes to request / get exclusive access to a particular device, and/or to wait until a device becomes available.

Error Handling

I/O requests can fail for many reasons, either transient (buffers overflow) or permanent (disk crash).

I/O requests usually return an error bit (or more) indicating the problem. UNIX systems also set the global variable errno to one of a hundred or so well-defined values to indicate the specific error that has occurred.

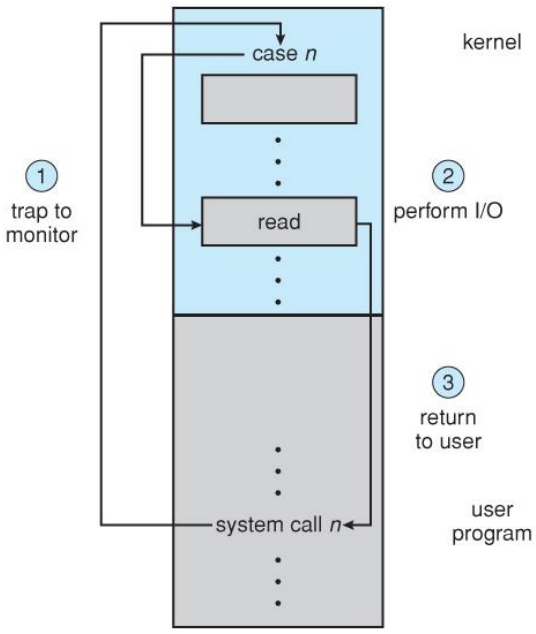
Some devices, such as SCSI devices, are capable of providing much more detailed information about errors, and even keep an on-board error log that can be requested by the host.

I/O Protection

The I/O system must protect against either accidental or deliberate erroneous I/O.

User applications are not allowed to perform I/O in user mode - All I/O requests are handled through system calls that must be performed in kernel mode.

Memory mapped areas and I/O ports must be protected by the memory management system, but access to these areas cannot be totally denied to user programs. (Video games and some other applications need to be able to write directly to video memory for optimal performance for example). Instead the memory protection system restricts access so that only one process at a time can access particular parts of memory, such as the portion of the screen memory corresponding to a particular window.



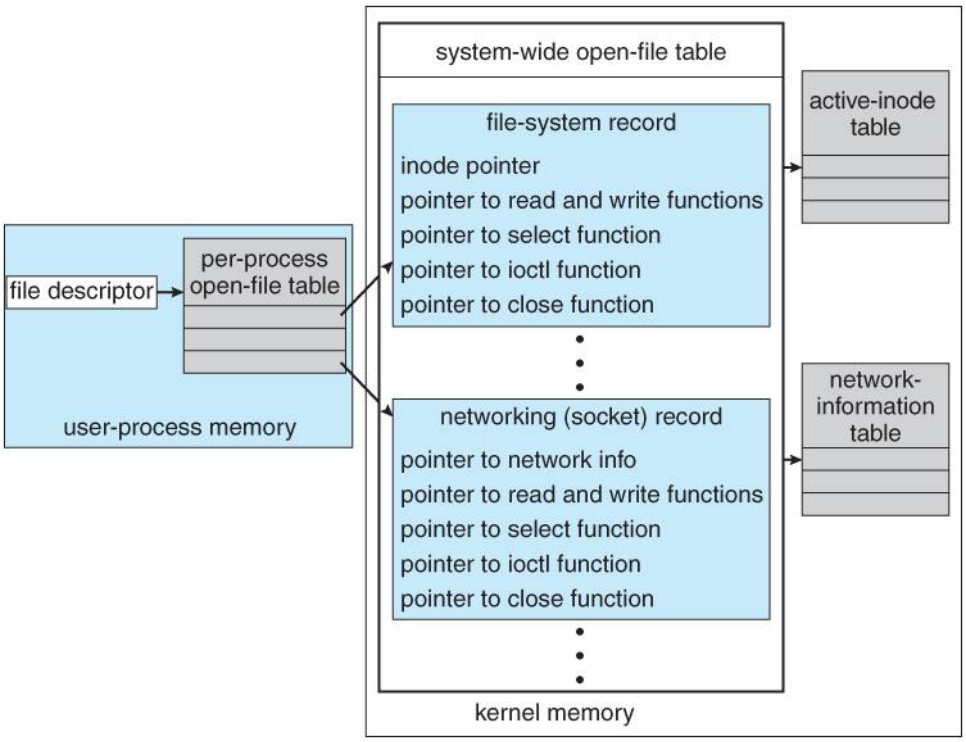
Use of a system call to perform I/O

Kernel Data Structures

The kernel maintains a number of important data structures pertaining to the I/O system, such as the open file table.

These structures are object-oriented, and flexible to allow access to a wide variety of I/O devices through a common interface.

Windows NT carries the object-orientation one step further, implementing I/O as a message-passing system from the source through various intermediaries to the device.



UNIX I/O kernel structure

Transforming I/O Requests to Hardware Operations

Users request data using file names, which must ultimately be mapped to specific blocks of data from a specific device managed by a specific device driver.

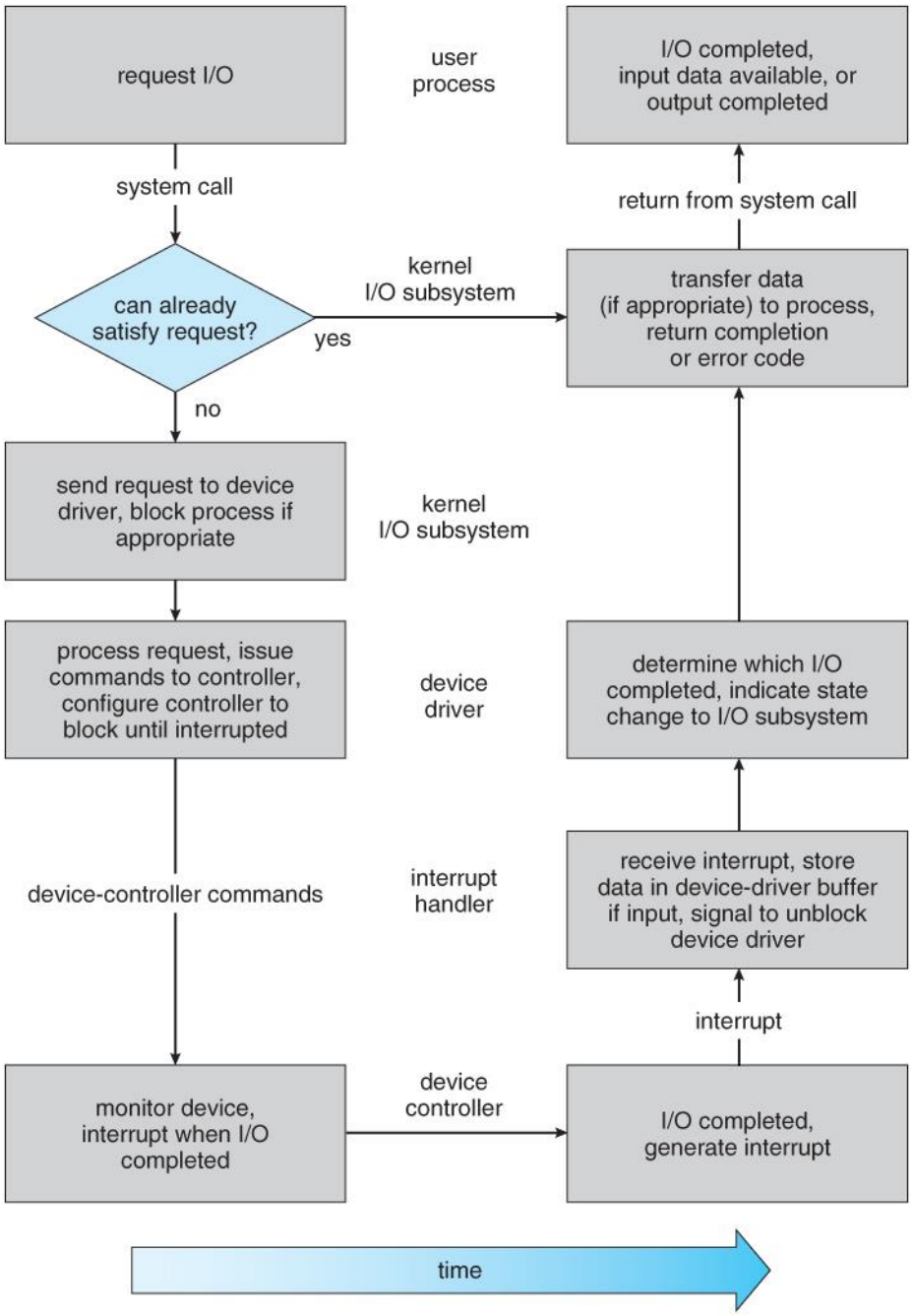
DOS uses the colon separator to specify a particular device (e.g. C:, LPT:, etc.)

UNIX uses a mount table to map filename prefixes (e.g. /usr) to specific mounted devices. Where multiple entries in the mount table match different prefixes of the filename the one that matches the longest prefix is chosen. (e.g. /usr/home instead of /usr where both exist in the mount table and both match the desired file).

UNIX uses special device files, usually located in /dev, to represent and access physical devices directly.

- Each device file has a major and minor number associated with it, stored and displayed where the file size would normally go.
- The major number is an index into a table of device drivers, and indicates which device driver handles this device. (e.g. the disk drive handler).
- The minor number is a parameter passed to the device driver, and indicates which specific device is to be accessed, out of the many which may be handled by a particular device driver. (e.g. a particular disk drive or partition).

A series of lookup tables and mappings makes the access of different devices flexible, and somewhat transparent to users.



Life cycle of an I/O request

TWO MARKS QUESTIONS WITH ANSWERS

1.What is meant by Seek Time? (May/June 2012)

It is the time taken for the disk arm to move the heads to the cylinder containing the desired sector.

2.What is meant by Rotational Latency? (May/June 2012)(Nov/Dec 2010)

It is defined as the additional time waiting for the disk to rotate the desired sector to the disk head.

3.What is meant by Low-level 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 and a trailer.

4.What is meant by Swap-Space Management?

It is a low-level task of the operating system. Efficient management of the swap space is called as Swap space management. This Swap space is the space needed for the entire process image including code and Data segments.

5.What is meant by Disk Scheduling?

Disk scheduling is a process of allocation of the disk to one process at a time. In multi-programmed system, many processes try to read or write the records on disks as the same time. To avoid disk arbitration, it is necessary.

6.Why Disk Scheduling necessary? (April/May 2010)

To avoid Disk arbitration which occurs when many processes try to read or write the records on disks at the same time, Disk Scheduling is necessary.

7.What are the characteristics of Disk Scheduling?

- •Throughput
- •Mean Response Time
- •Variance of Response time

8.What are the different types of Disk Scheduling? (May/June 2014)

Some of the Disk Scheduling are (i) SSTF Scheduling (ii) FCFS Scheduling (iii) SCAN Scheduling (iv) C-SCAN Scheduling (v) LOOK Scheduling (vi) C-LOOK Scheduling

9.What is meant by SSTF Scheduling?

SSTF algorithm selects the request with the minimum seek time from the current head position.
SSTF chooses the pending request to the current head position.

10.What is meant by FCFS Scheduling?

It is simplest form of disk scheduling. This algorithm serves the first come process always and is does not provide fast service.

11.What is meant by SCAN scheduling?

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end of the disk. At the other end, the direction of head movement is reversed and servicing continues across the disk.

12. What is meant by C-SCAN Scheduling?

C-SCAN means Circular SCAN algorithm. This Scheduling is a variant of SCAN designed to provide a more waiting time. This essentially treats the cylinder as a circular list that wraps around from the final cylinder to the first one.

13.Define Throughput.

It is defined as the number of requests serviced per unit time.

14.What is meant by Data Striping?

Data Stripping means splitting the bits of each byte across multiple disks. It is also called as Bit - level Striping.

15.What is meant by Boot Disk?

A Disk that has a boot partition is called as Boot Disk.

16.What are the Components of a Linux System?

Linux System composed of three main modules. They are:

(i) Kernel (ii) System libraries (iii) System utilities

17.What are the main supports for the Linux modules?

The Module support under Linux has three components. They are:

(i) Module Management
(ii) Driver Registration
(iii) Conflict Resolution mechanism.

18.What do you meant by Buffer cache?

It is the kernel's main cache for block-oriented devices such as disk drives and is the main mechanism through which I/O to these devices is performed.

19.What is meant by Kernel in Linux system?

Kernel is responsible for maintaining all the important abstractions of the operating system including such things as virtual memory and processes.

20.What are System Libraries?

System Libraries define a standard set of functions through which applications can interact with the kernel and that implement much of the operating -system functionality that doesn't need the full privileges of kernel code.

21.What are System Utilities?

System Utilities are system programs that perform individual, specialized management tasks. Some of the System utilities may be invoked just to initialize and configure some aspect of the system and others may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals or updating log files.

22.What is the function of Conflict Resolution mechanism?

This mechanism allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

23.What are Device drivers? (Nov/Dec 2013)

Device drivers include (i) Character devices such as printers, terminals (ii) Block devices (including all disk drives) and network interface devices.

24.How do you improve I/O performance? (May/June 2012)

Principles to improve the efficiency of I/O:

a.Reduce the no. of content switches.

b.Reduce the no. of time that data must be copied in memory while passing between device and application.

25.Which scheduling algorithm would be best to optimize the performance of a RAM disk? (Nov/Dec 2011)

Shortest seek time first algorithm.

26.Write the three basic functions which are provided by the hardware clocks and timers. (April/May 2011)

a.Provide current time, elapsed time, timer

b.Programmable interval timer used for timings, periodic interrupts.

c.loctl (on unix) covers odd aspects of I/O such as clocks and timers.

27.What is a file? List some operations on it. (Nov/Dec 2010)

It is a collection of records. The operations performed in it are Open, Close, Read & Write.

28.What is the content of a typical file control block? (April/May 2011)

•File permissions

•File dates (create, access, write)

•File owner, group

•File size

•File data blocks

29.What are File Attributes? (May/June 2012)(April/May 2011)

•Identifier

•Name

•Type, Size

•Location, protection

•Time, Date & User Identification

30.What is meant by stream?

A stream is a full-duplex connection between a device driver and a user-level process. That enables an application to assemble pipelines of driver code dynamically.

It consists of a stream head that interfaces with the user process, a driver end that controls the device, and zero or more stream modules between the stream head and the driver end.

31.State three advantages and disadvantages of placing functionality in a device controller, rather than in the kernel.

Three advantages:

a.Bugs are less likely to cause an operating system crash

b.Performance can be improved by utilizing dedicated hardware and hard-coded algorithms

c.The kernel is simplified by moving algorithms out of it

Three disadvantages:

a.Bugs are harder to fix—a new firmware version or new hardware is needed

b.Improving algorithms likewise require a hardware update rather than just a kernel or device-driver update

c.Embedded algorithms could conflict with application's use of the device, causing decreased performance.

32.How does DMA increase system concurrency? How does it complicate hardware design?

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

33.Distinguish between a STREAMS driver and a STREAMS module.

The STREAMS driver controls a physical device that could be involved in a STREAMS operation.

The STREAMS module modifies the flow of data between the head (the user interface) and the driver.

34.What problems could occur if a system allowed a file system to be mounted simultaneously at more than one location?

There would be multiple paths to the same file, which could confuse users or encourage mistakes (deleting a file with one path deletes the file in all the other paths).

35.Why is the bitmap for file allocation be kept on mass storage, rather than in main memory?

In case of system crash (memory failure) the free-space list would not be lost as it would be if the bit map had been stored in main memory.

36.Consider a system that supports the strategies of contiguous, linked, and indexed allocation. What criteria should be used in deciding which strategy is best utilized for a particular file?

- **Contiguous—**if file is usually accessed sequentially, if file is relatively small.
- **Linked—**if file is large and usually accessed sequentially.
- **Indexed—**if file is large and usually accessed randomly.

SIXTEEN MARK QUESTIONS WITH ANSWERS

1.Explain the various disk scheduling techniques (May/June 2014).

(i) SSTF Scheduling
(ii) FCFS Scheduling
(iii) SCAN Scheduling
(iv) C-SCAN Scheduling
(v) LOOK Scheduling
(vi) C-LOOK Scheduling

2.Write notes about disk management.

- Disk Formatting
- Boot Block
- Bad Blocks

3.Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling

a.FCFS b.SSTF c.SCAN d.LOOK e.C-SCAN f.C- LOOK

Compare the performance of C-SCAN and SCAN scheduling, assuming a uniform distribution of requests. Consider the average response time (the time between the arrival of a request and the completion of that request's service), the variation in response time, and the effective bandwidth. How does performance depend on the relative sizes of seek time and rotational latency?

4.Explain in detail about directory structure.

- Linear List
- Hash Table

5.Explain in detail about file sharing and protection.

File Sharing

- Multiple users
- Remote file systems
- Client server Mode
- Distributed information systems
- Failure modes
- Consistency Semantics
- Immutable shared files semantics

Protection

- Types of Access
- Access Control

6.Explain in detail about file system structure and implementation.

File System Structure

- Description about different Layers

File Implementation

- Description about File Control Block

7. Explain in detail about Allocation methods.

- Contiguous allocation
- Linked allocation
- Indexed allocation

8. Explain in detail about Free space management.

- Bit vector
- Linked list
- Grouping
- Counting
- Space maps

9.Explain in detail about Kernel I/O Structure.

- Kernel Structure
- Dimensions

10.Explain in detail about Kernel I/O Subsystem.

- I/O Scheduling
- Buffering
- Caching
- Spooling
- Device reservation
- Error handling