RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

CSE-Artificial Intelligence and Machine Learning/ Artificial Intelligence and Machine Learning, V-Semester

AL-501Operating Systems

COURSE OBJECTIVES:

To makestudentsunderstand the importance and overall functioning of an Operating System; To acquaint the students with the concepts and principles that underlie the modern Operating Systems, and to provide them an insight in the working of its various modules.

COURSE OUTCOMES:

After completing the course, student should be able to:

1. Get clear understanding about the need and objectives of an Operating System and various services provided by the Operating Systems.
2. Gain a detailed knowledge about the functions of different modules of an Operating System, viz. process management, file system management, memory management, device management etc.
3. Visualize the internal implementation of various modules of Operating System and correlate the same with the actual implementation of these modules in Unix/Linux and other contemporary Operating Systems.
4. Acquire the ability to design and implement small modules of Operating System, Shell and Commands, using system calls of Unix/Linux or some educational Operating System.

COURSE CONTENTS:

UNIT1: **Introduction to Operating Systems**: Function, Evolution, Different types of Operating Systems, Desirable Characteristics and features of an O/S.
Operating Systems Services: Types of Services, Different ways of providing these Services– Commands, System Calls.Need of System Calls, Low level implementation of System Calls, Portability issue, Operating System Structures.

UNIT II: File Systems (Secondary Storage Management): File Concept, User's and System Programmer's view of File System, Hard Disk Organization, Disk Formatting and File System Creation, Different Modules of a File System, Disk Space Allocation Methods – Contiguous, Linked, Indexed. Disk Partitioning andMounting; Directory Structures, File Protection; Virtual and Remote File Systems. Case Studies of File Systems being used in Unix/Linux & Windows; System Calls used in these Operating Systems for file management.

UNITIII: Process Management: Concept of a process, Process State Diagram, Different type of schedulers, CPU scheduling algorithms, Evaluation of scheduling algorithms, Concept of Threads: User level & Kernel level Threads, Thread Scheduling;Multiprocessor/Multicore Processor Scheduling.Case Studies of Process Management in Unix/Linux& Windows; System Calls used in these Operating Systems for

Process Management.
Concurrency& Synchronization: Real and Virtual Concurrency, Mutual Exclusion, Synchronization, Critical Section Problem, Solution to Critical Section Problem: Mutex Locks; Monitors; Semaphores,WAIT/SIGNAL operations and their implementation;Classical Problems of Synchronization;Inter-Process Communication.
Deadlocks: Deadlock Characterization, Prevention, Avoidance, Recovery.

UNIT IV: **Memory Management**: Different Memory Management Techniques –Contiguous allocation; Non-contiguous allocation: Paging, Segmentation, Paged Segmentation; Comparison of these techniques.
Virtual Memory – Concept, Overlay, Dynamic Linking and Loading, Implementation of Virtual Memory by Demand Paging etc.; Memory Management in Unix/Linux& Windows.

UNIT V: Input/Output Management: Overview of Mass Storage Structures, Disk Scheduling;I/O Systems: Different I/O Operations- Program Controlled, Interrupt Driven, Concurrent I/O, Synchronous/Asynchronous and Blocking/Non-Blocking I/O Operations, I/O Buffering,Application I/O Interface, Kernel I/O Subsystem, Transforming I/O requests to hardware operations.

Overview of Protection & Security Issues and Mechanisms; Introduction to Multiprocessor, Real Time, Embedded and Mobile Operating Systems; Overview of Virtualization.

TEXTBOOKSRECOMMENDED:

1. Silberschatz, Galvin, Gagne, “Operating System Concepts’’,John Wiley& Sons.
2. William Stalling, “Operating Systems: Internals and Design Principles”, Pearson.

REFERENCEBOOKS:

1. Andrew S. Tanenbaum, “Modern Operating Systems”, Prentice Hall.
2. Robert Love, “Linux Kernel Development”, Pearson.
3. Maurice J. Bach, “The Design of Unix Operating System”, Pearson.
4. Bovet &Cesati, “Understanding the Linux Kernel”, O'Reilly.

CS405 Operating Systems

UNIT-I

Operating System

An Operating System (OS) is an interface between computer user and computer hardware. An operating system is software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Examples-Some Popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

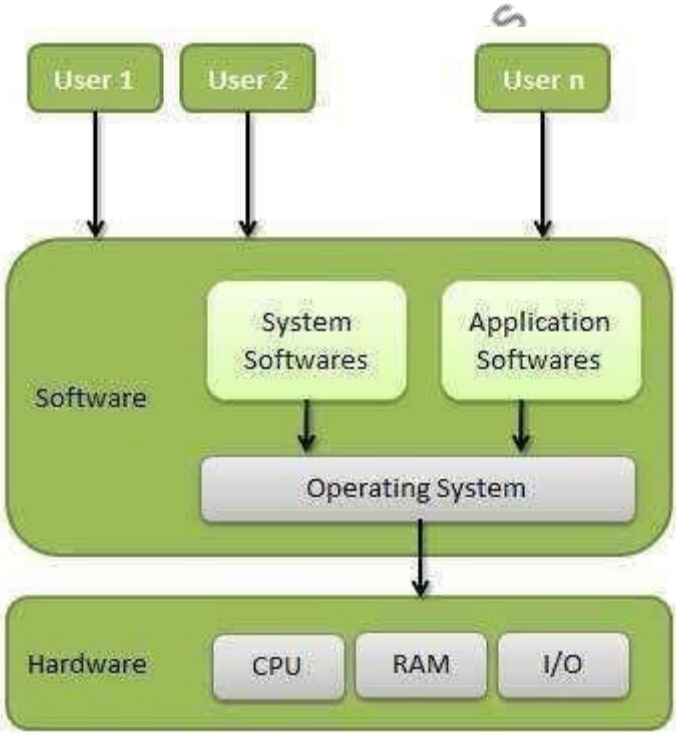


Fig 1.1Operating System

Desirable Characteristics & features of an Operating System

Following are some of important features of an operating System.

- Memory Management

- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part is not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Other Important Characteristics

Following are some of the important activities that an Operating System performs −

- **Security**− By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance**− Recording delays between request for a service and response from the system.
- **Job accounting**− Keeping track of time and resources used by various jobs and users.
- **Error detecting aids**− Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other software's and users**− Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Types of Operating System

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multi-programmed Batch Systems and Time-Sharing Systems is that in case of Multi-programmed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if nusers are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So, in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, scientific experiments, medical image systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard

real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects likes undersea exploration and planetary rovers, etc.

Operating Systems Services: Types of Services

Operating system services are responsible for the management of platform resources, including the processor, memory, files, and input and output. They generally shield applications from the implementation details of the machine. Types of Operating system services include:

1. **Kernel operations provide low-level services necessary to:**

- create and manage processes and threads of execution
- execute programs
- define and communicate asynchronous events
- define and process system clock operations
- implement security features
- manage files and directories, and
- Control input/output processing to and from peripheral devices.
- Some kernel services have analogues described in the paragraph on Object Services, such as concurrency control services.

2. **Command interpreter and utility services** include mechanisms for services at the operator level, such as:

- comparing, printing, and displaying file contents
- editing files
- searching patterns
- evaluating expressions
- logging messages
- moving files between directories
- sorting data
- executing command scripts
- local print spooling
- scheduling signal execution processes, and
- Accessing environment information.

3. **Batch processing services** support the capability to queue work (jobs) and manage the sequencing of processing based on job control commands and lists of data. These services also include support for the management of the output of batch processing, which frequently includes updated files or databases and information products such as printed reports or electronic documents. Batch processing is performed asynchronously from the user requesting the job.

4. **File and directory synchronization** services allow local and remote copies of files and directories to be made identical. Synchronization services are usually used to update files after periods of off line working on a portable system.

Operating System – Important Services

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system –

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection
- Resource Allocation
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management –

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, and data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.
Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

Different ways of providing these Services –

- **Utility Programs**
  A program performs very specific tasks.
  These programs usually related to managing system resources. Operating systems contain a number of utilities for managing disk drives, printers, and other devices. Utilities differ from applications mostly in terms of size, complexity and function. For example, word processors, spreadsheet programs, and database applications are considered applications because they are large programs that perform a variety of functions not directly related to managing computer resources. Utilities are sometimes installed as memory-resident programs.
  Examples of utility programs are antivirus software, backup software and disk tools.

- **System Calls**

A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface (API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Services Provided by System Calls:

- Process creation and management
- Main memory management
- File Access, Directory and File system management
- Device handling(I/O)
- Protection
- Networking, etc.

Types of System Calls: There are 5 different categories of system calls –

- Process control: end, abort, create, terminate, allocate and free memory.
- File management: create, open, close, delete, read file etc.
- Device management
- Information maintenance
- Communication

Operating System - Properties

Batch processing

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts. An operating system does the following activities related to batch processing –

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a job in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.

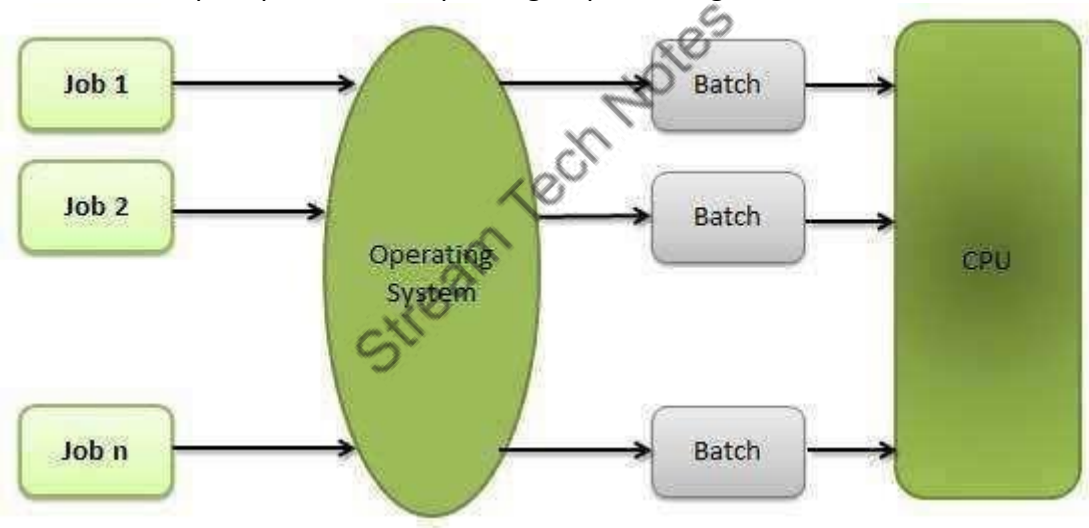


Fig 1.2Batch processing

Advantages

- Batch processing takes much of the work of the operator to the computer.
- Increased performance as a new job gets started as soon as the previous job is finished, without any manual intervention.

Disadvantages

- Difficult to debug program.

- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Multitasking

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking −

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.

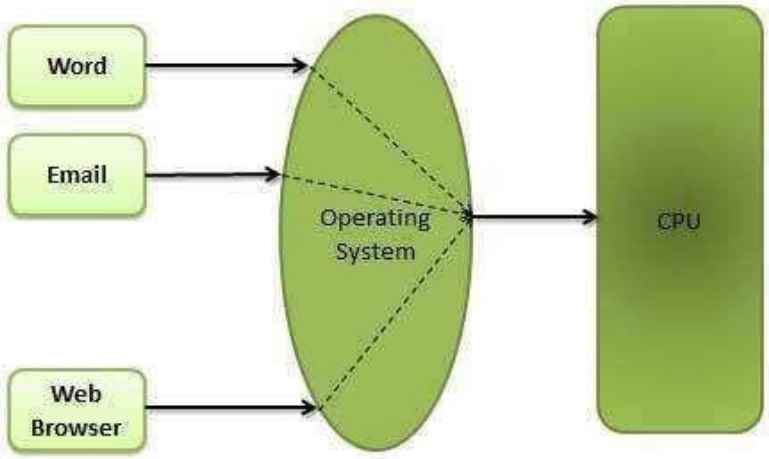


Fig 1.3Multitasking

- A program that is loaded into memory and is executing is commonly referred to as a **process**.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU, rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred as **multiprogramming**. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.

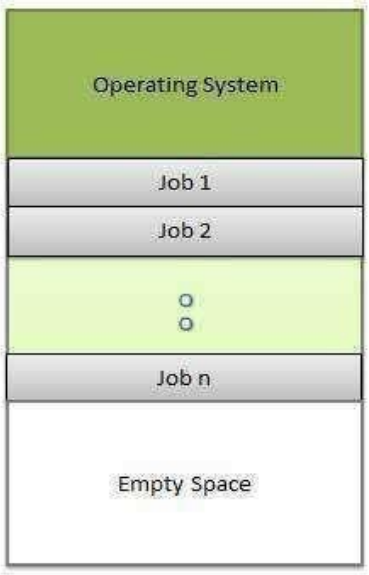


Fig 1.4 Multiprogramming

An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.
- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensure that the CPU is never idle, unless there are no jobs to process.

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

Interactivity

Interactivity refers to the ability of users to interact with a computer system. An Operating system does the following activities related to interactivity −

- Provides the user an interface to interact with the system.
- Manages input devices to take inputs from the user. For example, keyboard.
- Manages output devices to show outputs to the user. For example, Monitor.

The response time of the OS needs to be short, since the user submits and waits for the result.

Real Time System

Real-time systems are usually dedicated, embedded systems. An operating system does the following activities related to real-time system activity.

- In such systems, Operating Systems typically read from and react to sensor data.
- The Operating system must guarantee response to events within fixed periods of time to ensure correct performance.

Distributed Environment

A distributed environment refers to multiple independent CPUs or processors in a computer system. An operating system does the following activities related to distributed environment −

- The OS distributes computation logics among several physical processors.
- The processors do not share memory or a clock. Instead, each processor has its own local memory.
- The OS manages the communications between the processors. They communicate with each other through various communication lines.

Spooling

Spooling is an acronym for simultaneous peripheral operations on line. Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices.

An operating system does the following activities related to distributed environment −

- Handles I/O device data spooling as devices have different data access rates.
- Maintains the spooling buffer which provides a waiting station where data can rest while the slower device catches up.
- Maintains parallel computation because of spooling process as a computer can perform I/O in parallel fashion. It becomes possible to have the computer read data from a tape, write data to disk and to write out to a tape printer while it is doing its computing task.

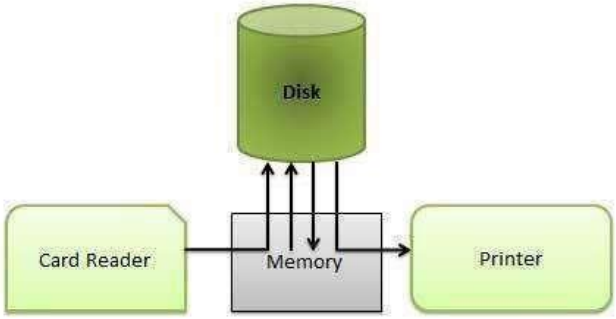


Fig 1.5 Spooling

CS-405 Operating System

UNIT-II

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

File Structure

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. UNIX, MS-DOS support minimum number of file structure.

File Type

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

Ordinary files

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

Directory files

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

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

- **Character special files**− data is handled character by character as in case of terminals or printers.
- **Block special files**− data is handled in blocks as in the case of disks and tapes.

User's and System Programmer's view of File System

User View

The user view of the computer refers to the interface being used. Such systems are designed for one user to monopolize its resources, to maximize the work that the user is performing. In these cases, the operating system is designed mostly for ease of use, with some attention paid to performance, and none paid to resource utilization.

System View

Operating system can be viewed as a resource allocator also. A computer system consists of many resources like - hardware and software - that must be managed efficiently. The operating system acts as the manager of the resources, decides between conflicting requests, controls execution of programs etc.

Disk Organization

A hard disk is a memory storage device which looks like this:

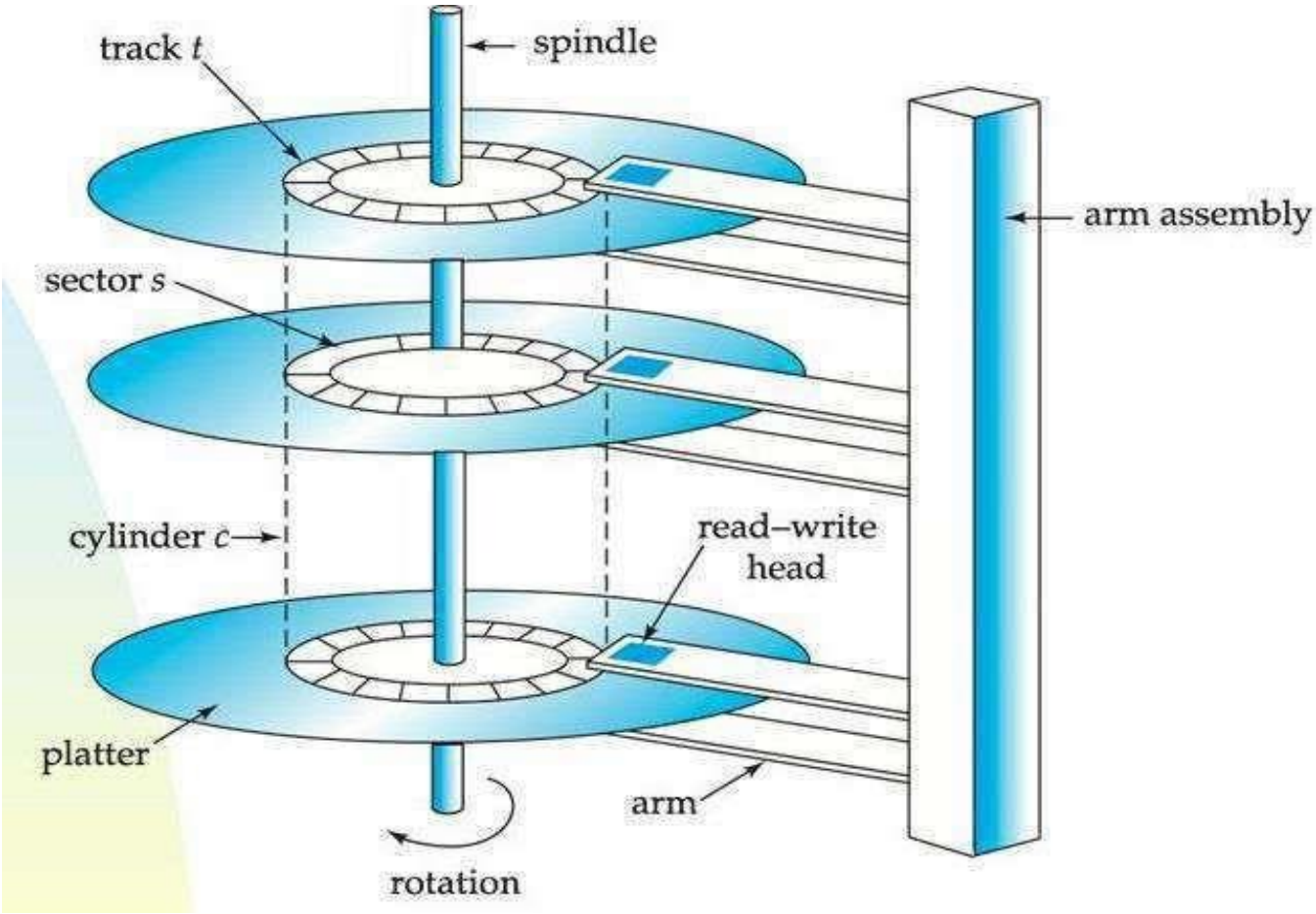


Fig 2.0 Hard disk

The disk is divided into **tracks**. Each track is further divided into **sectors**. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity. This is because the storage density is high in sectors of the inner tracks where as the bits are sparsely arranged in sectors of the outer

tracks. Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity.

Read-Write(R-W) head moves over the rotating hard disk. It is this Read-Write head that performs all the read and write operations on the disk and hence, position of the R-W head is a major concern. To perform a read or write operation on a memory location, we need to place the R-W head over that position. Some important terms must be noted here:

1. **Seek time –** The time taken by the R-W head to reach the desired track from it's current position.
2. **Rotational latency –** Time taken by the sector to come under the R-W head.
3. **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
4. **Controller time –** The processing time taken by the controller.
5. **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time.

Different Modules of a File System:

- **The basic file system level** works directly with the device drivers in terms of retrieving and storing raw blocks of data, without any consideration for what is in each block. Depending on the system, blocks may be referred to with a single block number or with head-sector-cylinder combinations.
- **The file organization module** knows about files and their logical blocks, and how they map to physical blocks on the disk. In addition to translating from logical to physical blocks, the file organization module also maintains the list of free blocks, and allocates free blocks to files as needed.
- **The logical file system** deals with all of the meta data associated with a file (UID, GID, mode, dates, etc), i.e. everything about the file except the data itself. This level manages the directory structure and the mapping of file names to file control blocks, FCBs, which contain all of the Meta data as well as block number information for finding the data on the disk.
- The layered approach to file systems means that much of the code can be used uniformly for a wide variety of different file systems, and only certain layers need to be file system specific. Common file systems in use include the UNIX file system, UFS, the Berkeley Fast File System, FFS, Windows systems FAT, FAT32, NTFS, CD-ROM systems ISO 9660, and for Linux the extended file systems ext2 and ext3 .

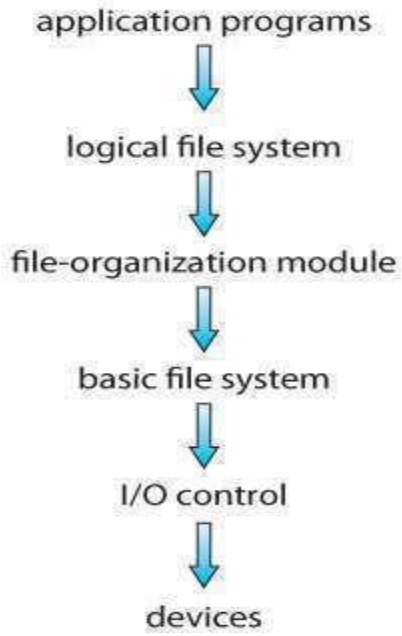


Fig 2.1 - Layered file system.

File system in Linux & Windows

Linux

Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel**– Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library**– System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.
- **System Utility**– System Utility programs are responsible to do specialized, individual level tasks.

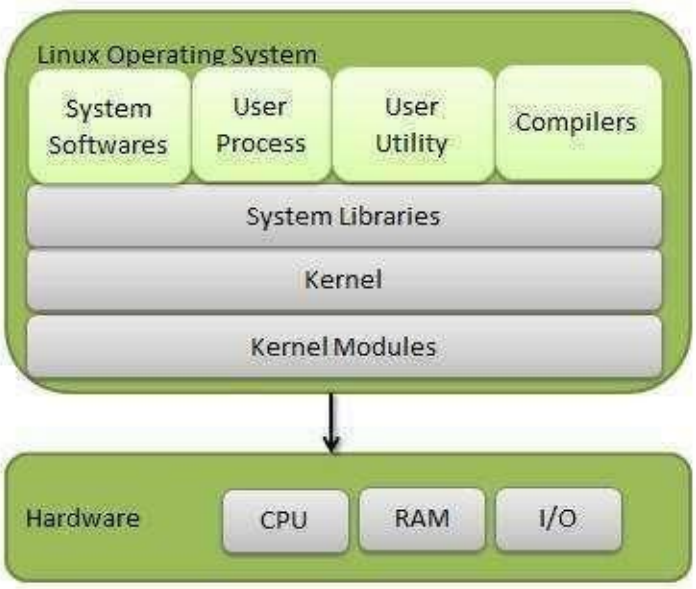


Fig 2.2 Linux Operating System

Directory systems

A directory is a location for storing files on your computer. Directories are found in a hierarchical file system, such as Linux, MS-DOS, OS/2, and Unix.

- A collection of nodes containing information about all files
- A directory system can be classified in to single level and hierarchical directory system:

Single level directory system: In this type of directory system, there is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. This type of directory system is used in cameras and phones.

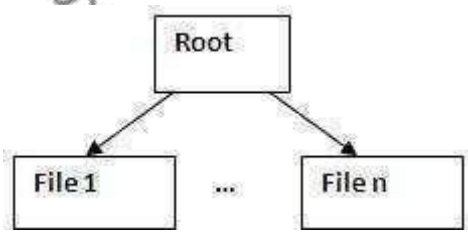


Fig 2.3 Directory System

Hierarchical directory system: In a hierarchical directory system, files are grouped together to form a sub directory at the top of the hierarchy is the root directory and then there are sub directories which has files. Advantage of hierarchical directory system is that users can be provided access to a sub directory rather than the entire directory. It provides a better structure to file system. Also, managing millions of files is easy with this architecture. Personal computers use hierarchical directory system for managing files.

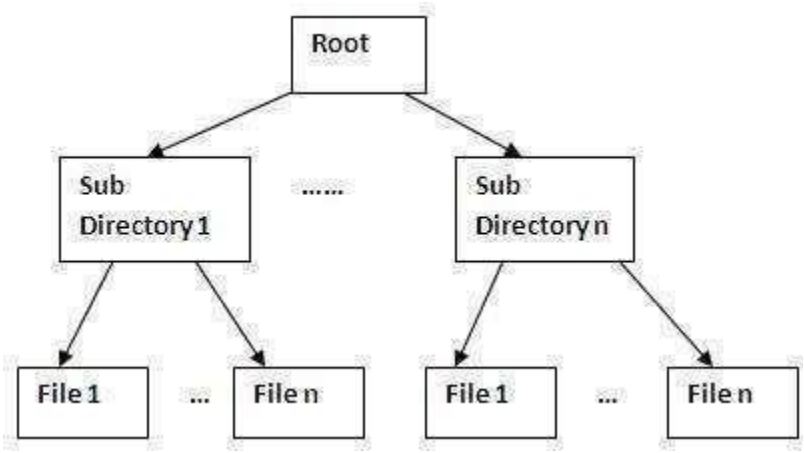


Fig 2.4 Hierarchical directory system

Disk Space Allocation Methods:

There are three major methods of storing files on disks: contiguous, linked, and indexed.

Contiguous Allocation

- ***Contiguous Allocation*** requires that all blocks of a file be kept together contiguously.
- Performance is very fast, because reading successive blocks of the same file generally requires no movement of the disk heads, or at most one small step to the next adjacent cylinder.
- Storage allocation involves the same issues discussed earlier for the allocation of contiguous blocks of memory (first fit, best fit, fragmentation problems, etc.) The distinction is that the high time penalty required for moving the disk heads from spot to spot may now justify the benefits of keeping files contiguously when possible.
- (Even file systems that do not by default store files contiguously can benefit from certain utilities that compact the disk and make all files contiguous in the process.)
- Problems can arise when files grow, or if the exact size of a file is unknown at creation time:
    - Over-estimation of the file's final size increases external fragmentation and wastes disk space.
    - Under-estimation may require that a file be moved or a process aborted if the file grows beyond its originally allocated space.
    - If a file grows slowly over a long time period and the total final space must be allocated initially, then a lot of space becomes unusable before the file fills the space.
- A variation is to allocate file space in large contiguous chunks, called ***extents.*** When a file outgrows its original extent, then an additional one is allocated. (For example an extent may be the size of a complete track or even cylinder, aligned on an appropriate track or cylinder boundary.) The high-performance files system Veritas uses extents to optimize performance.

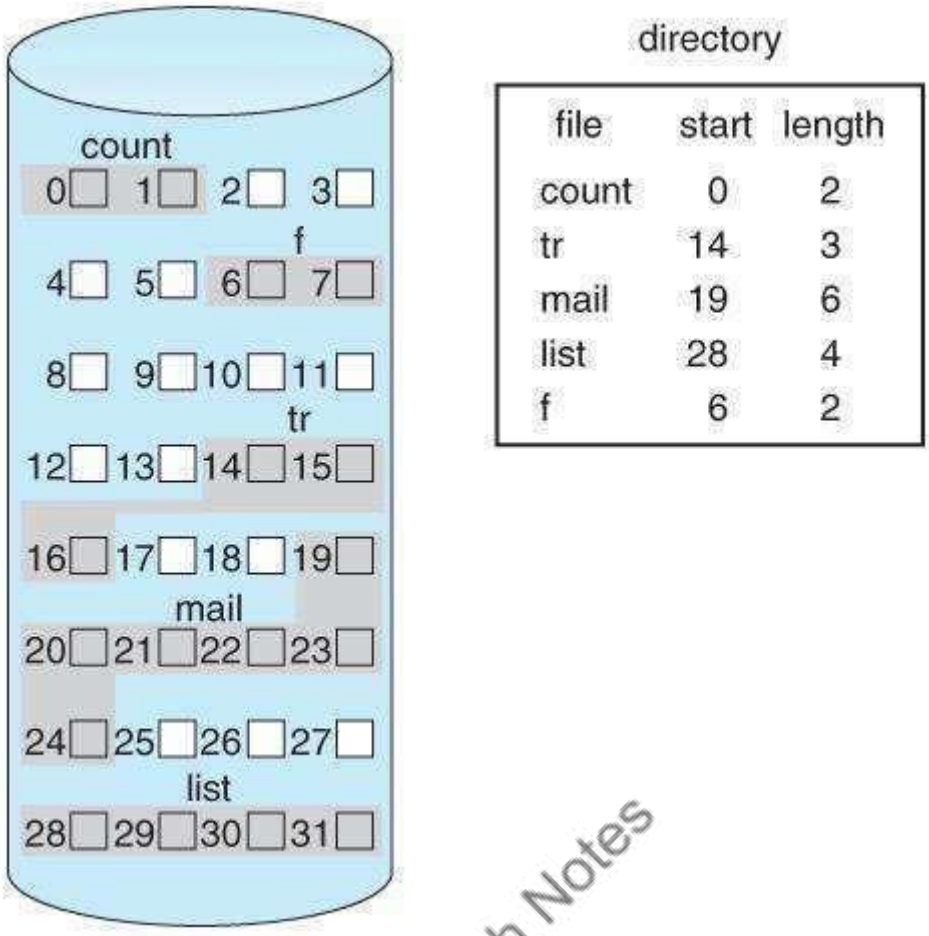


Fig 2.5 Contiguous allocation of disk space.

Linked Allocation

- Disk files can be stored as linked lists, with the expense of the storage space consumed by each link. (E.g. a block may be 508 bytes instead of 512.)
- Linked allocation involves no external fragmentation, does not require pre-known file sizes, and allows files to grow dynamically at any time.
- Unfortunately linked allocation is only efficient for sequential access files, as random access requires starting at the beginning of the list for each new location access.
- Allocating ***clusters*** of blocks reduces the space wasted by pointers, at the cost of internal fragmentation.
- Another big problem with linked allocation is reliability if a pointer is lost or damaged. Doubly linked lists provide some protection, at the cost of additional overhead and wasted space.

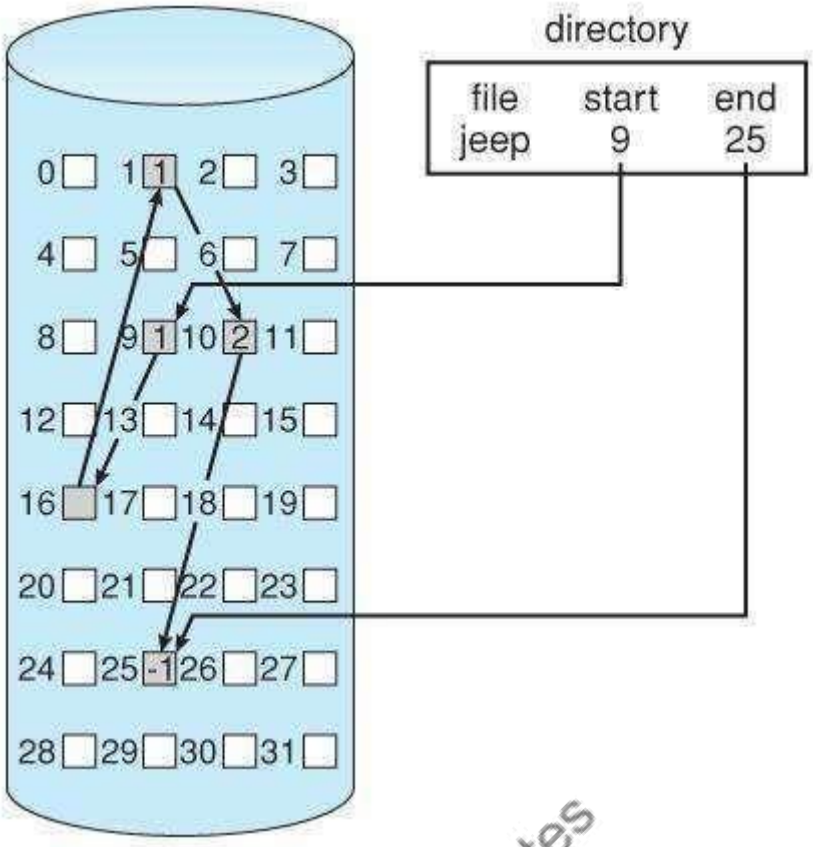


Fig 2.6 - Linked allocation of disk space.

- The ***File Allocation Table, FAT,*** used by DOS is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the disk. The benefit of this approach is that the FAT table can be cached in memory, greatly improving random access speeds.

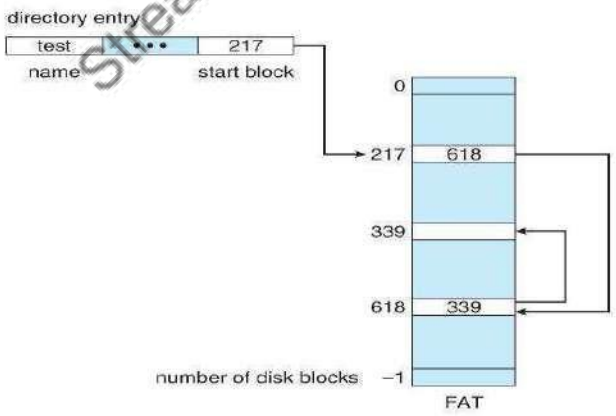


Fig 2.7 File-allocation table.

Indexed Allocation

Indexed Allocation combines all of the indexes for accessing each file into a common block (for that file), as opposed to spreading them all over the disk or storing them in a FAT table.

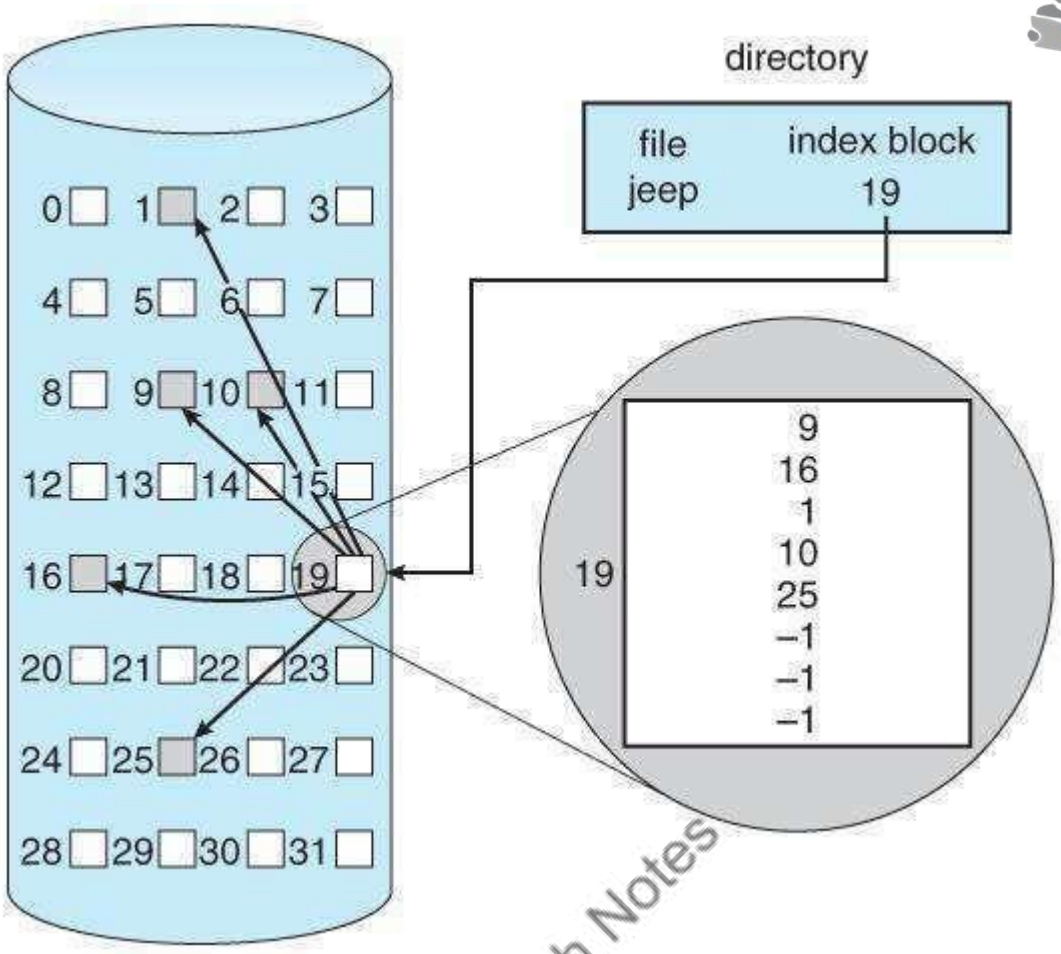


Fig 2.8 - Indexed allocation of disk space.

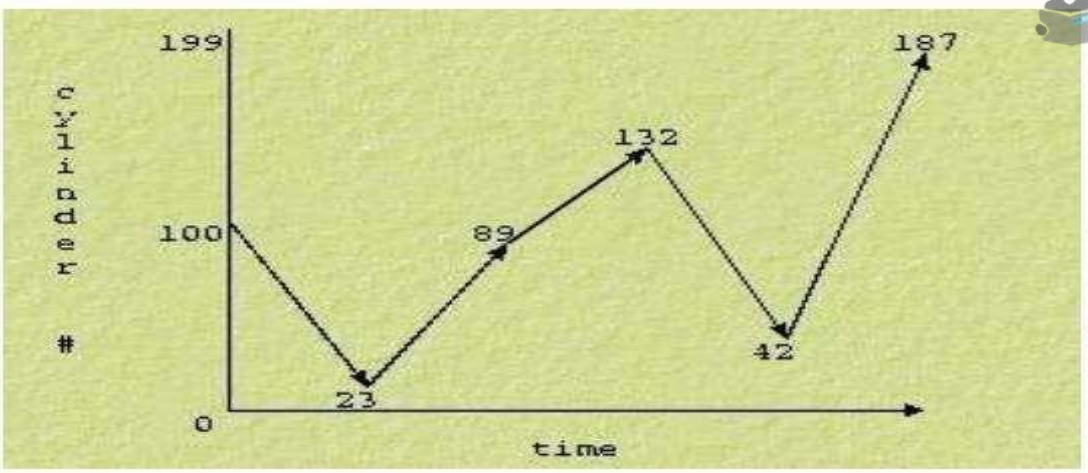
Disk Scheduling Algorithms

I/O request issues a system call to the OS. If desired disk drive or controller is available, request is served immediately. If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the OS has to choose which pending request to service next.

FCFS Scheduling

Simplest, perform operations in order requested no reordering of work queue „ no starvation: every request is serviced „ Doesn't provide fastest service Ex: a disk queue with requests for I/O to blocks on cylinders 23, 89, 132, 42, 187 with disk head initially at 100

FCFS 23, 89, 132, 42, 187



77+66+43+90+145=421

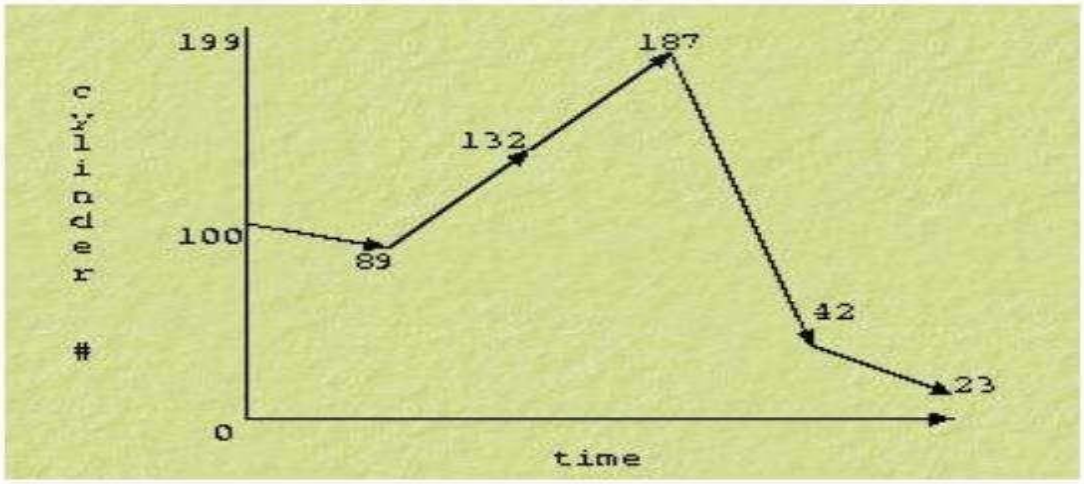
Fig 2.9 FCFS Scheduling

If the requests for cylinders 23 and 42 could be serviced together, total head movement could be decreased substantially.

SSTF Scheduling

Like SJF, select the disk I/O request that requires the least movement of the disk arm from its current position, regardless of direction reduces total seek time compared to FCFS. Disadvantages starvation is possible; stay in one area of the disk if very busy switching directions slow things down not the most optimal.

23, 89, 132, 42, 187



11+43+55+145+19=273

Fig 2.10 SSTF Scheduling

SCAN

Go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests. Sometimes called the elevator algorithm Reduces variance compared to SSTF. If a request arrives in the queue just in front of the head ‰ Just behind

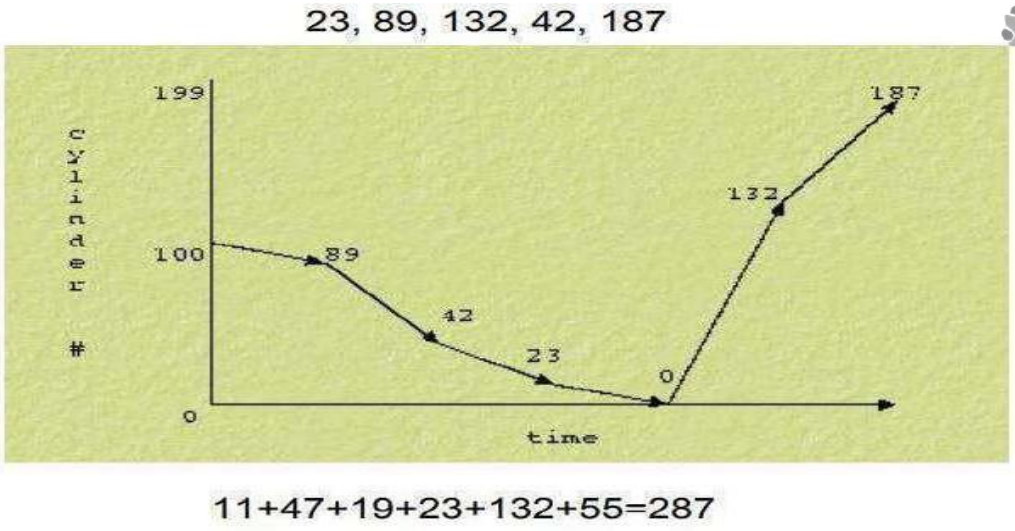


Fig 2.11 SCAN Scheduling

C-SCAN

Circular SCAN „ moves inwards servicing requests until it reaches the innermost cylinder; then jumps to the outside cylinder of the disk without servicing any requests. „ Why C-SCAN? ‰ Few requests are in front of the head, since these cylinders have recently been serviced. Hence provides a more uniform wait time.

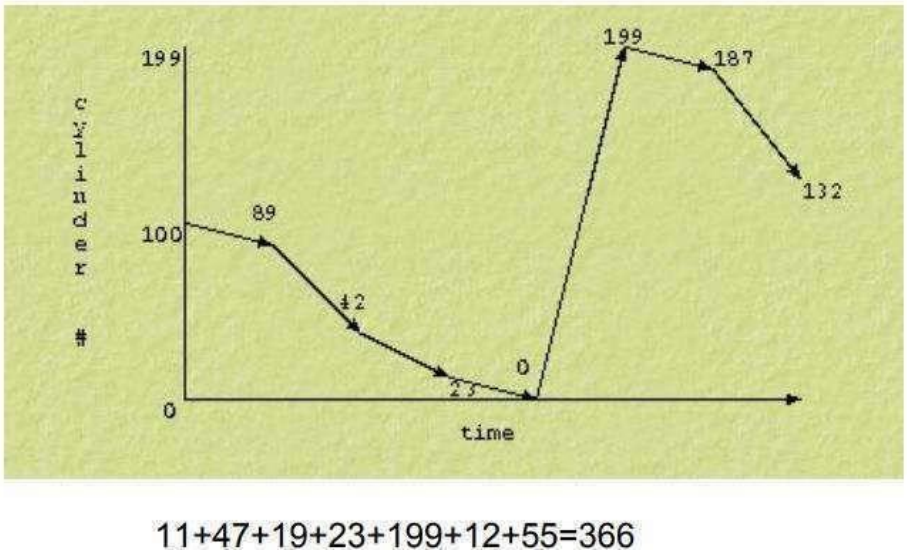


Fig 2.12 C-SCAN Scheduling

LOOK

Like SCAN but stops moving inwards (or outwards) when no more requests in that direction exist.

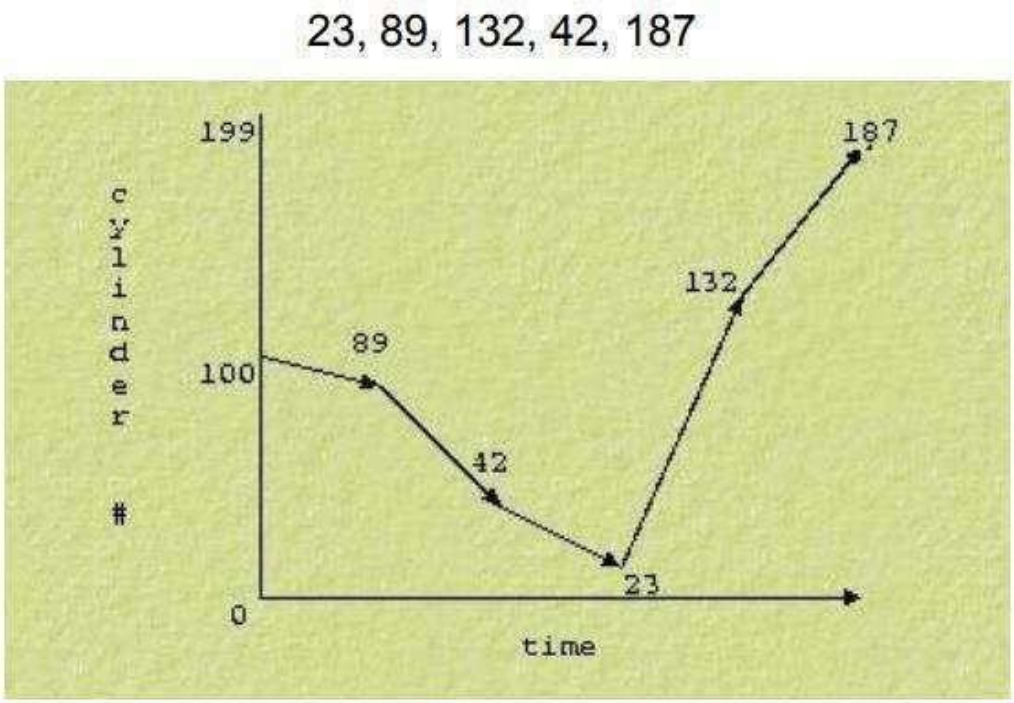


Fig 2.13 LOOK Scheduling

Compared to SCAN, LOOK saves going from 23 to 0 and then back. Most efficient for this sequence of requests

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. Bugs in the file-system software can also cause file contents to be lost. Protection can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multiuser system, however, other mechanisms are needed.

System calls for File Management

- **System call OPEN**

  Opening or creating a file can be done using the system call open. The syntax is:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path,
         int flags,... /* mode_t mod */);
```

- **System call CREAT**

  A new file can be created by:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(const char *path, mode_t mod);
```

- **System call READ**

  When we want to read a certain number of bytes starting from the current position in a file, we use the *read* call. The syntax is:

```
#include <unistd.h>
ssize_t read(int fd, void* buf, size_t noct);
```

- **System call WRITE**

  For writing a certain number of bytes into a file starting from the current position we use the *write* call. Its syntax is:

```
#include <unistd.h>
ssize_t write(int fd, const void* buf, size_t noct);
```

- **System call CLOSE**

  For closing a file and thus eliminating the assigned descriptor we use the system call *close*.

```
#include <unistd.h>
int close(int fd);
```

- **System call LSEEK**

  To position a pointer that points to the current position in an absolute or relative way can be done by calling the *lseek* function. Read and write operations are done relative to the current position in the file. The syntax for *lseek* is:

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int ref);
```


- **System call LINK**

  To link an existing file to another directory (or to the same directory) link can be used. To make such a link in fact means to set a new name or a path to an existing file. The *link* system call creates a hard link. Creating symbolic links can be done using *symlink* system call. The syntax of link is:

```
#include <unistd.h>
int link(const char* oldpath, const char* newpath);
int symlink(const char* oldpath, const char* newpath);
```

UNIT III

Processes Concept

Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory

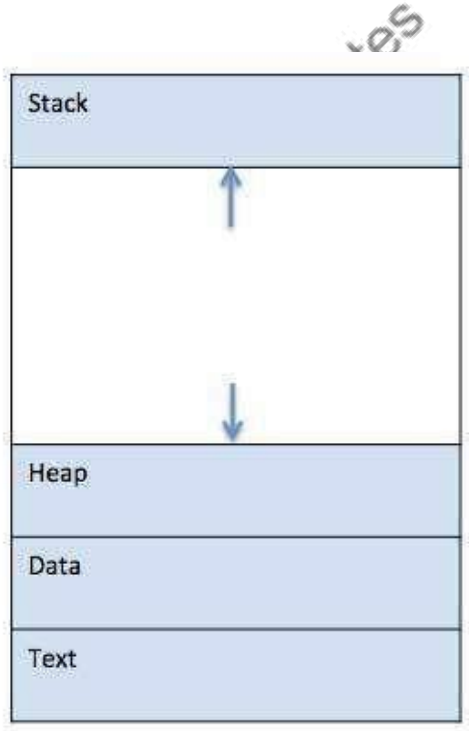


Fig 3.1 Process

S.N.	Component & Description
1	**Stack -** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap -** This is dynamically allocated memory to a process during its run time.
3	**Text -** This includes the current activity represented by the value of Program Counter

	and the contents of the processor's registers.
4	**Data -** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include<stdio.h>


intmain () {

printf("Hello, World! \n");

return0;

}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as software.

Scheduling Concepts

Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue**− this queue keeps all the processes in the system.
- **Ready queue**− this queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

- **Device queues**– the processes which are blocked due to unavailability of an I/O device constitute this queue.

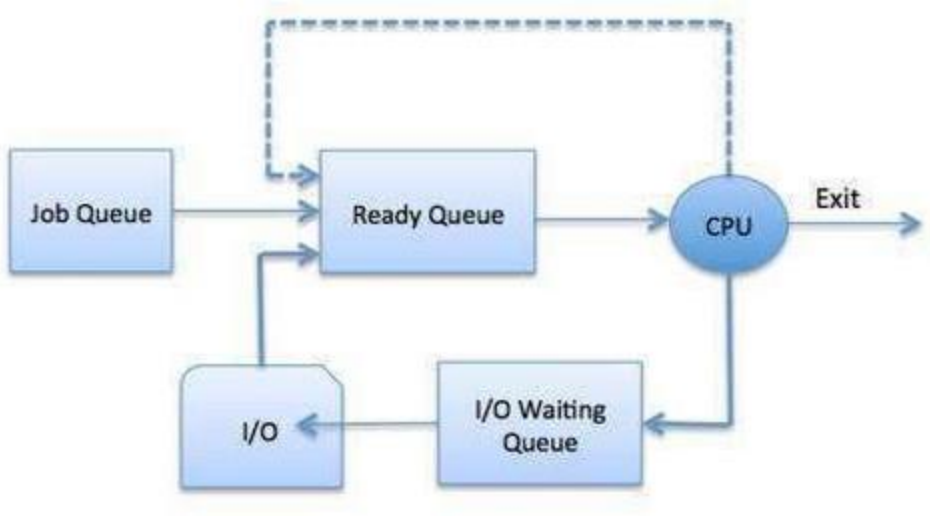


Fig 3.2 Process Scheduling Queues

The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below –

S.N.	State & Description
1	**Running -** When a new process is created, it enters into the system as in the running state.
2	**Not Running -** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Types of Schedulers

Schedulers are special system software which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types –

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.
The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long-term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing	It is also minimal in time sharing system	It is a part of Time sharing systems.

	system		
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

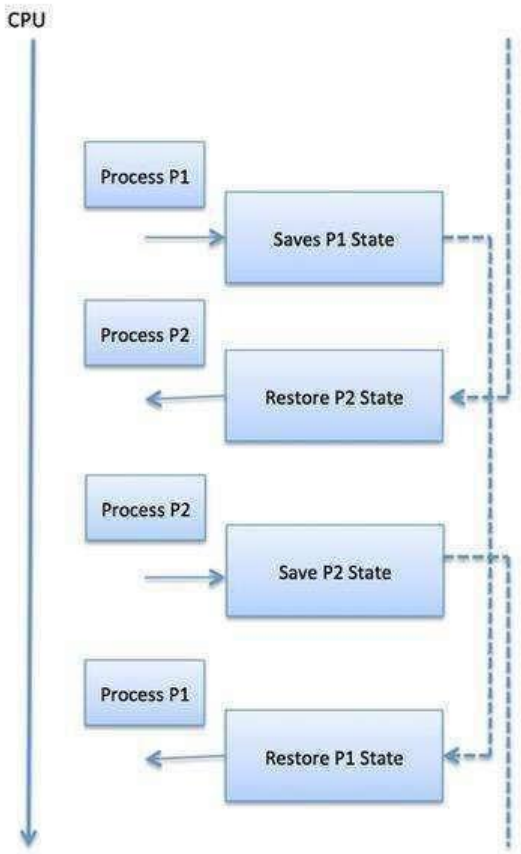


Fig 3.3 Context Switch

Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information

- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

Process Life Cycle & Process State diagram

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start -** This is the initial state when a process is first started/ created.
2	**Ready -** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running -** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting -** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit -** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

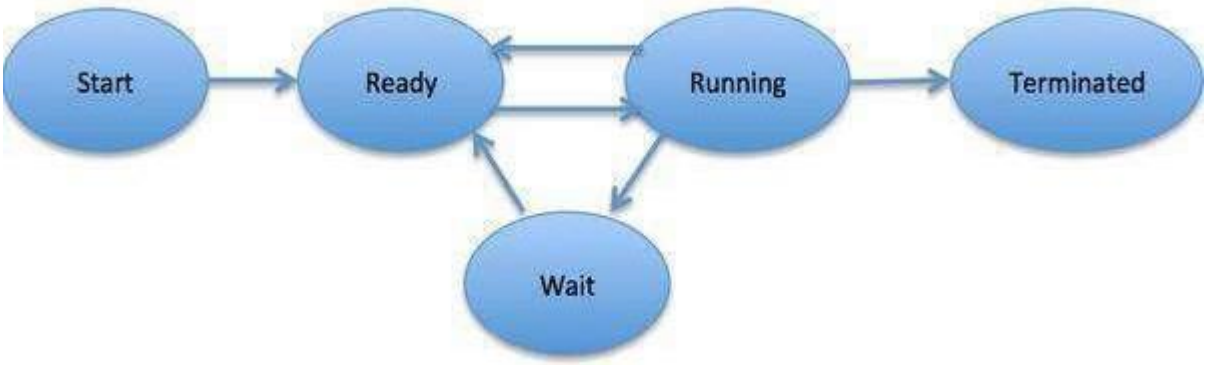


Fig 3.4 Process State Diagram

Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

S.N.	Information & Description
1	**Process State -** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges -** This is required to allow/disallow access to system resources.
3	**Process ID -** Unique identification for each of the process in the operating system.
4	**Pointer -** A pointer to parent process.
5	**Program Counter -** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers -** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information -** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information -** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information -** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information -** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB –

Fig 3.5 Process Control Block (PCB)

The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Scheduling Algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms

- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling
- First-Come, First-Served (FCFS) Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

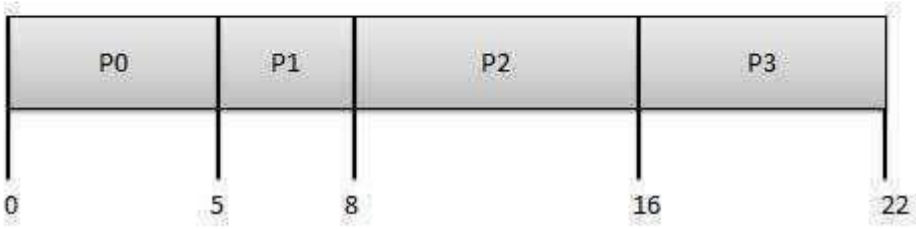


Fig 3.6 First Come First Serve (FCFS)

Wait time of each process is as follows −

Process	**Wait Time: Service Time - Arrival Time**
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6

P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processes should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8

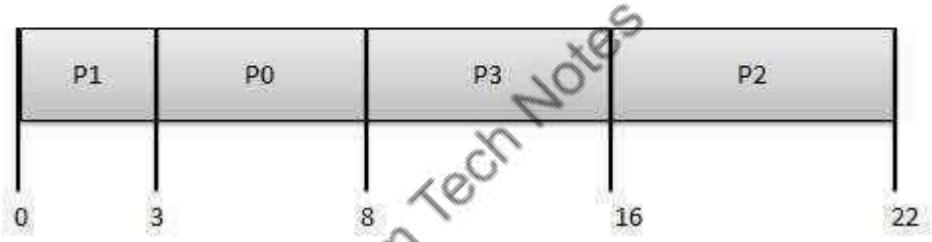


Fig 3.7 Shortest Job Next (SJN)

Wait time of each process is as follows −

Process	**Wait Time: Service Time - Arrival Time**
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

Average Wait Time: (3+0+14+5) / 4 = 5.50

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0

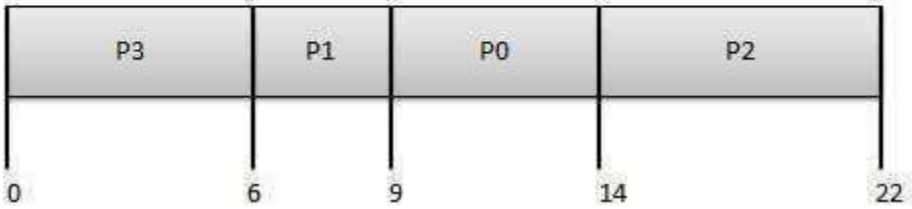


Fig 3.8 Priority Based Scheduling

Wait time of each process is as follows –

Process	Wait Time: Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

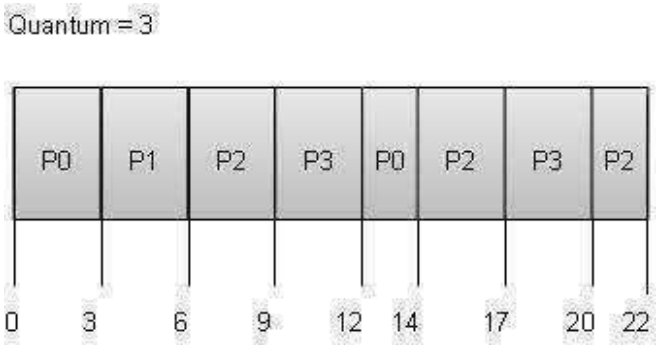


Fig 3.9 Round Robin Scheduling

Wait time of each process is as follows −

Process	Wait Time: Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Algorithm Evaluation

How do we select a CPU scheduling algorithm for a particular system?

There are many scheduling algorithms, each with its own parameters. As a result, selecting an algorithm can be difficult. The first problem is defining the criteria to be used in selecting an algorithm. Criteria are often defined in terms of CPU utilization, response time, or throughput. To select an algorithm, we must first define the relative importance of these measures. Our criteria may include several measures, such as:

- Maximizing CPU utilization under the constraint that the maximum response time is 1 second
- Maximizing throughput such that turnaround time is (on average) linearly proportional to total execution time once the selection criteria have been defined, we want to evaluate the algorithms under consideration. We next describe the various evaluation methods we can use.

Deterministic Modeling

One major class of evaluation methods is analytic evaluation. Analytic evaluation uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload. One type of analytic evaluation is deterministic modeling. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload. For example, assume that we have the workload shown below. All five processes arrive at time 0, in the order given, with the length of the CPU burst given in milliseconds:

Queuing Models

Another method of evaluating scheduling algorithms is to use queuing theory. Using data from real processes we can arrive at a probability distribution for the length of a burst time and the I/O times for a process. We can now generate these times with a certain distribution.

We can also generate arrival times for processes (arrival time distribution).

If we define a queue for the CPU and a queue for each I/O device we can test the various scheduling algorithms using queuing theory.

Knowing the arrival rates and the service rates we can calculate various figures such as average queue length, average wait time, CPU utilization etc.

One useful formula is L**ittle's Formula.**

$n = \lambda w$

Where

n is the average queue length
λ is the average arrival rate for new processes (e.g. five a second)
w is the average waiting time in the queue

Knowing two of these values we can, obviously, calculate the third. For example, if we know that eight processes arrive every second and there are normally sixteen processes in the queue we can compute that the average waiting time per process is two seconds.

The main disadvantage of using queuing models is that it is not always easy to define realistic distribution times and we have to make assumptions. This results in the model only being an approximation of what actually happens.

Simulations

Rather than using queuing models we simulate a computer. A Variable, representing a clock is incremented. At each increment the state of the simulation is updated.
Statistics are gathered at each clock tick so that the system performance can be analyzed.

The data to drive the simulation can be generated in the same way as the queuing model, although this leads to similar problems.

Alternatively, we can use trace data. This is data collected from real processes on real machines and is fed into the simulation. This can often provide good results and good comparisons over a range of scheduling algorithms.
However, simulations can take a long time to run, can take a long time to implement and the trace data may be difficult to collect and require large amounts of storage.

Implementation

The best way to compare algorithms is to implement them on real machines. This will give the best results but does have a number of disadvantages.

- It is expensive as the algorithm has to be written and then implemented on real hardware.
- If typical workloads are to be monitored, the scheduling algorithm must be used in a live situation. Users may not be happy with an environment that is constantly changing.
- If we find a scheduling algorithm that performs well there is no guarantee that this state will continue if the workload or environment changes.

System Calls for Process Management

Basic process management is done with a number of system calls, each with a single (simple) purpose. These system calls can then be combined to implement more complex behaviors.

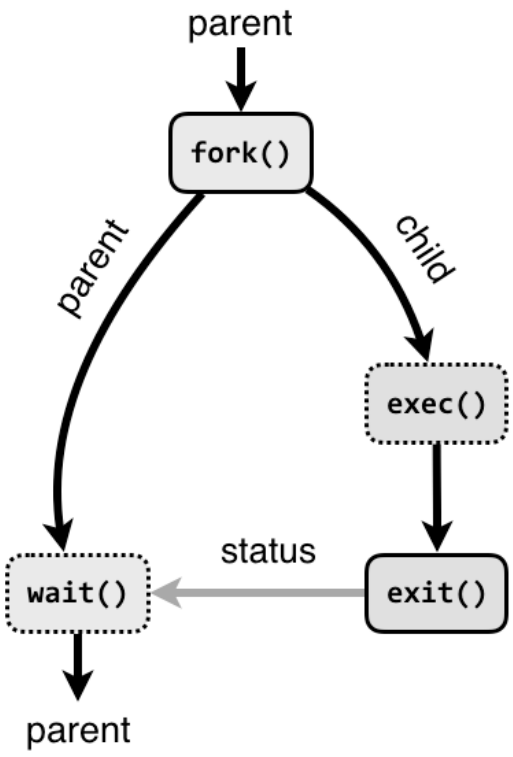


Fig: 3.10 System Calls for Process Management

The following system calls are used for basic process management.

- **fork :**A parent process uses fork to create a new child process. The child process is a copy of the parent. After fork, both parent and child executes the same program but in separate processes.
- **exec:** Replaces the program executed by a process. The child may use exec after a fork to replace the process' memory space with a new program executable making the child execute a different program than the parent.
- **exit:** Terminates the process with an exit status.
- **wait:** The parent may use wait to suspend execution until a child terminates. Using wait the parent can obtain the exit status of a terminated child.

Multiple Processor Schedulers:

In multiple-processor scheduling **multiple CPU's** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality; we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the Master Server and the other processors executes only the user code. This is simple and reduces the need of data sharing. This entire scenario is called Asymmetric Multiprocessing.
A second approach uses Symmetric Multiprocessing where each processor is self scheduling. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

Processor Affinity

Processor Affinity means a process has an affinity for the processor on which it is currently running. When a process runs on a specific processor there are certain effects on the cache memory. The data most recently accessed by the process populate the cache for the processor and as a result successive memory accesses by the process are often satisfied in the cache memory. Now if the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated. Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor. This is known as **PROCESSOR AFFINITY**.

- **Soft Affinity –** When an operating system has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.
- **Hard Affinity –** Some systems such as Linux also provide some system calls that support Hard Affinity which allows a process to migrate between processors.

Thread

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads little information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

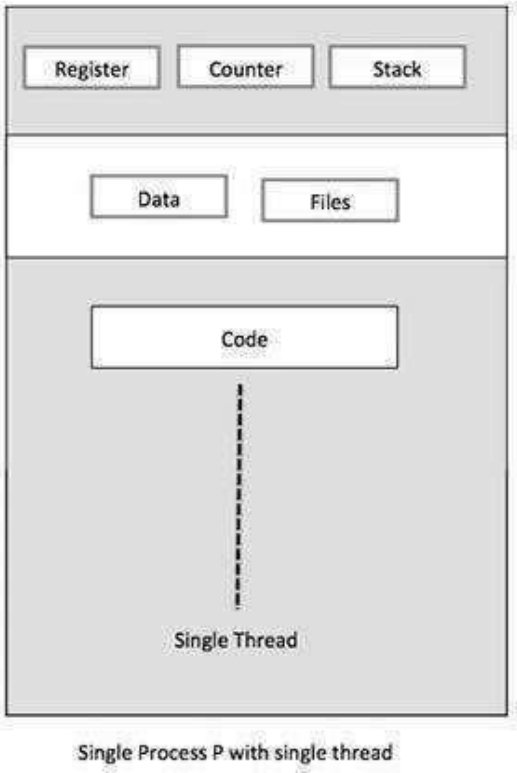


Fig 3.11 Thread

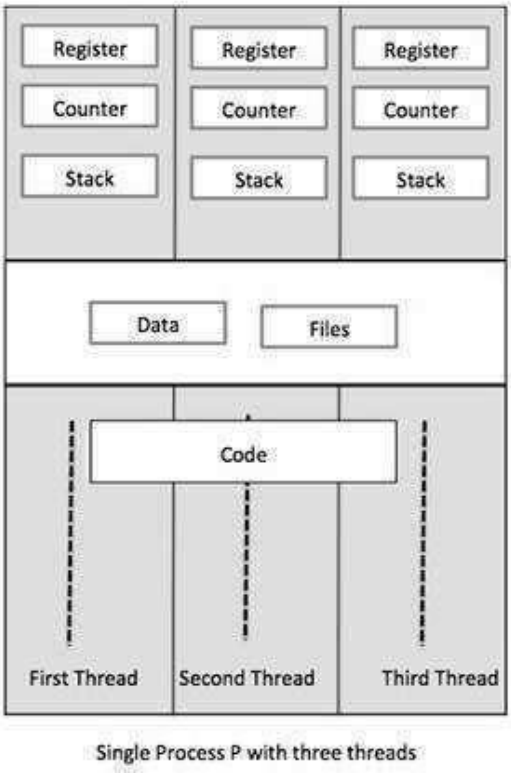


Fig 3.12 Thread

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked, and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes, each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

Threads are implemented in following two ways –

- **User Level Threads**– User managed threads.
- **Kernel Level Threads**– Operating System managed threads acting on kernel, an operating system core.

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

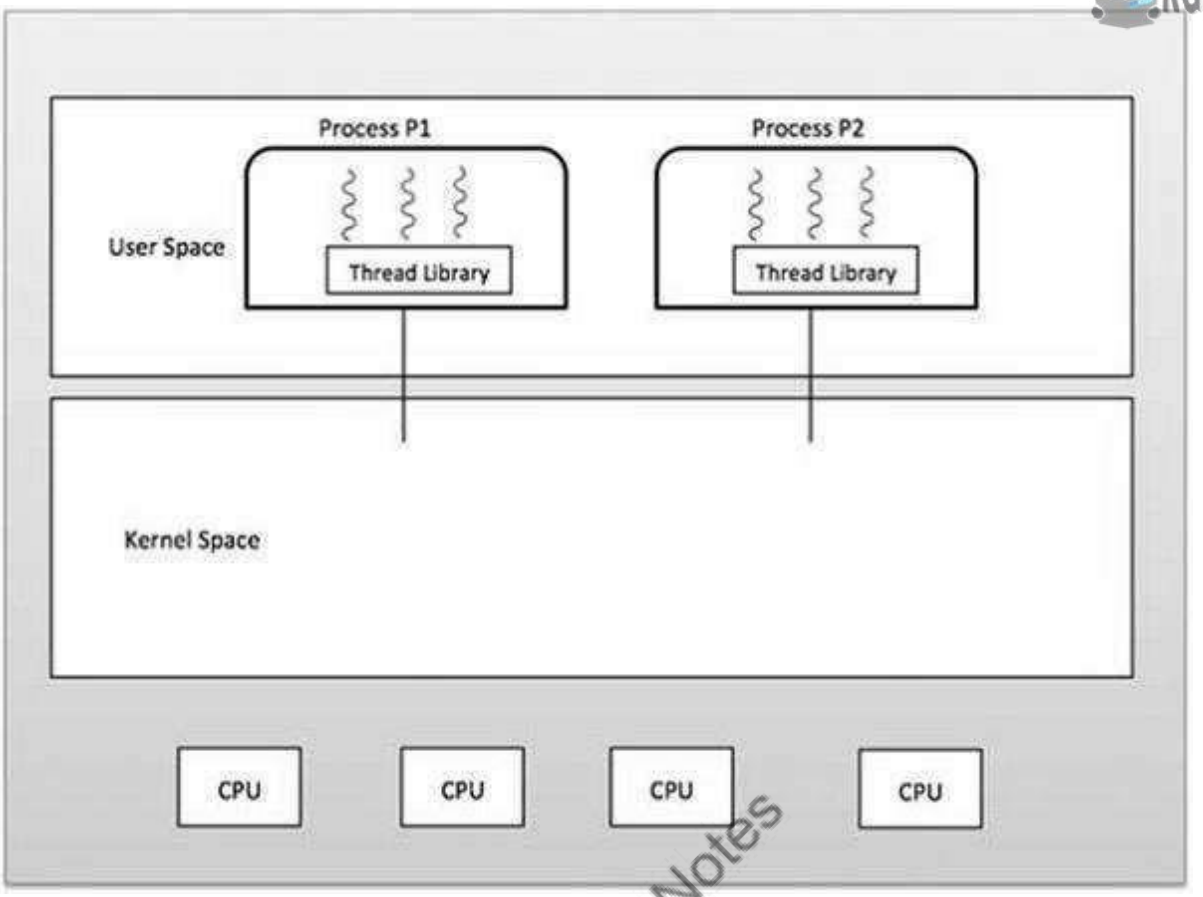


Fig 3.13 User Level Threads

Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual's threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Multithreading Models

Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many too many relationships.
- Many to one relationship.
- One to one relationship.

Many too Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

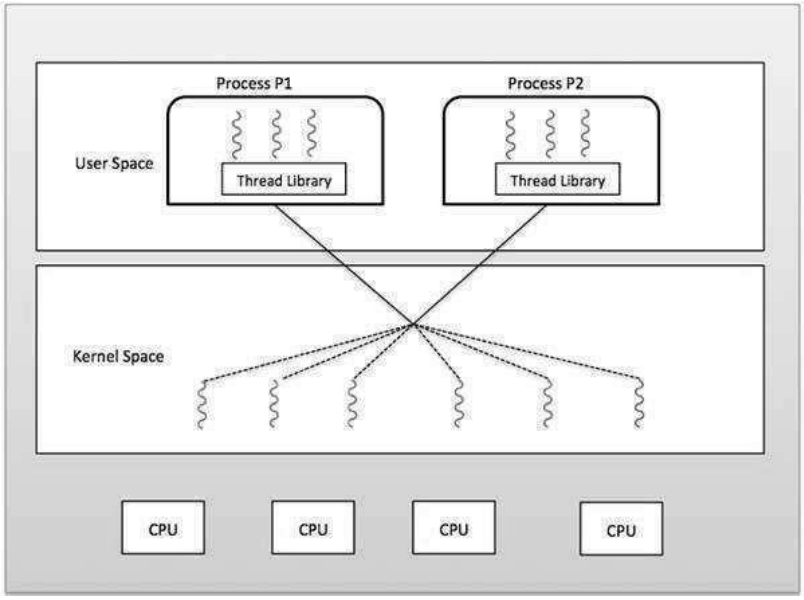


Fig 3.14 Many too Many Model

Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

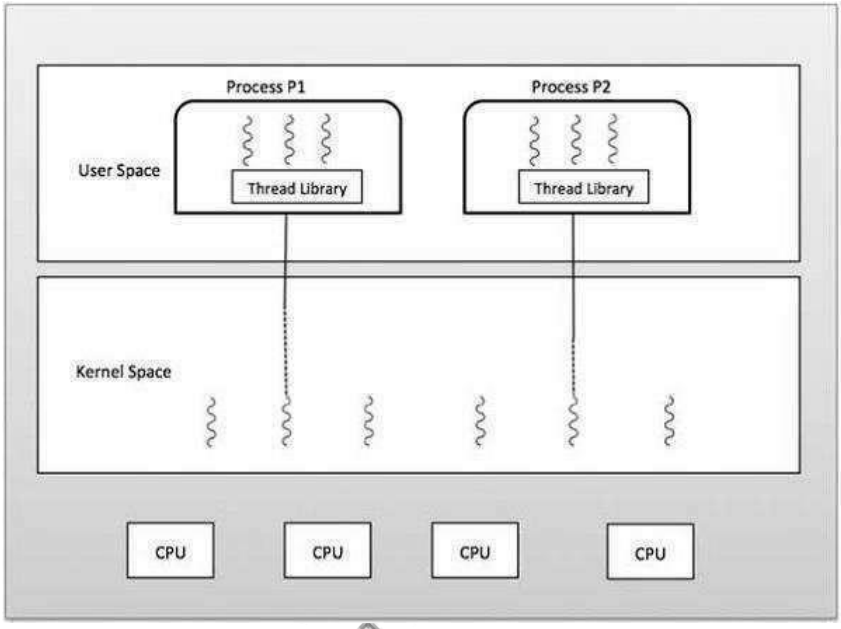


Fig 3.15 Many to One Model

One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.
Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, Windows NT and windows 2000 use one to one relationship model.

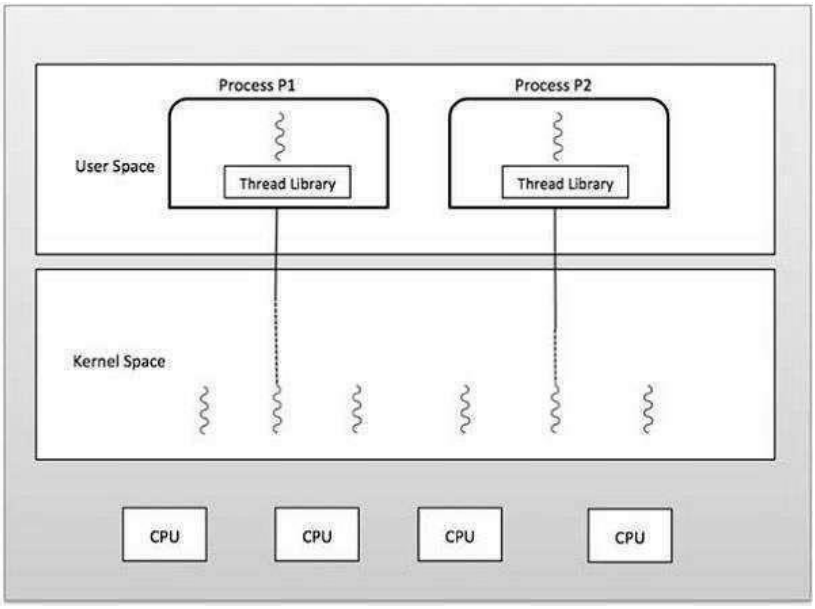


Fig 3.16 One to One Model

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Memory Management

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Process Address Space

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated –

S.N.	Memory Addresses & Description
1	**Symbolic addresses -** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses -** At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	**Physical addresses -** The loader generates these addresses at the time when a program is loaded into main memory.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.
The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Memory Allocation (Partitioning)

Main memory usually has two partitions –

- **Low Memory**– Operating system resides in this memory.
- **High Memory**– User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation -** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation -** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition

	becomes available for another process.

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

S.N.	Fragmentation & Description
1	**External fragmentation -** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation -** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory –

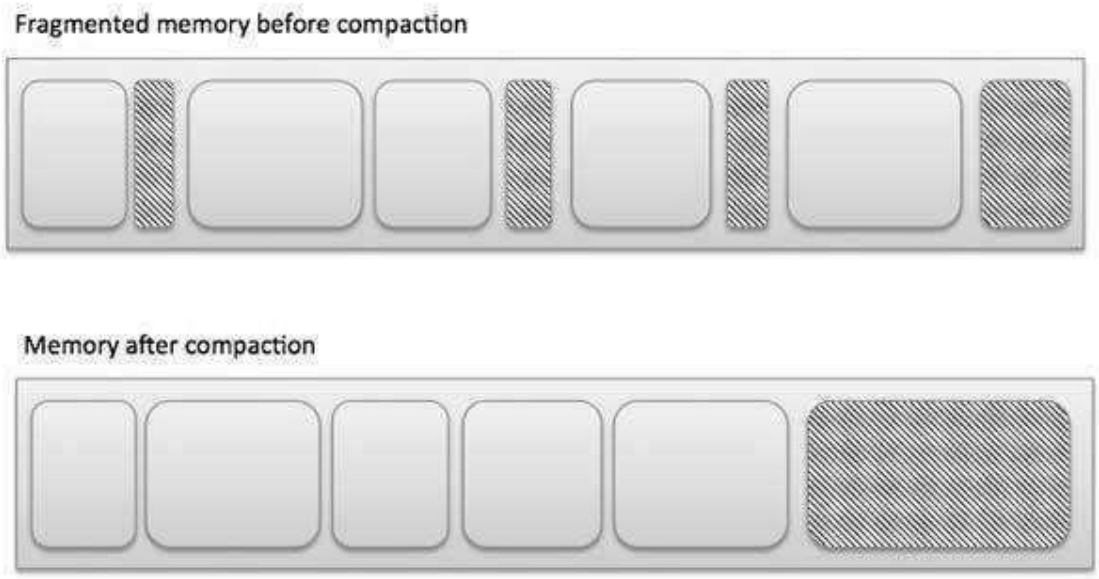


Fig 3.17 Fragmentation

External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Swapping

Swapping is mechanisms in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At

some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason. **Swapping is also known as a technique for memory compaction**.

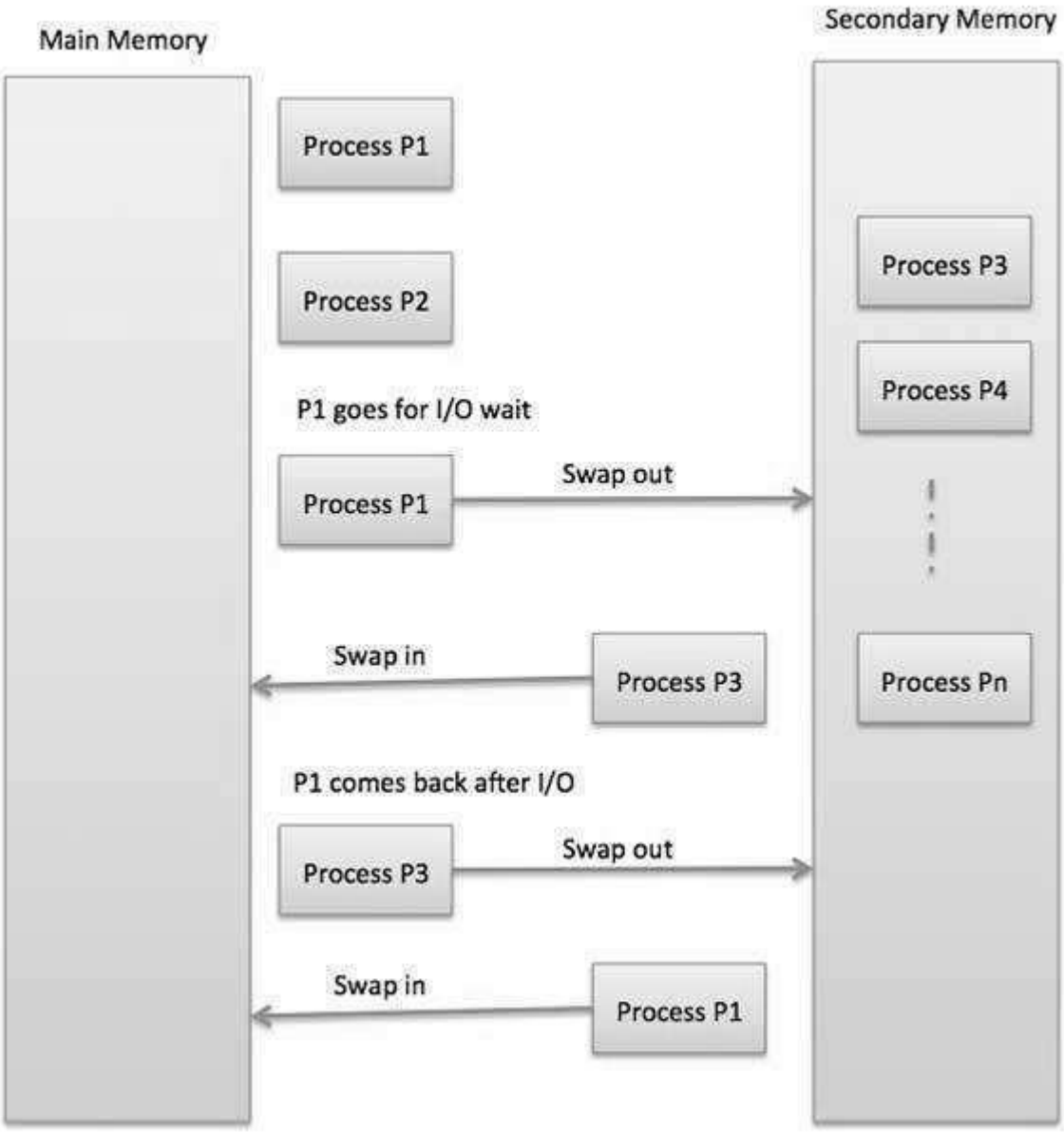


Fig 3.18 Swapping

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

```
2048KB / 1024KB per second
```

= 2 seconds

= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

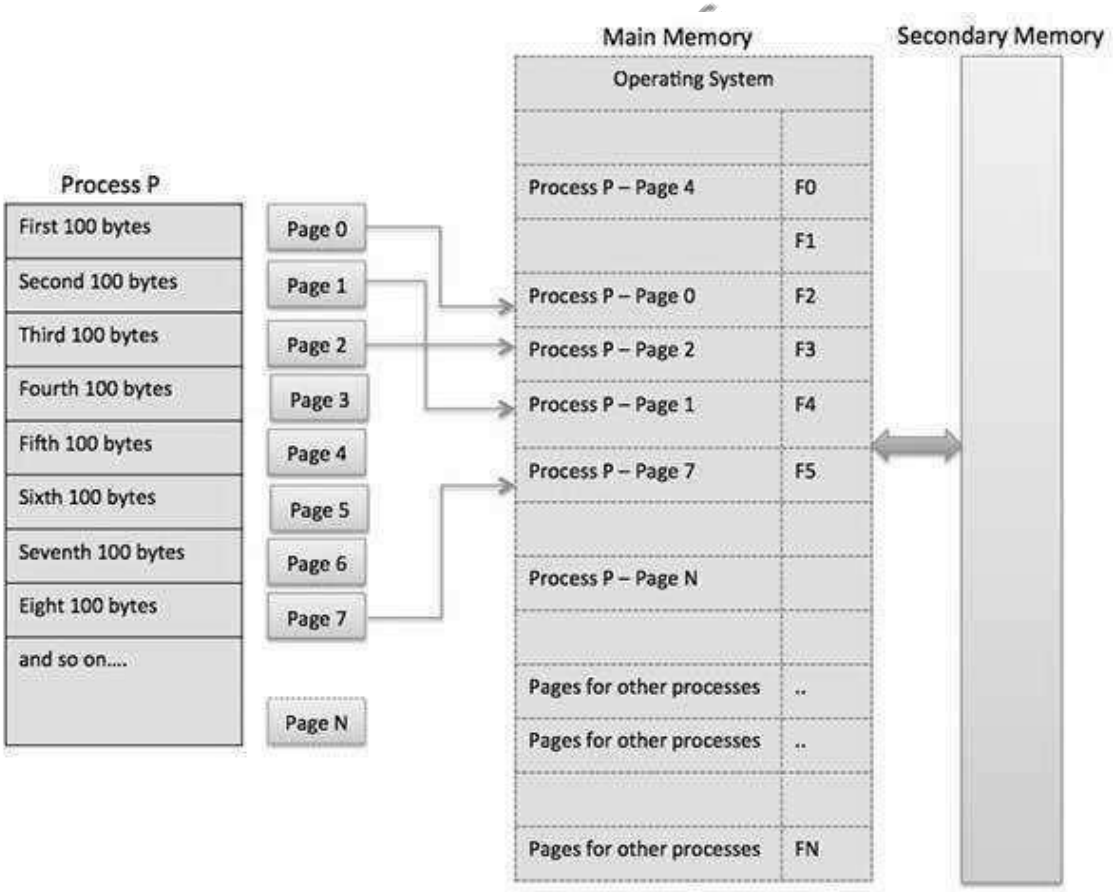


Fig 3.19 Paging

Address Translation

Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a pages of a process to a frame in physical memory.

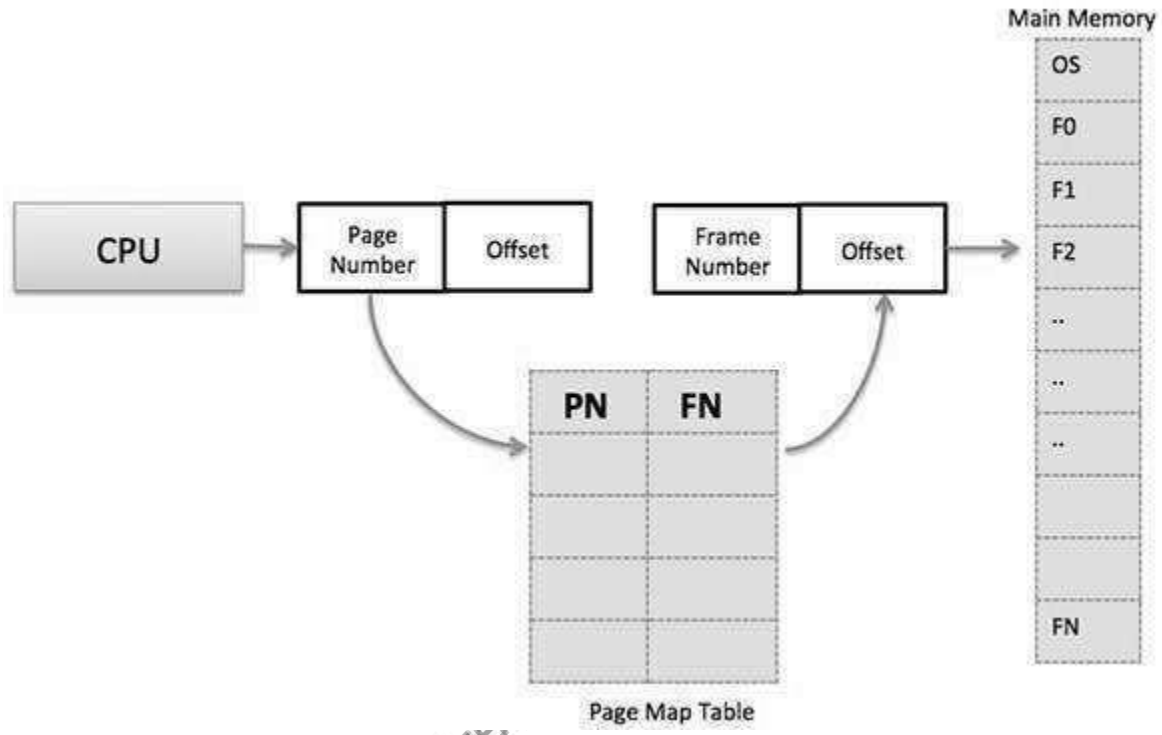


Fig 3.20 Address Translation

When the system allocates a frame to any page, it translates this logical address into a physical address and creates entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation is loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

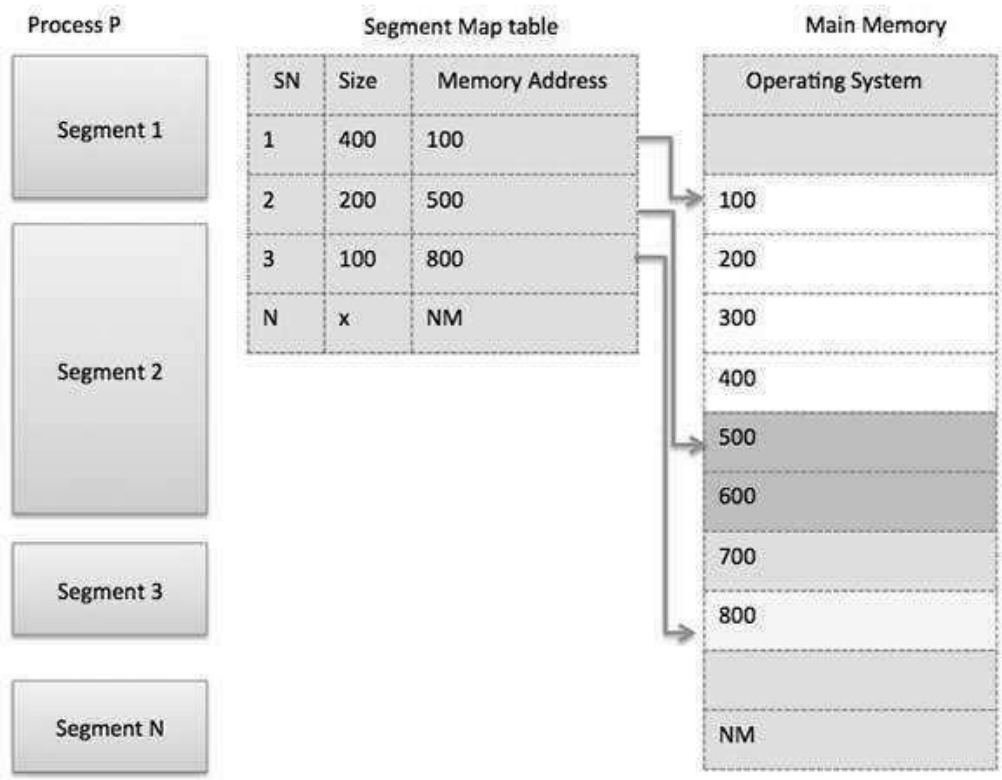


Fig 3.21 Segmentation

Segmented Paging

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

Segment Number → It points to the appropriate Segment Number.

Page Number → It Points to the exact page within the segment

Page Offset → Used as an offset within the page frame

Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.

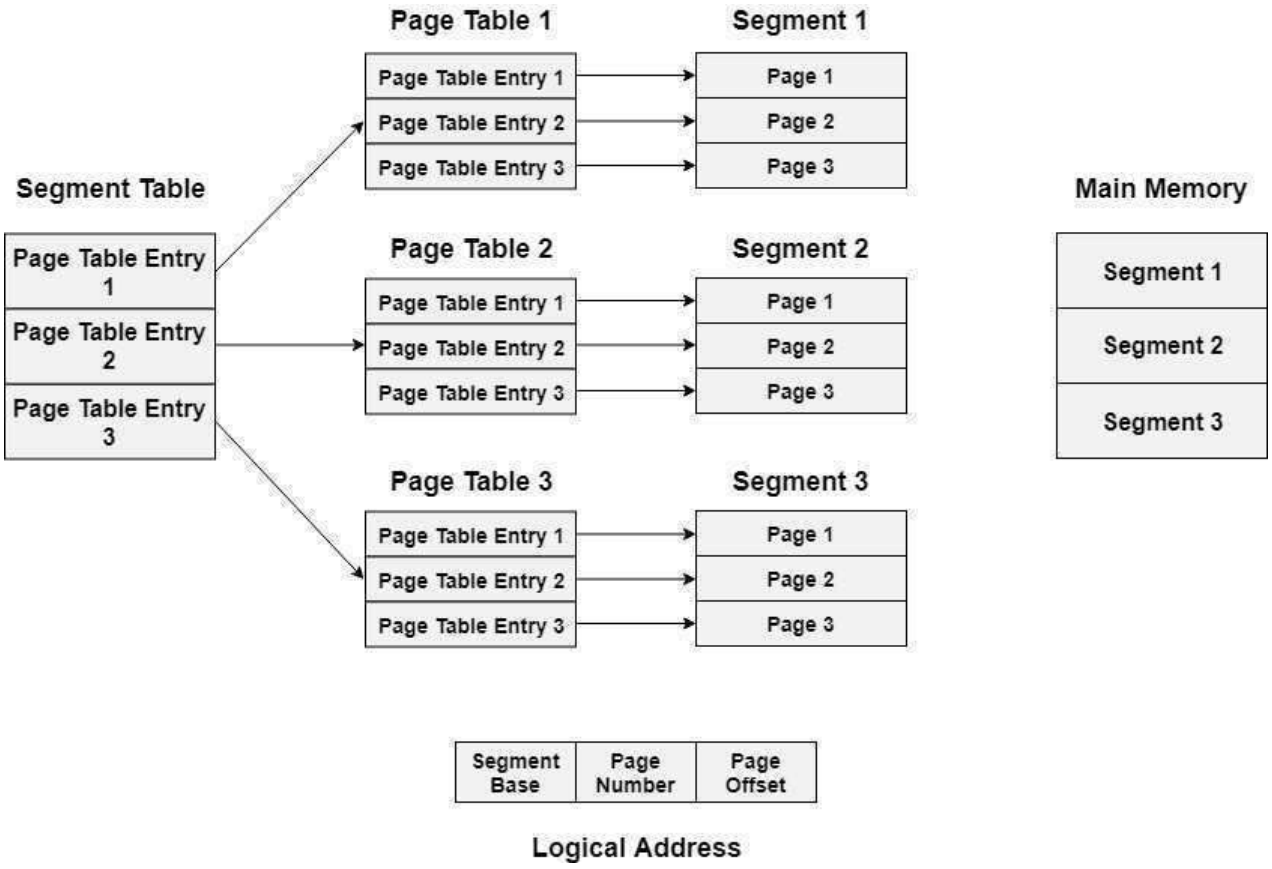


Fig. 3.22 Segmented Paging

Translation of logical address to physical address

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.

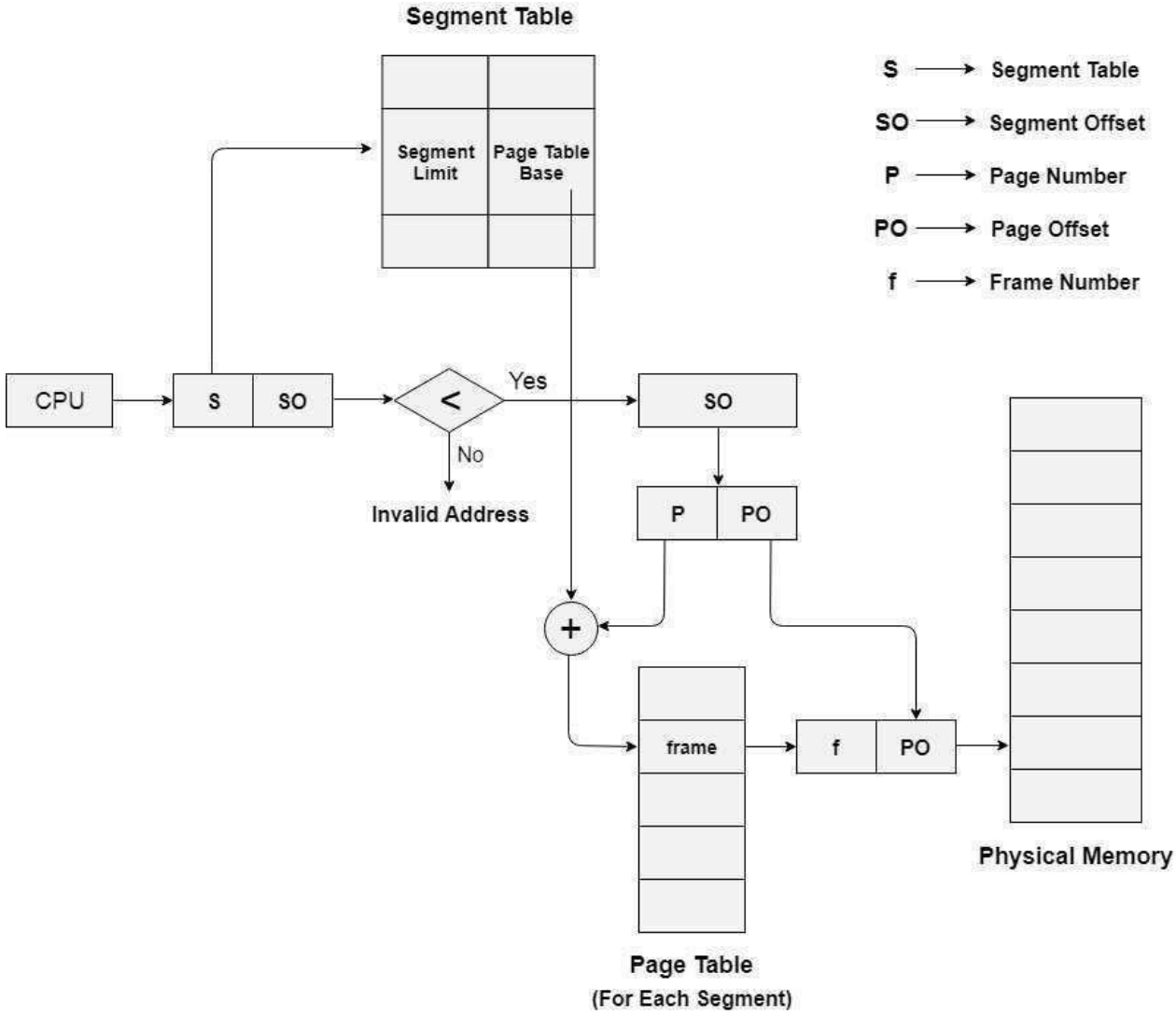


Fig. 3.23 Translation of logical address to physical address

Advantages of Segmented Paging

1. It reduces memory usage.
2. Page table size is limited by the segment size.
3. Segment table has only one entry corresponding to one actual segment.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

Disadvantages of Segmented Paging

1. Internal Fragmentation will be there.
2. The complexity level will be much higher as compare to paging.
3. Page Tables need to be contiguously stored in the memory.

Comparison between Paging and Segmentation technique

Sr. No.	Paging	Segmentation
1	A page is a physical unit of information.	A segment is a logical unit of information.
2	A page is invisible to the user's program.	A segment is visible to the user's program.
3	A page is of fixed size e.g. 4Kbytes.	A segment is of varying size.
4	The page size is determined by the machine architecture.	A segment size is determined by the user.
5	Fragmentation may occur.	Segmentation eliminates fragmentation.
6	Page frames on main memory are required.	No frames are required.

Technique for Execution of large Programs: Overlay:

The main problem in fixed partitioning is the size of a process has to be limited by the maximum size of the partition, which means a process can never be span over another. In order to solve this problem, earlier people have used some solution which is called as Overlays.
The concept of **overlays** is that whenever a process is running it will not use the complete program at the same time, it will use only some part of it. Then overlays concept says that whatever part you required, you load it an once the part is done, then you just unload it, means just pull it back and get the new part you required and run it.

"The process of **transferring a block** of program code or other data into internal memory, replacing what is already stored".
Sometimes it happens that compare to the size of the biggest partition, the size of the program will be even more, then, in that case, you should go with overlays. So overlay is a technique to run a program that is bigger than the size of the physical memory by keeping only those instructions and data that are needed at any given time. Divide the program into modules in such a way that not all modules need to be in the memory at the same time.

Advantage –

- Reduce memory requirement
- Reduce time requirement

Disadvantage –

- Overlap map must be specified by programmer
- Programmer must know memory requirement
- Overlapped module must be completely disjoint
- Programming design of overlays structure is complex and not possible in all cases

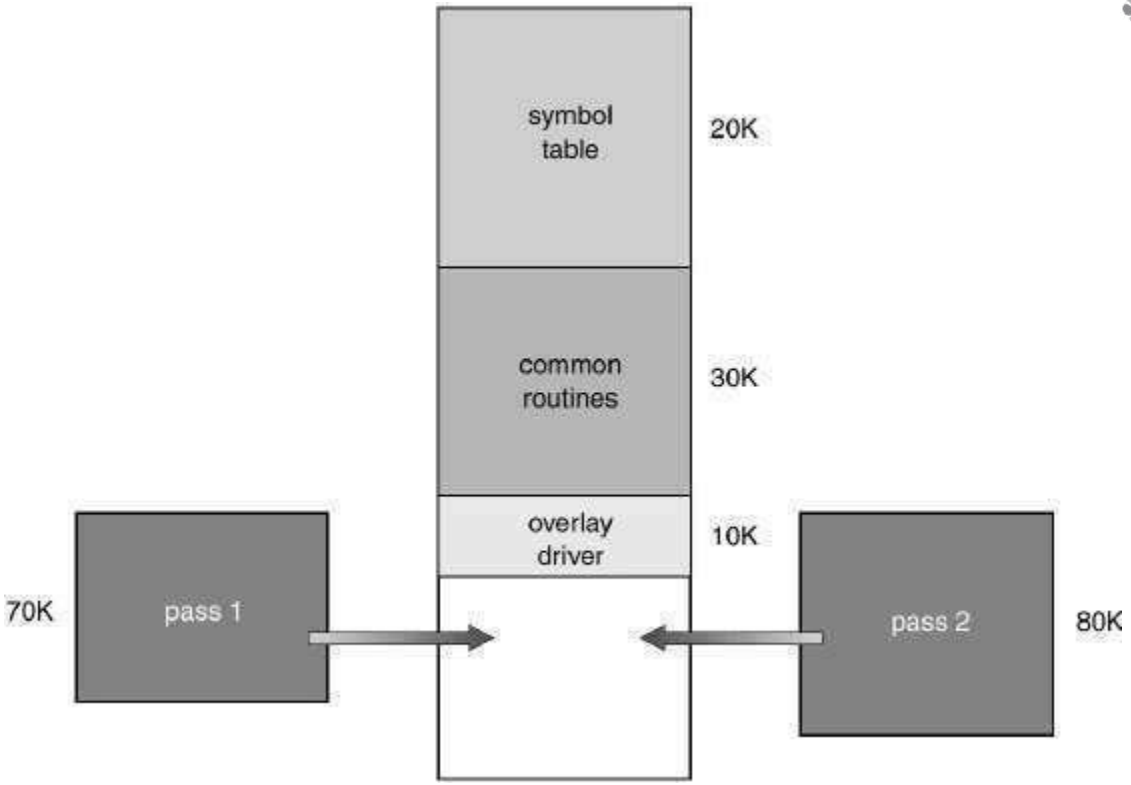


Fig 3.24 Overlays for two pas Assembler

Static vs Dynamic Loading

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

Static vs Dynamic Linking

As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in UNIX are good examples of dynamic libraries.

Virtual Memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory; more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below –

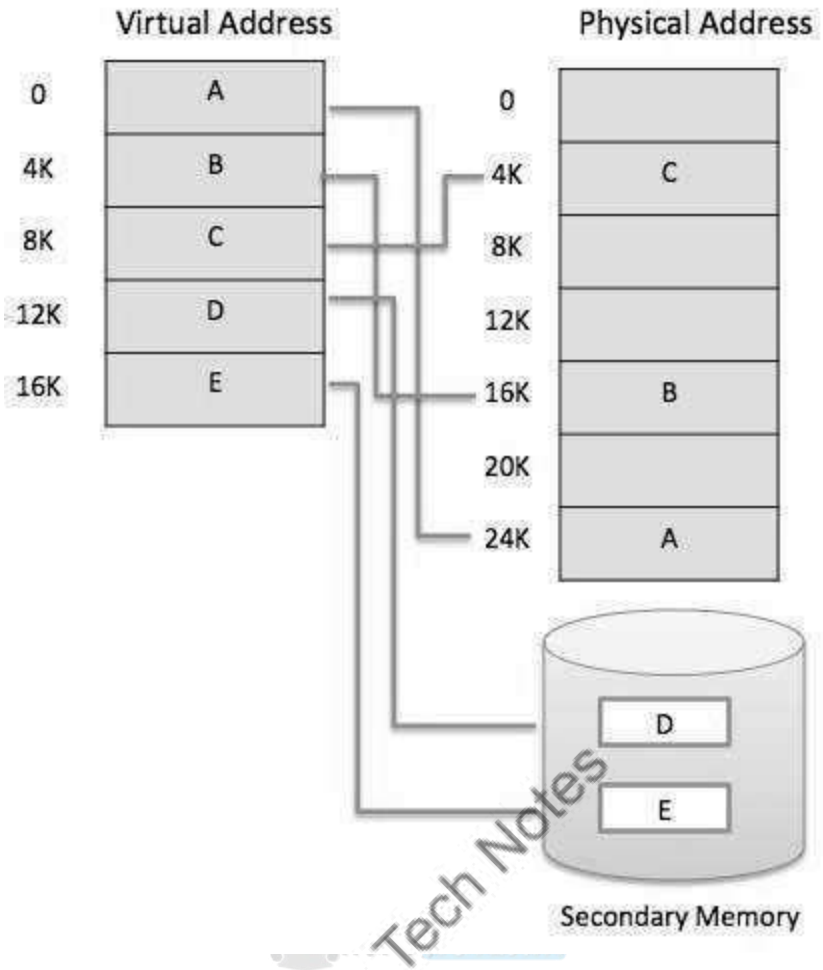


Fig 3.25 Virtual Memory

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

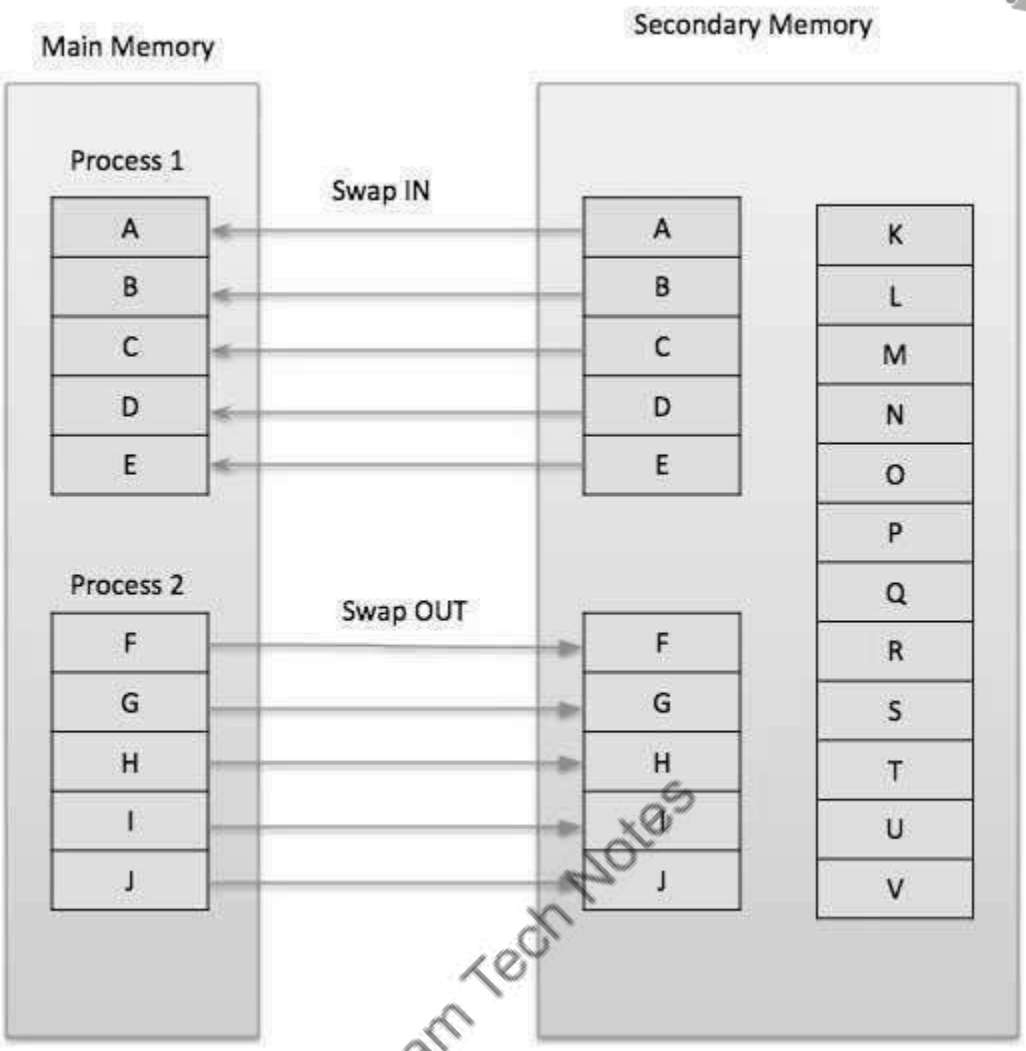


Fig 3.26 Demand Paging

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,
Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses – 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First in First out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

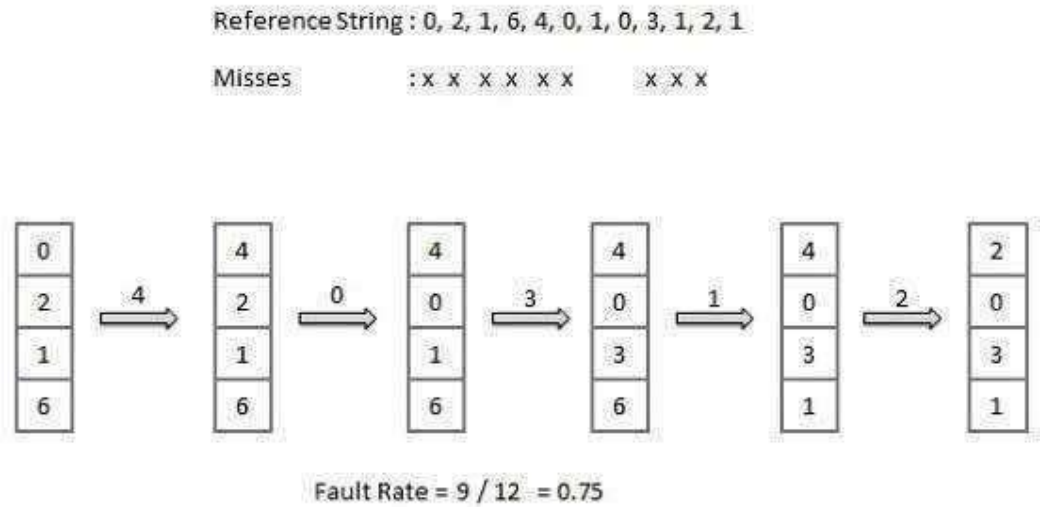


Fig 3.27 First in First out (FIFO) algorithm

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

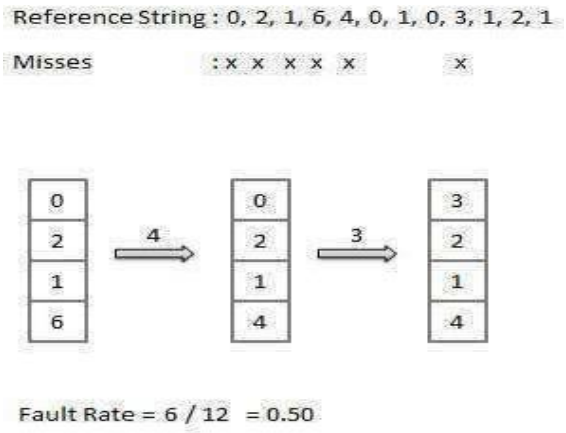


Fig 3.28 Optimal Page algorithm

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

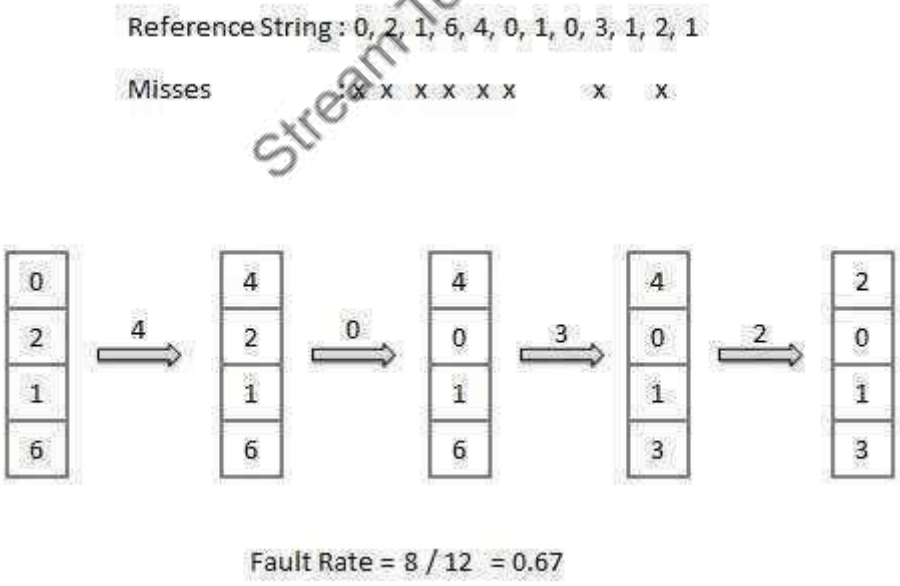


Fig 3.29 Least Recently Used (LRU) algorithm

Page buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used (LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used (MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

UNIT-IV

Input / Output: Principles and Programming

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

- **Block devices**− A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices**− a character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sound cards etc.

Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.

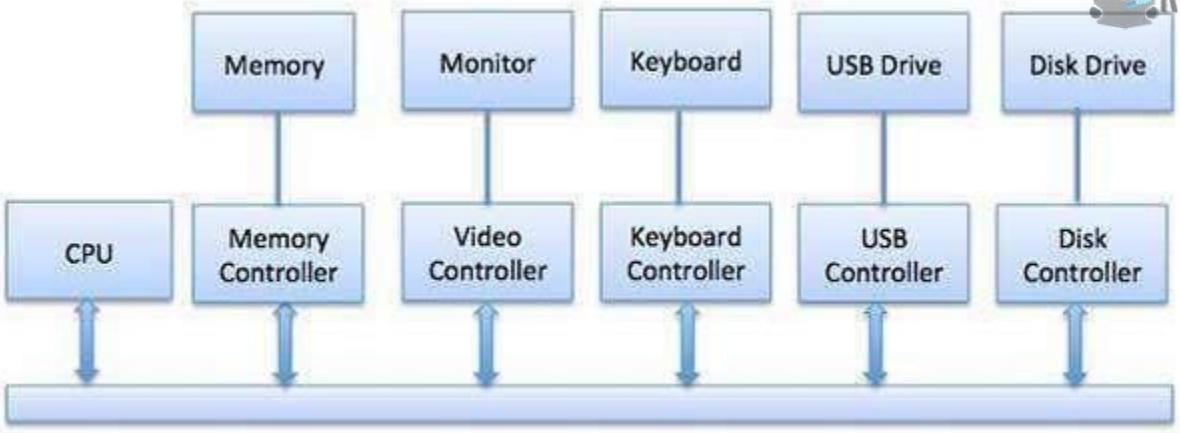


Fig 4.1 Device Controllers

Input /Output Problems

When we analyze device communication, we notice that communication is required at the following three levels:

- The need for a human to input information and receive output from a computer.
- The need for a device to input information and receive output from a computer.
- The need for computers to communicate (receive/send information) over networks.

- The first kind of IO devices operate at rates good for humans to interact. These may be character-oriented devices like a keyboard or an event-generating device like a mouse. Usually, human input using a key board will be a few key depressions at a time. This means that the communication is rarely more than a few bytes. Also, the mouse events can be encoded by a small amount of information (just a few bytes). Even though a human input is very small, it is stipulated that it is very important, and therefore requires an immediate response from the system. A communication which attempts to draw attention often requires the use of an interrupt mechanism or a programmed data mode of operation.
- The second kind of IO requirement arises from devices which have a very high character density such as tapes and disks. With these characteristics, it is not possible to regulate communication with devices on a character by character basis. The information transfer, therefore, is regulated in blocks of information. Additionally, sometimes this may require some kind of format control to structure the information to suit the device and/or data characteristics. For instance, a disk drive differs from a line printer or an image scanner. For each of these devices, the format and structure of information is different. It should be observed that the rate at which a device may provide data and the rates at which an end application may consume it may be considerably different. In spite of these differences, the OS should provide uniform and easy to use IO mechanisms. Usually, this is done by providing a buffer. The OS manages this

buffer so as to be able to comply with the requirements of both the producer and consumer of data.

- The third kind of IO requirements emanate from the need to negotiate system IO with the communications infrastructure. The system should be able to manage communications traffic across the network. This form of IO facilitates access to internet resources to support e-mail, file-transfer amongst machines or Web applications. Additionally now we have a large variety of options available as access devices. These access devices may be in the form of Personal Digital Assistant (PDA), or mobile phones which have infrared or wireless enabled communications. This rapidly evolving technology makes these forms of communications very challenging

Asynchronous Operations

- **Synchronous I/O**– In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O**– I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.

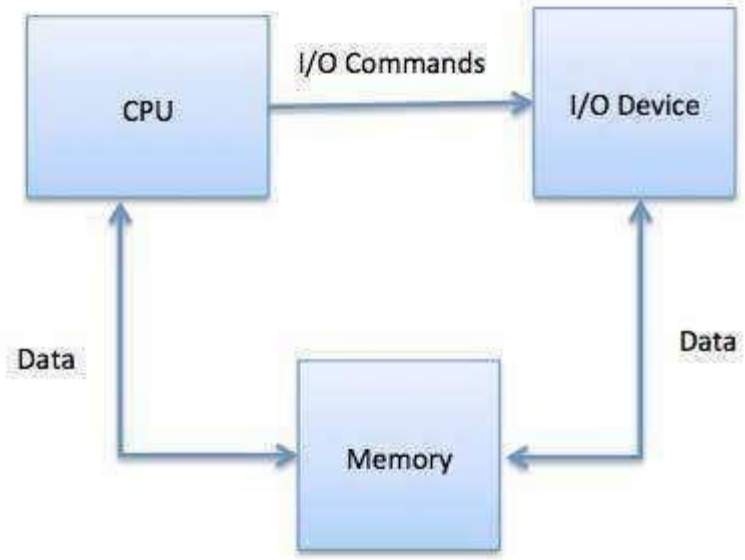


Fig 4.2 Memory-mapped I/O

While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

I/O Interface (Interrupt and DMA Mode)

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access (DMA).

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.
   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

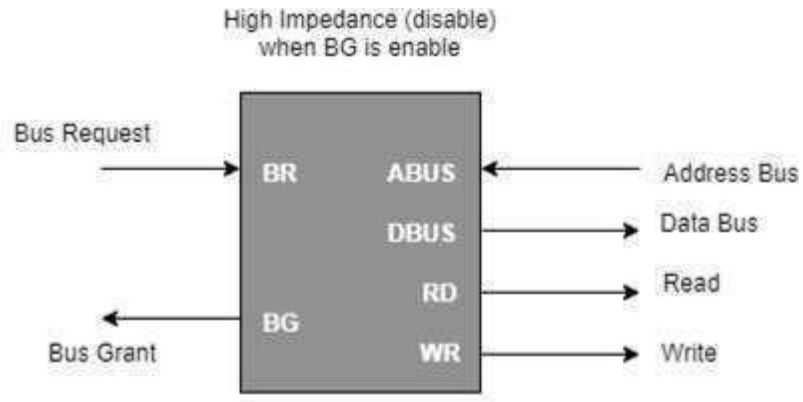


Fig 4.3 CPU Bus signals for DMA transfer

Concurrent I/O

- In AIX® operating systems, you can use concurrent I/O in addition to direct I/O for chunks that use cooked files. Concurrent I/O can improve performance, because it allows multiple reads and writes to a file to occur concurrently, without the usual serialization of noncompeting read and write operations.
- Concurrent I/O can be especially beneficial when you have data in a single chunk file striped across multiple disks.
- Concurrent I/O, which you enable by setting the DIRECT_IO configuration parameter to 2, includes the benefit of avoiding file system buffering and is subject to the same limitations and use of KAIO as occurs if you use direct I/O without concurrent I/O. Thus, when concurrent I/O is enabled, you get both un-buffered I/O and concurrent I/O.

Concurrent Processes

Concurrency is the ability of a database to allow multiple Processes to affect multiple transactions. This is one of the main properties that separate a database from other forms of data storage like spreadsheets.

The ability to offer concurrency is unique to databases. Spreadsheets or other flat file means of storage are often compared to databases, but they differ in this one important regard. Spreadsheets cannot offer several users the ability to view and work on the different data in the same file, because once the first user opens the file it is locked to other users. Other users can read the file, but may not edit data.

Concurrency

Distributed processing involves multiple processes on multiple systems. All of these involve cooperation, competition, and communication between processes that either run simultaneously or are interleaved in arbitrary ways to give the appearance of running simultaneously. Concurrent processing is thus central to operating systems and their design.

Principles and Problems in Concurrency

Concurrency is the interleaving of processes in time to give the appearance of simultaneous execution. Thus it differs from parallelism, which offers genuine simultaneous execution. However the issues and difficulties raised by the two overlap to a large extent:

- Sharing global resources safely is difficult
- Optimal allocation of resources is difficult
- Locating programming errors can be difficult, because the contexts in which errors occur cannot always be reproduced easily.

Parallelism also introduces the issue that different processors may run at different speeds, but again this problem is mirrored in concurrency because different processes progress at different rates.

A Simple Example

The fundamental problem in concurrency is processes interfering with each other while accessing a shared global resource. This can be illustrated with a surprisingly simple example:

chin = getchar();

chout = chin;

putchar(chout);

Imagine two processes P1 and P2 both executing this code at the "same" time, with the following interleaving due to multi-programming.

- P1 enters this code, but is interrupted after reading the character x into chin.
- P2 enters this code, and runs it to completion, reading and displaying the character y.
- P1 is resumed, but chin now contains the character y, so P1 displays the wrong character.

The essence of the problem is the shared global variable chin. P1 sets chin, but this write is subsequently lost during the execution of P2. The general solution is to allow only one process at a time to enter the code that accesses

chin: such code is often called a critical section. When one process is inside a critical section of code, other processes must be prevented from entering that section. This requirement is known as mutual exclusion.

Mutual Exclusion

Mutual exclusion is in many ways the fundamental issue in concurrency. It is the requirement that when a process P is accessing a shared resource R, no other process should be able to access R until P has finished with R. Examples of such resources includes files, I/O devices such as printers, and shared data structures.

There are essentially three approaches to implementing mutual exclusion.

- Leave the responsibility with the processes themselves: this is the basis of most software approaches. These approaches are usually highly error-prone and carry high overheads.
- Allow access to shared resources only through special-purpose machine instructions: i.e. a hardware approach. These approaches are faster but still do not offer a complete solution to the problem, e.g. they cannot guarantee the absence of deadlock and starvation.
- Provide support through the operating system, or through the programming language. We shall outline three approaches in this category: semaphores, monitors, and message passing.

Semaphores

The fundamental idea of semaphores is that processes "communicate" via global counters that are initialized to a positive integer and that can be accessed only through two atomic operations

semSignal(x) increments the value of the semaphore x.

semWait(x) tests the value of the semaphore x: if x > 0, the process decrements x and continues; if x = 0, the process is blocked until some other process performs a semSignal, then it proceeds as above.

A critical code section is then protected by bracketing it between these two operations:

semWait (x);

<critical code section>

semSignal (x);

In general the number of processes that can execute this critical section simultaneously is determined by the initial value given to x. If more than this number tries to enter the critical section, the excess processes will be blocked until some processes exit. Most often, semaphores are initialized to one.

Monitors

The principal problem with semaphores is that calls to semaphore operations tend to be distributed across a program, and therefore these sorts of programs can be difficult to get correct, and very difficult indeed to prove correct, Monitors address this problem by imposing a higher-level structure on accesses to semaphore variables. A monitor is essentially an object (in the Java sense) which has the semaphore variables as internal (private) data and the semaphore operations as (public) operations. Mutual exclusion is provided by allowing only one process to execute the monitor's code at any given time. Monitors are significantly easier to validate than "bare" semaphores for at least two reasons:

- All synchronization code is confined to the monitor
- Once the monitor is correct, any number of processes sharing the resource will operate correctly.

Inter Process Communication

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical,

there are many situations when co-operative nature can be utilized for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure below shows a basic structure of communication between processes via shared memory method and via message passing.

An operating system can implement both method of communication. First, we will discuss the shared memory method of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process. Let's discuss an example of communication between processes using shared memory method.

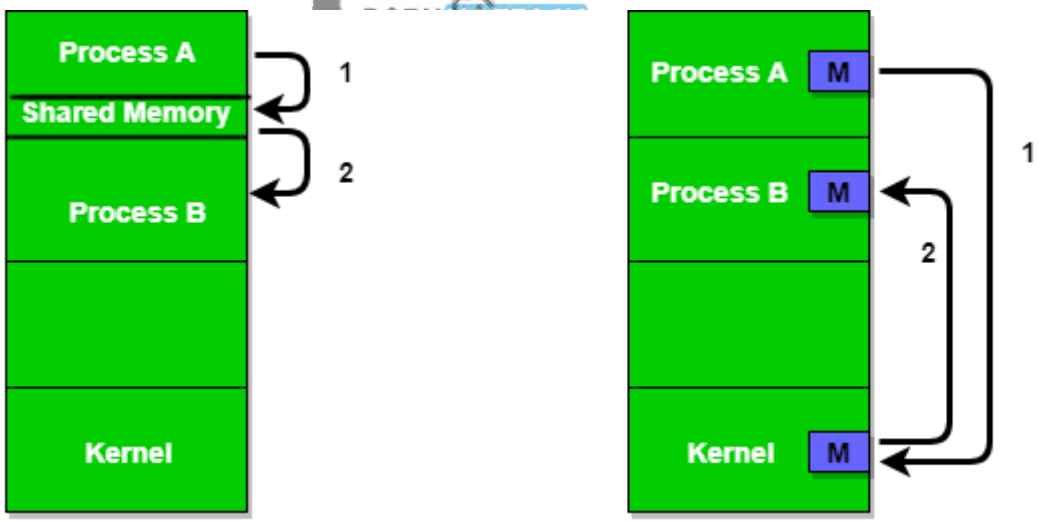


Fig 4.4 Shared Memory

Process Synchronization

Process Synchronization means sharing system resources by processes in such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.

Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

3. Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

Semaphores

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called **semaphore**. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, **Wait** and **Signal** by P(S) and V(S) respectively.

In very simple words, **semaphore** is a variable which can hold only a non-negative Integer value, shared between all the threads, with operations **wait** and **signal**, which work as follow:

```
P(S): if S ≥ 1 then S := S - 1
    else <block and enqueue the process>;

V(S): if <some process is blocked on the queue>
     then <unblock a process>
    else S := S + 1;
```


The classical definitions of **wait** and **signal** are:

- **Wait**: Decrements the value of its argument S, as soon as it would become non-negative (greater than or equal to 1).
- **Signal**: Increments the value of its argument S, as there is no more process blocked on the queue.

Properties of Semaphores

1. It's simple and always has a non-negative Integer value.
2. Works with many processes.
3. Can have many different critical sections with different semaphores.
4. Each critical section has unique access semaphores.
5. Can permit multiple processes into the critical section at once, if desirable.

Types of Semaphores

Semaphores are mainly of two types:

1. **Binary Semaphore:**

   It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a **Mutex**. A binary semaphore is initialized to 1 and only takes the values 0 and 1 during execution of a program.

2. **Counting Semaphores:**

   These are used to implement bounded concurrency.

Example of Use

Here is a simple step wise implementation involving declaration and usage of semaphore.

```
Shared var mutex: semaphore = 1;
Process i
   begin
   .
   .
   P(mutex);
   execute CS;
   V(mutex);
   .
   .
   End;
```

Limitations of Semaphores

1. **Priority Inversion** is a big limitation of semaphores.
2. Their use is not enforced, but is by convention only.
3. With improper use, a process may block indefinitely. Such a situation is called **Deadlock**. We will be studying deadlocks in details in coming lessons.

Deadlock

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Deadlock can arise if four conditions hold simultaneously:

1. **Mutual Exclusion**: A resource can only be shared in mutually exclusive manner. It implies, if two processes cannot use the same resource at the same time.
2. **Hold and Wait:** A process waits for some resources while holding another resource at the same time.
3. **No preemption**: The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.
4. **Circular Wait**: All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

Deadlocks Prevention

Deadlocks can be prevented by prevent at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion**
   Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

2. **Hold and Wait**
   In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. **No Preemption**
   Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. **Circular Wait**
   Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing (or decreasing) order.

Handling Deadlock

The above points focus on preventing deadlocks. But what to do once a deadlock has occured. Following three strategies can be used to remove deadlock after its occurrence.

1. **Preemption**
   We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

2. **Rollback**
   In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last

checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3. **Kill one or more processes**
   This is the simplest way, but it works.

Deadlock Avoidance

- The general idea behind deadlock avoidance is to prevent deadlocks from ever happening, by preventing at least one of the aforementioned conditions.
- This requires more information about each process, AND tends to lead to low device utilization. (it is a conservative approach.)
- In some algorithms the scheduler only needs to know the maximum number of each resource that a process might potentially use. In more complex algorithms the scheduler can also take advantage of the schedule of exactly what resources may be needed in what order.
- When a scheduler sees that starting a process or granting resource requests may lead to future deadlocks, then that process is just not started or the request is not granted.
- A resource allocation **state** is defined by the number of available and allocated resources and the maximum requirements of all processes in the system.

Safe State

- A state is **safe** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock state.
- More formally, a state is safe if there exists a **safe sequence** of processes {P0, P1, P2, ..., PN} such that all of the resource requests for Pi can be granted using the resources currently allocated to Pi and all processes Pj where j < i. (I.e. if all the processes prior to Pi finish and free up their resources, then Pi will be able to finish also, using the resources that they have freed up.)
- If a safe sequence does not exist, then the system is in an unsafe state, which **MAY** lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks.)

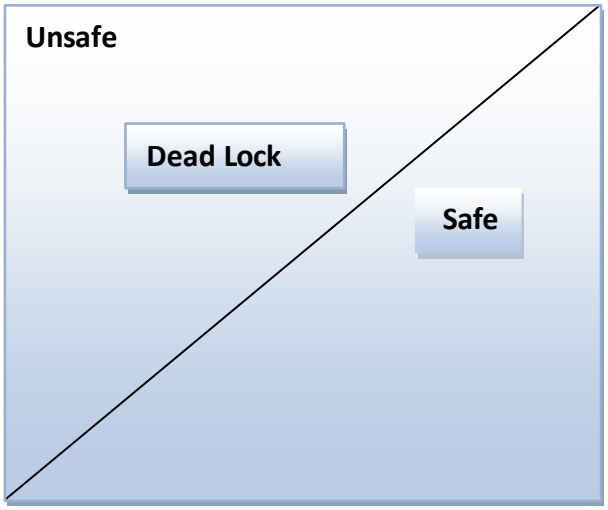


Fig: 4.5 Safe, unsafe, and deadlocked state spaces.

- For example, consider a system with 12 tape drives, allocated as follows. Is this a safe state? What is the safe sequence?

	Maximum Needs	Current Allocation
P0	10	5
P1	4	2
P2	9	2

- What happens to the above table if process P2 requests and is granted one more tape drive?
- Key to the safe state approach is that when a request is made for resources, the request is granted only if the resulting allocation state is a safe one.

Resource-Allocation Graph Algorithm

- If resource categories have only single instances of their resources, then deadlock states can be detected by cycles in the resource-allocation graphs.
- In this case, unsafe states can be recognized and avoided by augmenting the resource-allocation graph with **claim edges**, noted by dashed lines, which point from a process to a resource that it may request in the future.
- In order for this technique to work, all claim edges must be added to the graph for any particular process before that process is allowed to request any resources. (Alternatively, processes may only make requests for resources for which they have already established claim edges, and claim edges cannot be added to any process that is currently holding resources.)

- When a process makes a request, the claim edge Pi->Rj is converted to a request edge. Similarly when a resource is released, the assignment reverts back to a claim edge.
- This approach works by denying requests that would produce cycles in the resource-allocation graph, taking claim edges into effect.
- Consider for example what happens when process P2 requests resource R2:

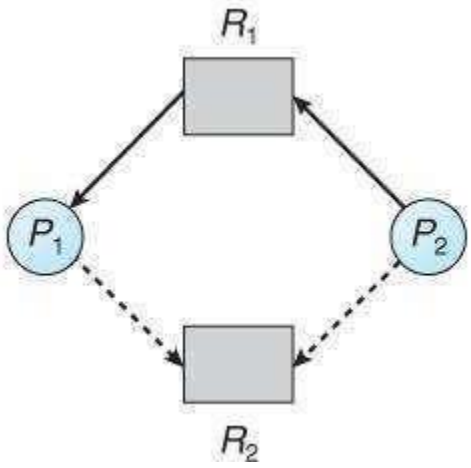


Fig 4.6 Resource allocation graph for deadlock avoidance

- The resulting resource-allocation graph would have a cycle in it, and so the request cannot be granted.

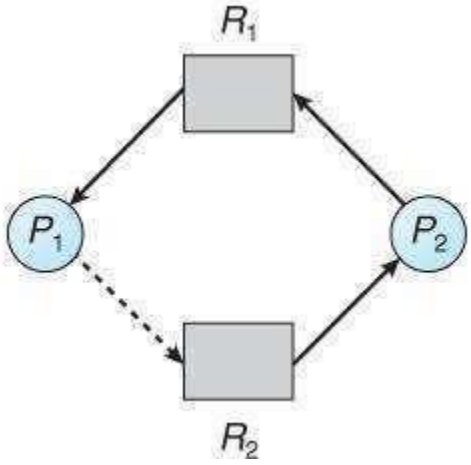


Fig. 4.6 An unsafe state in a resource allocation graph

Banker's Algorithm

- For resource categories that contain more than one instance the resource-allocation graph method does not work, and more complex (and less efficient) methods must be chosen.

- The Banker's Algorithm gets its name because it is a method that bankers could use to assure that when they lend out resources they will still be able to satisfy all their clients. (A banker won't loan out a little money to start building a house unless they are assured that they will later be able to loan out the rest of the money to finish the house.)
- When a process starts up, it must state in advance the maximum allocation of resources it may request, up to the amount available on the system.
- When a request is made, the scheduler determines whether granting the request would leave the system in a safe state. If not, then the process must wait until the request can be granted safely.
- The banker's algorithm relies on several key data structures: (where n is the number of processes and m is the number of resource categories.)
    - Available[m] indicates how many resources are currently available of each type.
    - Max[n][m] indicates the maximum demand of each process of each resource.
    - Allocation[n][m] indicates the number of each resource category allocated to each process.
    - Need[n][m] indicates the remaining resources needed of each type for each process. (Note that Need[i][j] = Max[i][j] - Allocation[i][j] for all i, j.)
- For simplification of discussions, we make the following notations / observations:
    - One row of the Need vector, Need[i], can be treated as a vector corresponding to the needs of process i, and similarly for Allocation and Max.
    - A vector X is considered to be <= a vector Y if X[i] <= Y[i] for all i.

Safety Algorithm

- In order to apply the Banker's algorithm, we first need an algorithm for determining whether or not a particular state is safe.
- This algorithm determines if the current state of a system is safe, according to the following steps:
    1. Let Work and Finish be vectors of length m and n respectively.
        - Work is a working copy of the available resources, which will be modified during the analysis.
        - Finish is a vector of Booleans indicating whether a particular process can finish. (or has finished so far in the analysis.)
        - Initialize Work to Available, and Finish to false for all elements.

2. Find an i such that both (A) Finish[i] == false, and (B) Need[i] < Work. This process has not finished, but could with the given available working set. If no such i exists, go to step 4.
3. Set Work = Work + Allocation[i], and set Finish[i] to true. This corresponds to process i finishing up and releasing its resources back into the work pool. Then loop back to step 2.
4. If finish[i] == true for all i, then the state is a safe state, because a safe sequence has been found.

- (JTB's Modification:
    1. In step 1. instead of making Finish an array of booleans initialized to false, make it an array of ints initialized to 0. Also initialize an int s = 0 as a step counter.
    2. In step 2, look for Finish[i] == 0.
    3. In step 3, set Finish[i] to ++s. S is counting the number of finished processes.
    4. For step 4, the test can be either Finish[i] > 0 for all i, or s >= n. The benefit of this method is that if a safe state exists, then Finish[ ] indicates one safe sequence (of possibly many.))

Resource-Request Algorithm (The Bankers Algorithm)

- Now that we have a tool for determining if a particular state is safe or not, we are now ready to look at the Banker's algorithm itself.
- This algorithm determines if a new request is safe, and grants it only if it is safe to do so.
- When a request is made (that does not exceed currently available resources), pretend it has been granted, and then see if the resulting state is a safe one. If so, grant the request, and if not, deny the request, as follows:
    1. Let Request[n][m] indicate the number of resources of each type currently requested by processes. If Request[i] > Need[i] for any process i, raise an error condition.
    2. If Request[i] > Available for any process i, then that process must wait for resources to become available. Otherwise the process can continue to step 3.
    3. Check to see if the request can be granted safely, by pretending it has been granted and then seeing if the resulting state is safe. If so, grant the request, and if not, then the process must wait until its request can be granted safely.The procedure for granting a request (or pretending to for testing purposes) is:
        - Available = Available - Request

- Allocation = Allocation + Request
- Need = Need - Request

An Illustrative Example

- Consider the following situation:

	Allocation	Max	Available	Need
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

- And now consider what happens if process P1 requests 1 instance of A and 2 instances of C. (Request[1] = (1, 0, 2))

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- What about requests of (3, 3,0) by P4? or (0, 2, 0) by P0? Can these be safely granted? Why or why not?

Recovery from Deadlock

There are three basic approaches to recovery from deadlock:

1. Inform the system operator, and allow him/her to take manual intervention.
2. Terminate one or more processes involved in the deadlock
3. Preempt resources.

Process Termination

- Two basic approaches, both of which recover resources allocated to terminated processes:
  - Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.
  - Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.
- In the latter case there are many factors that can go into deciding which processes to terminate next:
  1. Process priorities.
  2. How long the process has been running, and how close it is to finishing.
  3. How many and what type of resources is the process holding. (Are they easy to preempt and restore?)
  4. How many more resources does the process need to complete.
  5. How many processes will need to be terminated
  6. Whether the process is interactive or batch.
  7. (Whether or not the process has made non-restorable changes to any resource.)

Resource Preemption

When preempting resources to relieve deadlock, there are three important issues to be addressed:

1. **Selecting a victim** - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
2. **Rollback** - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (I.e. abort the process and make it start over.)
3. **Starvation** - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

Livelock:

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.

A human example of livelock would be two people who meet face-to-face in a corridor and each move aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

UNIT-V

Distributed Operating System

These types of operating system is a recent advancement in the world of computer technology and are being widely accepted all-over the world and, that too, with a great pace. Various autonomous interconnected computers communicate each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred as **loosely coupled systems** or distributed systems. These systems processors differ in sizes and functions. The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.

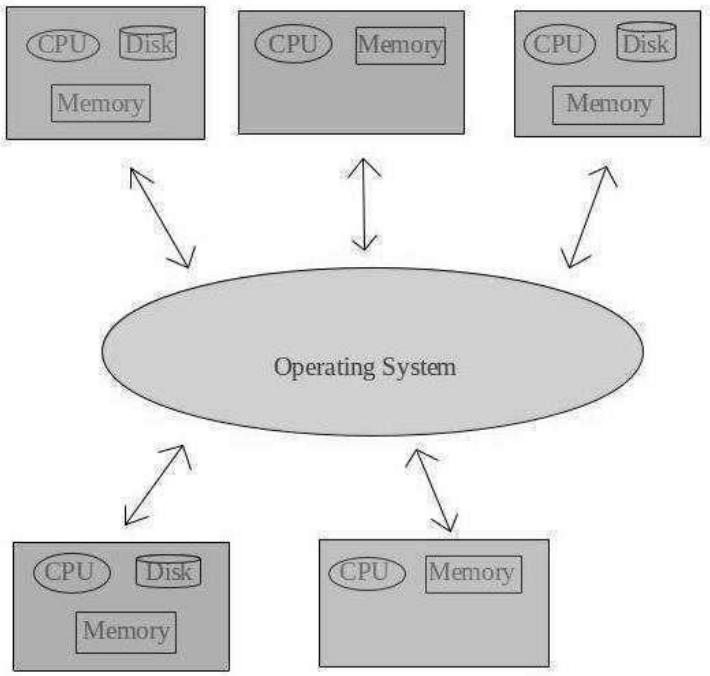


Fig. 5.1 Distributed Operating System

Advantages of Distributed Operating System:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

Disadvantages of Distributed Operating System:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which are used are not well defined yet
- These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet

Examples of Distributed Operating System are- LOCUS etc.

Network Operating System

These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions. These type of operating systems allows shared access of files, printers, security, applications, and other networking functions over a small private network. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc. and that's why these computers are popularly known as **tightly coupled systems**.

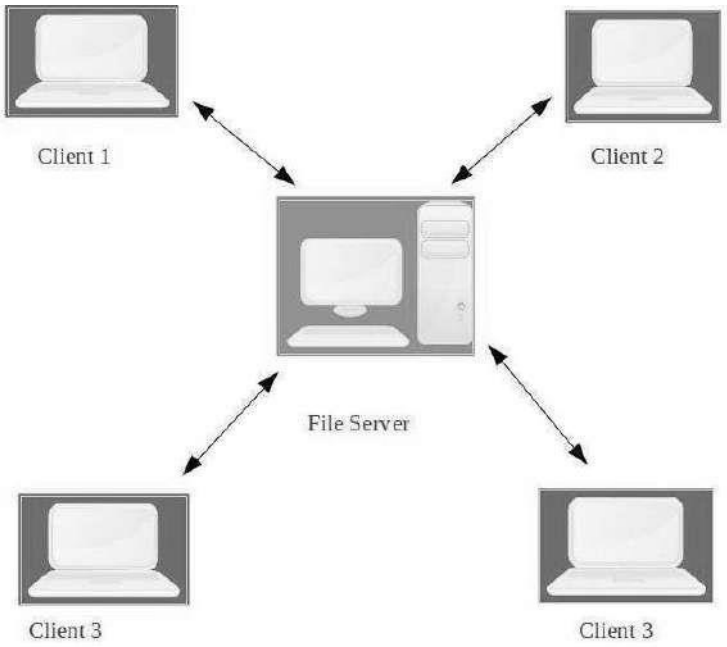


Fig. 5.2 Network Operating System

Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated to the system
- Server access are possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on central location for most operations
- Maintenance and updates are required regularly

Examples of Network Operating System are: Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD etc.

Differences between Network Operating System and Distributed Operating System

Sr. No.	Network Operating System	Distributed Operating System
1	A network operating system is made up of software and associated protocols that allow a set of computer network to be used together.	A distributed operating system is an ordinary centralized operating system but runs on multiple independent CPUs.
2	Environment users are aware of multiplicity of machines.	Environment users are not aware of multiplicity of machines.
3	Control over file placement is done manually by the user.	It can be done automatically by the system itself.
4	Performance is badly affected if certain part of the hardware starts malfunctioning.	It is more reliable or fault tolerant i.e distributed operating system performs even if certain part of the hardware starts malfunctioning.
5	Remote resources are accessed by either logging into the desired remote machine or transferring data from the remote machine to user's own machines.	Users access remote resources in the same manner as they access local resources.

Multiprocessor Operating System

It refers to the use of two or more central processing units (CPU) within a single computer system. These multiple CPUs are in a close communication sharing the computer bus, memory and other peripheral devices. These systems are referred as *tightly* coupled systems.

These types of systems are used when very high speed is required to process a large volume of data. These systems are generally used in environment like satellite control, weather forecasting etc. The basic organization of multiprocessing system is shown in fig.

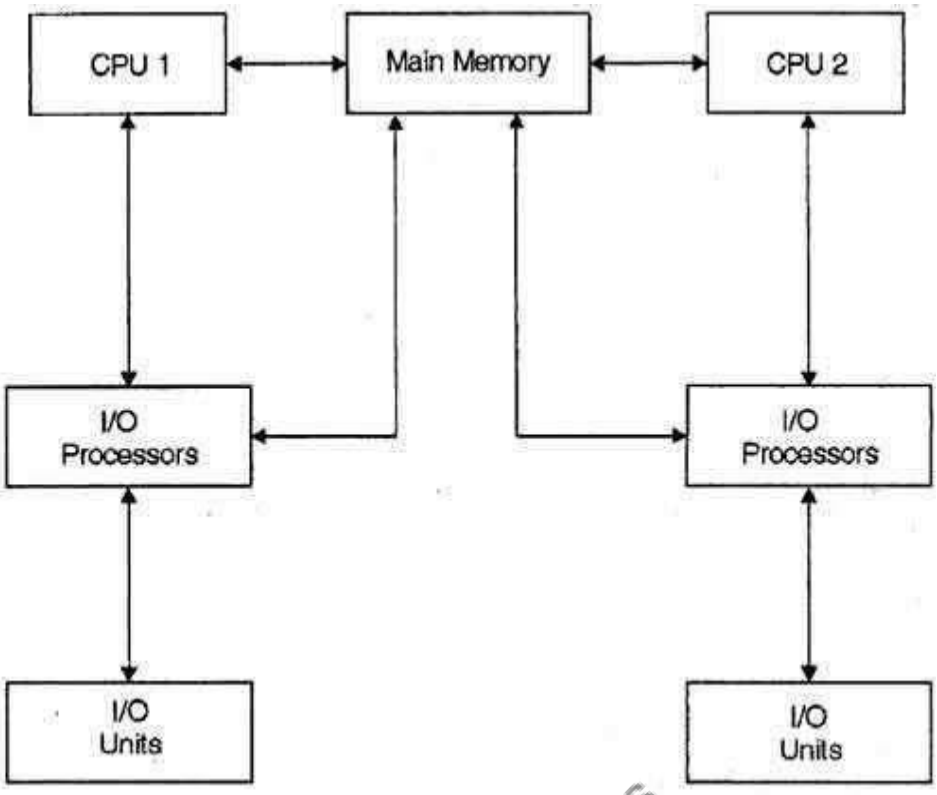


Fig. 5.3 Multiprocessor Operating System

Multiprocessing system is based on the symmetric multiprocessing model, in which each processor runs an identical copy of operating system and these copies communicate with each other. In this system processor is assigned a specific task. A master processor controls the system. This scheme defines a master-slave relationship. These systems can save money in compare to single processor systems because the processors can share peripherals, power supplies and other devices. The main advantage of multiprocessor system is to get more work done in a shorter period of time. Moreover, multiprocessor systems prove more reliable in the situations of failure of one processor. In this situation, the system with multiprocessor will not halt the system; it will only slow it down.

In order to employ multiprocessing operating system effectively, the computer system must have the followings:

1. Motherboard Support:

A motherboard capable of handling multiple processors. This means additional sockets or slots for the extra chips and a chipset capable of handling the multiprocessing arrangement.

2. Processor Support:

Processors those are capable of being used in a multiprocessing system.

The whole task of multiprocessing is managed by the operating system, which allocates different tasks to be performed by the various processors in the system.

Applications designed for the use in multiprocessing are said to be threaded, which means that they are broken into smaller routines that can be run independently. This allows the operating

system to let these threads run on more than one processor simultaneously, which is multiprocessing that results in improved performance.

Multiprocessor system supports the processes to run in parallel. Parallel processing is the ability of the CPU to simultaneously process incoming jobs. This becomes most important in computer system, as the CPU divides and conquers the jobs. Generally the parallel processing is used in the fields like artificial intelligence and expert system, image processing, weather forecasting etc.

Locking system: In order to provide safe access to the resources shared among multiple processors, they need to be protected by locking scheme. The purpose of a locking is to serialize accesses to the protected resource by multiple processors. Undisciplined use of locking can severely degrade the performance of system. This form of contention can be reduced by using locking scheme, avoiding long critical sections, replacing locks with lock-free algorithms, or, whenever possible, avoiding sharing altogether.

Shared data: The continuous accesses to the shared data items by multiple processors (with one or more of them with data write) are serialized by the cache coherence protocol. Even in a moderate-scale system, serialization delays can have significant impact on the system performance. In addition, bursts of cache coherence traffic saturate the memory bus or the interconnection network, which also slows down the entire system. This form of contention can be eliminated by either avoiding sharing or, when this is not possible, by using replication techniques to reduce the rate of write accesses to the shared data.

False sharing: This form of contention arises when unrelated data items used by different processors are located next to each other in the memory and, therefore, share a single cache line: The effect of false sharing is the same as that of regular sharing bouncing of the cache line among several processors. Fortunately, once it is identified, false sharing can be easily eliminated by setting the memory layout of non-shared data.

Case Study: Linux:

Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries

implement most of the functionalities of the operating system and do not requires kernel module's code access rights.

- **System Utility** – System Utility programs are responsible to do specialized, individual level tasks.

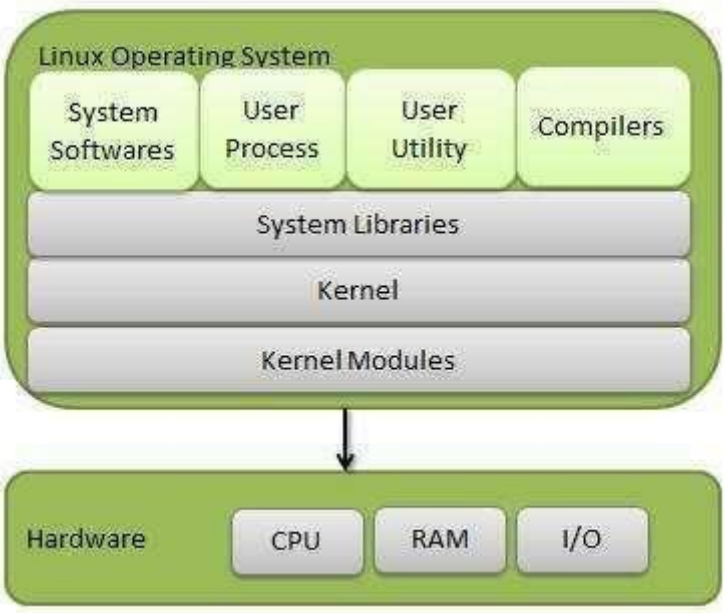


Fig. 5.4 Linux Operating System

Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each process and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

- **Portable** – Portability means software can works on different types of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

- **Multiprogramming** – Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture

The following illustration shows the architecture of a Linux system –

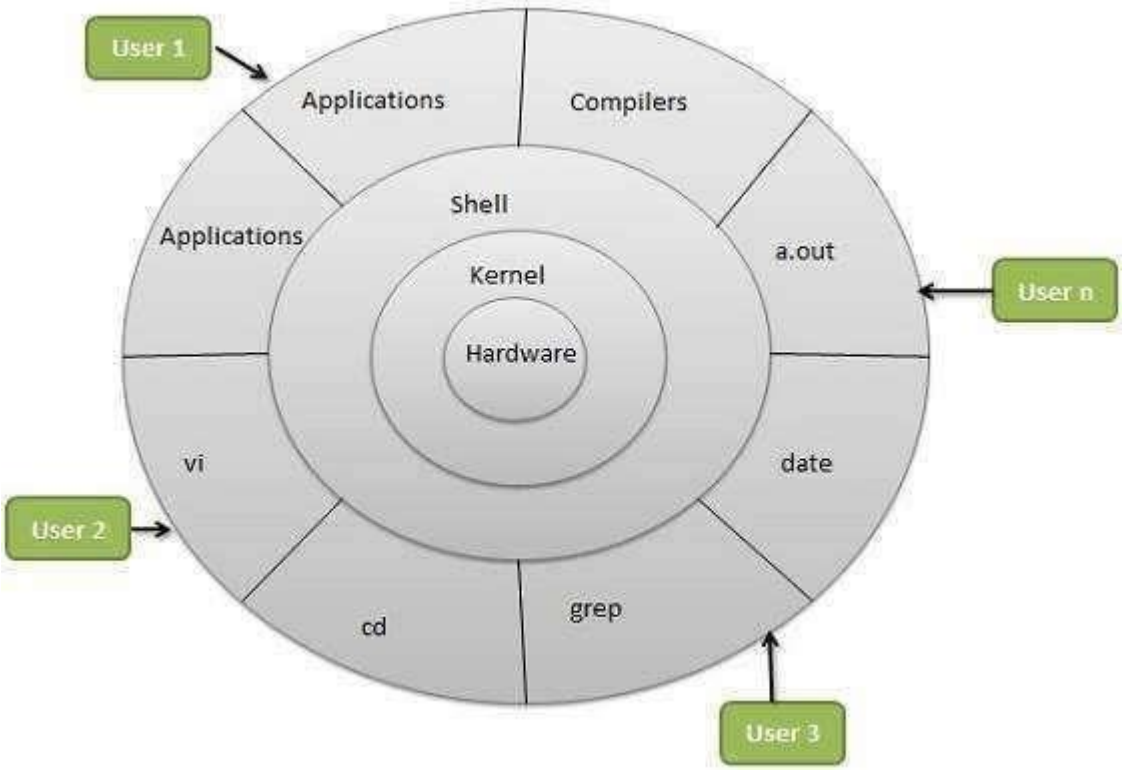


Fig. 5.5 Linux Operating System Architecture

The architecture of a Linux System consists of the following layers –

- **Hardware layer** – Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** – It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** – An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.
- **Utilities** – Utility programs that provide the user most of the functionalities of an operating systems.

The Shell

- The shell is the interface between the command language user and the OS
- The shell is a user interface and comes in many forms (Bourne Shell, sh; Berkeley C Shell, csh; Korn Shell, ksh; Restricted Shell, rsh)
- User allowed to enter input when prompted ($ or %)
- System supports all shells running concurrently. Appropriate shell is loaded at login, but user can usually (except in sh, rsh) dynamically change the shell
- A UNIX command takes the form of

  executable_file [-options] arguments

- The shell runs a **command interpretation loop**
  - accept command
  - read command
  - process command
  - execute command
- Executing the command involves creating a child process running in another shell (an environment within which the process can run). This is done by Forking.
- The parent process usually waits for the child to terminate before re-entering the command interpretation loop
- Programs can be run in the background by suffixing the command-line entry with an ampersand (&). Parent will not wait for child to terminate

The Processing Environment

Input and Output

- UNIX automatically opens three files for the process

  STDIN - standard input (attached to keyboard)

  STDOUT - standard output (attached to terminal)

  STDERR - standard error (attached to terminal)

- Because UNIX treats I/O devices as special types of files, STDIO can be easily redirected to other devices and files

  who > list _of _users

The Kernel

- Central part of the OS which provides system services to application programs and the shell
- The kernel manages processes, memory, I/O and the Timer - so this is not the same as the kernel that we covered in Lecture 3!
- UNIX supports multiprogramming
- Processes have their own address space - for protection
- Each process's process environment is composed of an unmodifiable re-entrant text (code) region, a modifiable data region and a stack region.
- The text region is shareable
- Processes can modify their environment only through calls to the OS

The File System

- UNIX uses HDS with root as its origin
- A directory is a special UNIX file which contains file names and their i-nodes (index nodes)
- Subdirectories appear as file entries
- Directories cannot be modified directly, but can are changed by the operating system when files and subdirectories are created and deleted
- File and Directory names must be unique within a particular directory (i.e., the path name must be unique)
- The File System is a data structure that is resident on disk
- It contains a super block (definition of the file system); an array of i-nodes (definition of the files in the system); the actual file data blocks; and a collection of free blocks
- Space allocation is performed in fixed-size blocks

The i-node contains:

the file owner's user-id and group-id

protection bits for owner, group, and world

the block locator

file size

accounting information

number of links to the file

file type

The Block Locator

- Consists of 13 fields
- First 10 fields points directly to first 10 file blocks
- 11th field is an indirect block address
- 12th field is a double-indirect block address
- 13th field is a triple-indirect block address

Permissions

- Each UNIX file and directory has 3 sets of permission bits associated with it
- These give permissions for owner, group and world
- System files (inc. devices) are owned by root, wizard, or superuser (terminology!)
- Root has unlimited access to the entire installation - whoever owns the files!

Setuid

- When you need to change your password, you need to modify a file called /etc/passwd. But this file is owned by root and nobody other than root has write permission!
- The **passwd** command (to change passwords) is owned by root, with execute permission for world.
- The setuid is a bit which when set on an executable file temporarily gives the user the same privileges as the owner of the file
- This is similar in concept to some OS commands executing in Supervisor mode to perform a service for an otherwise unauthorised process

Process Management

- Description of Process Management in SunOS

Scheduling

- Priority-based pre-emptive scheduling. Priorities in range -20 to 20. Default 0.
- Priorities for runnable processes are recomputed every second
- Allows for ageing, but also increases or decreases process's priority based on past behaviour
- I/O-bound processes receive better service
- CPU-bound processes do not suffer indefinite postponement because the algorithm `forgets' 90% CPU usage in 5*n seconds (where n is the average number of runnable processes in the past 60 seconds)

Signals

- Signals are software equivalents to hardware interrupts used to inform processes asynchronously of the occurrence of an event

Interprocess Communication

- UNIX System V uses semaphores to control access to shared resources
- For processes to exchange data or communicate, pipes are used
- A pipe is a unidirectional channel between 2 processes
- UNIX automatically provides buffering, scheduling services and synchronisation to processes in a pipe line
- The presence of a pipe causes the processes in the pipe line to share STDIO devices

  who | grep cstaff

- The output from **who** is directed to a buffer. **grep** will take its input from this buffer. The output from **grep** will be displayed on the terminal

Timers

- UNIX makes 3 interval timers available to each process
- Each counts down to zero and then generates a signal
- The first runs continuously
- The second runs while a process is executing process code
- The third runs while the process executes process code or kernel code

Memory Management

Address Mapping (Virtual Storage) - Paged MMS

- Virtual address V is dynamically translated to real address (P, D)
- Direct Mapping is used, with the Page Map held in a high-speed RAM cache
- Each Page Map Entry contains a modified bit, an accessed bit, a valid bit (if the page is resident in PM) and protection bits
- The system maintains 8 page maps - 1 for the kernel (not available to processes) and 7 for processes (contexts)
- 2 context registers are used - one points to the running process's page map and the other to the kernel's page map
- The replacement strategy replaces the page that has not been active for longest (LRU)

Paging

- SunOS maintains 2 data structures to control paging
- The free list contains empty page frames
- The loop contains an ordered list of all allocated page frames (except for the kernel)
- The pager ensures that there is always free space in memory
- When a page is swapped out (not necessarily replaced) the system judges whether the page is likely to be used again

- If the page contains a text region, the page is added to the bottom of the free list, otherwise it is added to the top
- When a page fault occurs, if the page is still in the free list it is reclaimed

I/O

Data

- All data is treated as a byte stream
- UNIX does not impose any structure on data - the applications do
- So data can be manipulated in any way - but programmers must explicitly structure the data

Devices

- A device is just a special type of file
- These files can have protection bits, so that a printer, e.g., cannot be read
- Permission to use sensitive devices, e.g., magnetic disk, is restricted to root and all other users have to use system calls to executable files which have their setuid bit set

Case Study: Windows

- Windows OS**,** computer operating system (OS) developed by Microsoft Corporation to run personal computers (PCs). Featuring the first graphical user interface (GUI) for IBM-compatible PCs, the Windows OS soon dominated the PC market. Approximately 90 percent of PCs run some version of Windows.
- The first version of Windows, released in 1985, was simply a GUI offered as an extension of Microsoft's existing disk operating system, or MS-DOS. Based in part on licensed concepts that Apple Inc. had used for its Macintosh System Software, Windows for the first time allowed DOS users to visually navigate a virtual desktop, opening graphical "windows" displaying the contents of electronic folders and files with the click of a mouse button, rather than typing commands and directory paths at a text prompt.
- Subsequent versions introduced greater functionality, including native Windows File Manager, Program Manager, and Print Manager programs, and a more dynamic interface. Microsoft also developed specialized Windows packages, including the networkable Windows for Workgroups and the high-powered Windows NT, aimed at businesses. The 1995 consumer release Windows 95 fully integrated Windows and DOS and offered built-in Internet support, including the World Wide Web browser Internet Explorer.
- With the 2001 release of Windows XP, Microsoft united its various Windows packages under a single banner, offering multiple editions for consumers, businesses, multimedia developers, and others. Windows XP abandoned the long-used Windows 95 kernel (core software code) for a more powerful code base and offered a more practical interface and improved application and memory management. The highly successful XP standard was succeeded in late 2006 by Windows Vista, which experienced a troubled rollout and

met with considerable marketplace resistance, quickly acquiring a reputation for being a large, slow, and resource-consuming system. Responding to Vista's disappointing adoption rate, Microsoft developed Windows 7, an OS whose interface was similar to that of Vista but was met with enthusiasm for its noticeable speed improvement and its modest system requirements.

History of Windows:

Microsoft's Windows operating system was first introduced in 1985.

Windows 1

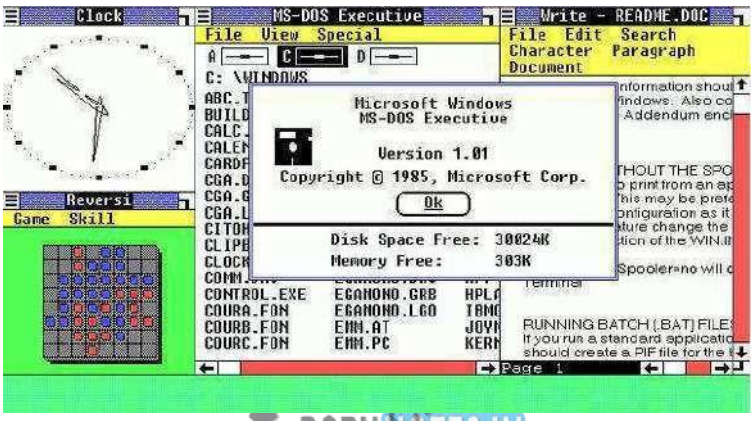


Fig. 5.6 Windows 1

This is where it all started for Windows. The original Windows 1 was released in November 1985 and was Microsoft's first true attempt at a graphical user interface in 16-bit.

Development was spearheaded by Microsoft founder Bill Gates and ran on top of MS-DOS, which relied on command-line input.

Windows 2

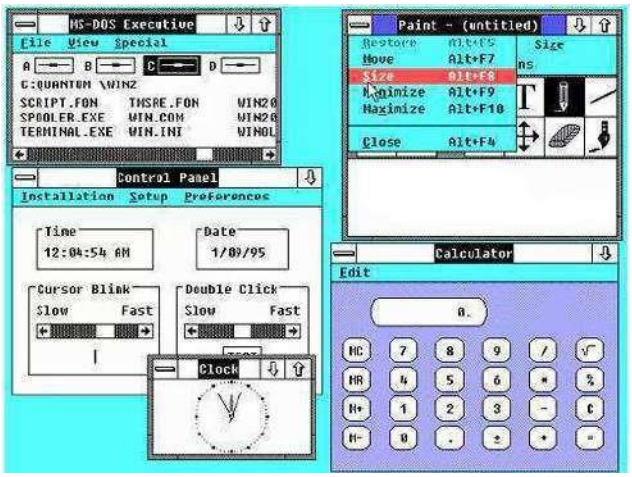


Fig. 5.6 Windows 2

Two years after the release of Windows 1, Microsoft's Windows 2 replaced it in December 1987. The big innovation for Windows 2 was that windows could overlap each other, and it also introduced the ability to minimise or maximise windows instead of "iconising" or "zooming".

The control panel, where various system settings and configuration options were collected together in one place, was introduced in Windows 2 and survives to this day.

Microsoft Word and Excel also made their first appearances running on Windows

Windows 3

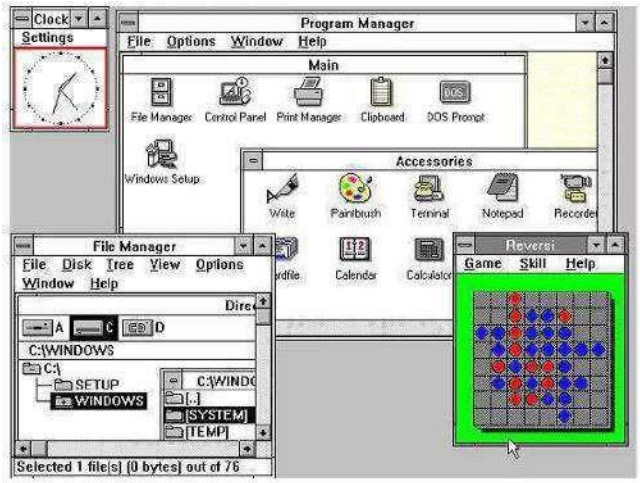


Fig. 5.7 Windows 3

The first Windows that required a hard drive launched in 1990. Windows 3 was the first version to see more widespread success and be considered a challenger to Apple's Macintosh and the Commodore Amiga graphical user interfaces, coming pre-installed on computers from PC-compatible manufacturers including Zenith Data Systems.

Windows 3 introduced the ability to run MS-DOS programmes in windows, which brought multitasking to legacy programmes, and supported 256 colors bringing a more modern, colorful look to the interface.

Windows 3.1

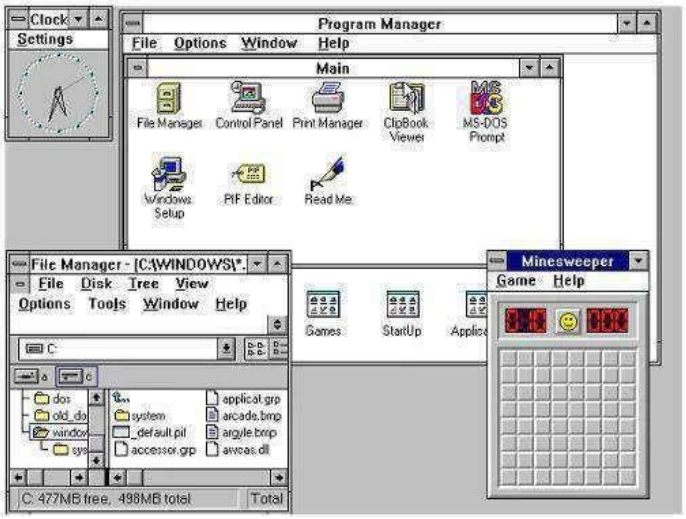


Fig. 5.8 Windows 3.1

Windows 3.1 released in 1992 is notable because it introduced TrueType fonts making Windows a viable publishing platform for the first time.

Minesweeper also made its first appearance. Windows 3.1 required 1MB of RAM to run and allowed supported MS-DOS programs to be controlled with a mouse for the first time. Windows 3.1 was also the first Windows to be distributed on a CD-ROM, although once installed on a hard drive it only took up 10 to 15MB (a CD can typically store up to 700MB).

Windows 95

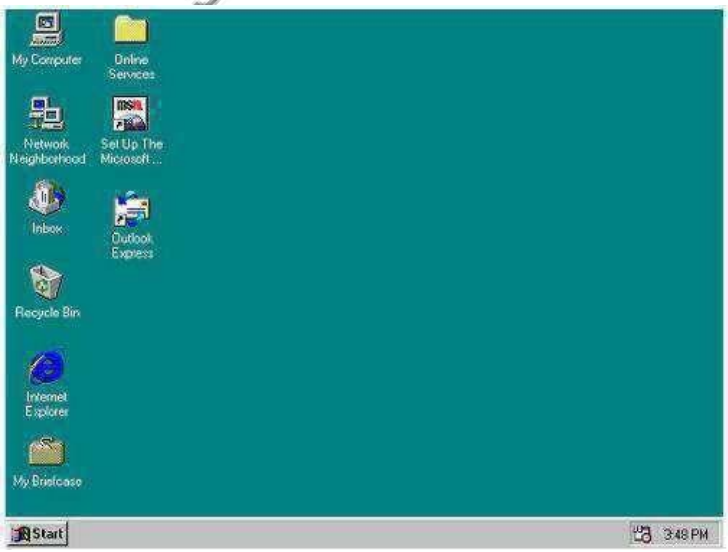


Fig. 5.9 Windows 95

As the name implies, Windows 95 arrived in August 1995 and with it brought the first ever Start button and Start menu.

Windows 98

Released in June 1998, Windows 98 built on Windows 95 and brought with it IE 4, Outlook Express, Windows Address Book, Microsoft Chat and NetShow Player, which was replaced by Windows Media Player 6.2 in Windows 98 Second Edition in 1999.

Windows ME

Considered a low point in the Windows series by many – at least, until they saw Windows Vista

– Windows Millennium Edition was the last Windows to be based on MS-DOS, and the last in the Windows 9x line.

Released in September 2000, it was the consumer-aimed operating system twined with Windows 2000 aimed at the enterprise market. It introduced some important concepts to consumers, including more automated system recovery tools.

Windows 2000

The enterprise twin of ME, Windows 2000 was released in February 2000 and was based on Microsoft's business-orientated system Windows NT and later became the basis for Windows XP.

Windows XP

Arguably one of the best Windows versions, Windows XP was released in October 2001 and brought Microsoft's enterprise line and consumer line of operating systems under one roof.

It was based on Windows NT like Windows 2000, but brought the consumer-friendly elements from Windows ME. The Start menu and task bar got a visual overhaul, bringing the familiar green Start button, blue task bar and vista wallpaper, along with various shadow and other visual effects.

Windows Vista

Windows XP stayed the course for close to six years before being replaced by Windows Vista in January 2007. Vista updated the look and feel of Windows with more focus on transparent elements, search and security. Its development, under the codename "Longhorn", was troubled, with ambitious elements abandoned in order to get it into production.

Windows 7

Considered by many as what Windows Vista should have been, Windows 7 was first released in October 2009. It was intended to fix all the problems and criticism faced by Vista, with slight tweaks to its appearance and a concentration on user-friendly features and less "dialogue box overload".

Windows 8

Released in October 2012, Windows 8 was Microsoft's most radical overhaul of the Windows interface, ditching the Start button and Start menu in favour of a more touch-friendly Start screen.

The new tiled interface saw programme icons and live tiles, which displayed at-a-glance information normally associated with "widgets", replace the lists of programmes and icons. A desktop was still included, which resembled Windows 7.

Windows 8.1

A free point release to Windows 8 introduced in October 2013, Windows 8.1 marked a shift towards yearly software updates from Microsoft and included the first step in Microsoft's U-turn around its new visual interface.

Windows 10

Announced on 30 September 2014, Windows 10 has only been released as a test version for keen users to try. The "technical preview" is very much still a work in progress.

Windows 10 represents another step in Microsoft's U-turn, bringing back the Start menu and more balance to traditional desktop computer users.

Windows operating system structure

The design of operating system architecture traditionally follows the separation of concerns principle. This principle suggests structuring the operating system into relatively independent parts that provide simple individual features, thus keeping the complexity of the design manageable.

Besides managing complexity, the structure of the operating system can influence key features such as robustness or efficiency:

- The operating system possesses various privileges that allow it to access otherwise protected resources such as physical devices or application memory. When these privileges are granted to the individual parts of the operating system that require them, rather than to the operating system as a whole, the potential for both accidental and malicious privileges misuse is reduced.
- Breaking the operating system into parts can have adverse effect on efficiency because of the overhead associated with communication between the individual parts. This overhead can be exacerbated when coupled with hardware mechanisms used to grant privileges.

1. **Simple Structure**

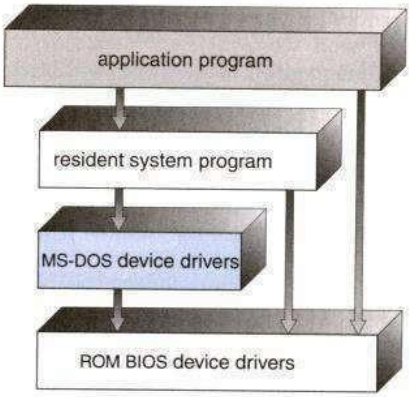


Fig. 5.10 Simple Structure

In MS-DOS, applications may bypass the operating system.

- Operating systems such as MS-DOS and the original UNIX did not have well-defined structures.
- There was no CPU Execution Mode (user and kernel), and so errors in applications could cause the whole system to crash.

2. **Monolithic Approach**

- Functionality of the OS is invoked with simple function calls within the kernel, which is one large program.
- Device drivers are loaded into the running kernel and become part of the kernel.

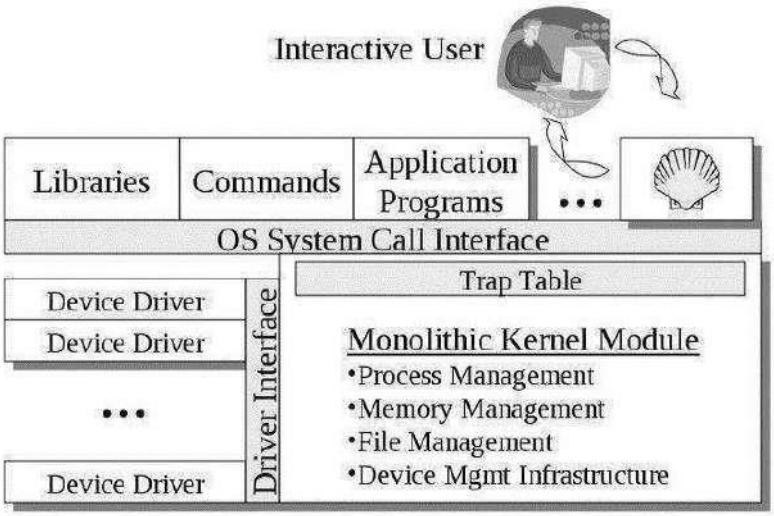


Fig. 5.11 Monolithic Structure

3. **Layered Approach**

This approach breaks up the operating system into different layers.

- This allows implementers to change the inner workings, and increases modularity.
- As long as the external interface of the routines doesn't change, developers have more freedom to change the inner workings of the routines.
- With the layered approach, the bottom layer is the hardware, while the highest layer is the user interface.
- The main advantage is simplicity of construction and debugging.
- The main difficulty is defining the various layers.
- The main disadvantage is that the OS tends to be less efficient than other implementations.

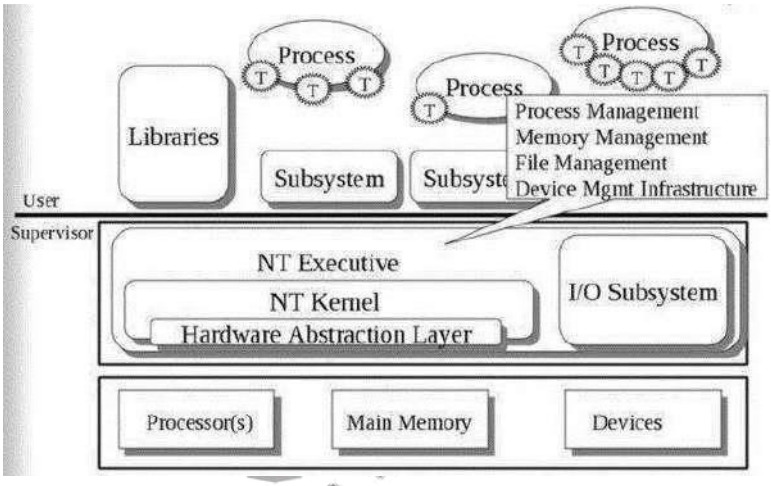


Fig. 5.12 Layered approach

The Microsoft Windows NT Operating System has the lowest level is a monolithic kernel, but many OS components are at a higher level, but still part of the OS.

4. **Microkernels**

This structures the operating system by removing all nonessential portions of the kernel and implementing them as system and user level programs.

- Generally they provide minimal process and memory management, and a communications facility.
- Communication between components of the OS is provided by message passing.

The benefits of the microkernel are as follows:

- Extending the operating system becomes much easier.
- Any changes to the kernel tend to be fewer, since the kernel is smaller.
- The microkernel also provides more security and reliability.

Main disadvantage is poor performance due to increased system overhead from message passing.

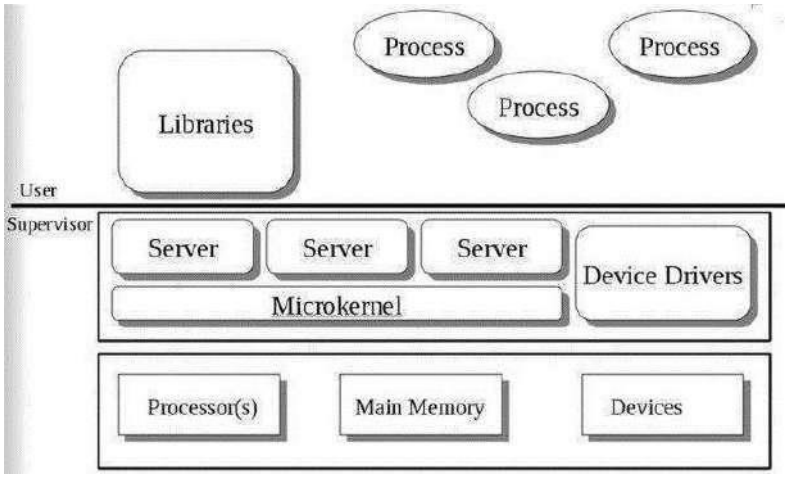


Fig. 5.13 Micro Kernel

Memory management

Every Windows administrator has to field user complaints about client performance. Client-system performance can be affected by factors such as memory, CPU, disk and the network. Of these factors, the most confusing is memory management, which admins need to understand for making informed decisions and troubleshooting. Users typically equate adding memory to resolving performance bottlenecks, and it's relatively cheap and easy to add memory.

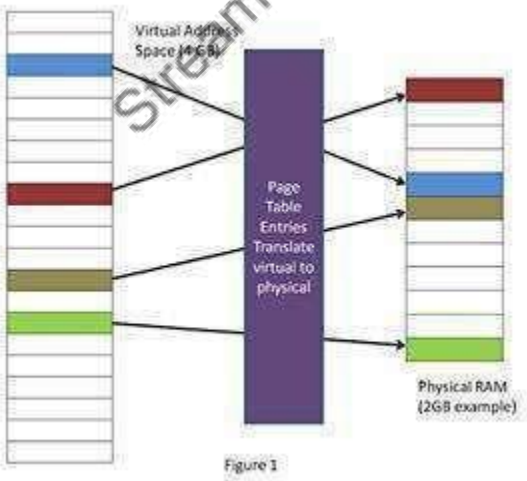


Fig. 5.14 Memory management

Above Fig. shows the memory component of the Windows XP and Windows 7 Task Manager. Note that there are fundamental differences between Windows XP, Vista and Windows 7 Task Manager versions.

It's important to know the difference between physical and virtual memory. Physical memory is the amount of physical RAM available in the computer. Physical memory can be visualized as a

table shown in below Figure, where data is stored. Each cell shown in the table is a unique "address" where data is stored.

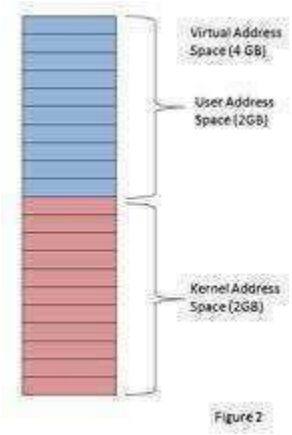


Fig. 5.15 Memory management

Virtual memory essentially allows each process -- applications, dynamic link libraries (DLLs), etc.

To operate in a protected environment where it thinks it has its own private address space. Figure 1 shows the virtual memory table for a process on a computer with 2 GB of RAM. The CPU translates or maps the virtual addresses into physical addresses in RAM using page table entries (PTEs).

Virtual memory limits

The virtual address space for 32-bit architecture has a physical limit of about 4 GB, regardless of the amount of RAM in the computer. Windows divides this into two sections, as shown in Figure 2: user space and kernel space. The addresses in the kernel space are reserved for system processes. Only those in the user space are accessible for applications. So, each application has a virtual memory limit of 2 GB. Again, this is regardless of physical RAM. That means that no process can ever address more than 2 GB of virtual address space by default. Exceeding this limit produces an "out of virtual memory" error and can occur even when plenty of physical memory is available.

Note that, as shown in Figure 2, the use of virtual memory allows the three applications, each with 2 GB of virtual address space, to share the 2 GB RAM in the computer. This is accomplished by paging infrequently used data to disk, then paging it back to RAM when needed.

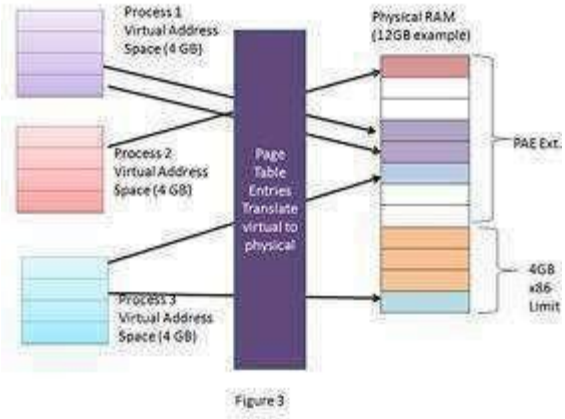


Fig. 5.16 Memory management

Processes will run faster if they reside in memory as opposed to requiring the memory manager page data in from the disk and put it back in memory. Thus, more memory in the system allows more processes to reside in memory and reduces paging from disk.

4. Physical memory management

- Windows reports how much physical memory is currently installed on your computer along with how much memory is available to the operating system and the hardware reserved memory.
- Windows may show that the usable memory may be less than the installed memory (RAM). The indicative Usable memory is a calculated amount of the total physical memory minus "hardware reserved" memory.

Physical Memory

One of the most fundamental resources on a computer is physical memory. Windows' memory manager is responsible with populating memory with the code and data of active processes, device drivers, and the operating system itself. Because most systems access more code and data than can fit in physical memory as they run, physical memory is in essence a window into the code and data used over time. The amount of memory can therefore affect performance, because when data or code a process or the operating system needs is not present, the memory manager must bring it in from disk.

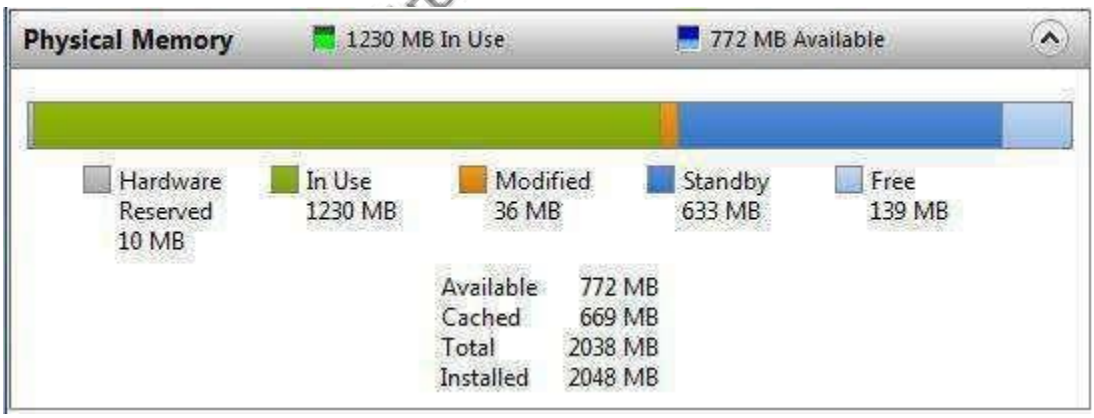
Besides affecting performance, the amount of physical memory impacts other resource limits. For example, the amount of non-paged pool, operating system buffers backed by physical memory, is obviously constrained by physical memory. Physical memory also contributes to the system virtual memory limit, which is the sum of roughly the size of physical memory plus the maximum configured size of any paging files. Physical memory also can indirectly limit the maximum number of processes, which I'll talk about in a future post on process and thread limits.

Memory allocation	Description
Hardware Reserved	Memory that is reserved for use by the BIOS and some drivers for other peripherals
In Use	Memory that is used by process, drivers, or the operating system
Modified	Memory whose contents must go to disk before it can be used for another purpose
Standby	Memory that contains cached data and code that is not actively in use
Free	Memory that does not contain any valuable data and that will be used first when processes, drivers, or the operating system need more memory

Memory allocation	Description
Available	Amount of memory (including standby and free memory) that is immediately available for use by processes, drivers, and the operating system
Cached	Amount of memory (including standby and modified memory) that contains cached data and code for rapid access by processes, drivers, and the operating system
Total	Amount of physical memory that is available to the operating system, device drivers, and processes
Installed	Amount of physical memory installed in the computer

To find out how memory is being used on your computer, type **Resource Monitor** in start search and hit Enter.

Click the Memory tab, and view the Physical Memory section at the bottom of the page.



5. Memory management system calls.

System Calls

The system call provides an interface to the operating system services.

Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained:

- Portability: as long a system supports an API, any program using that API can compile and run.
- Ease of Use: using the API can be significantly easier then using the actual system call.

System Call Parameters

Three general methods exist for passing parameters to the OS:

1. Parameters can be passed in registers.
2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
3. Parameters can also be pushed on or popped off the stack by the operating system.

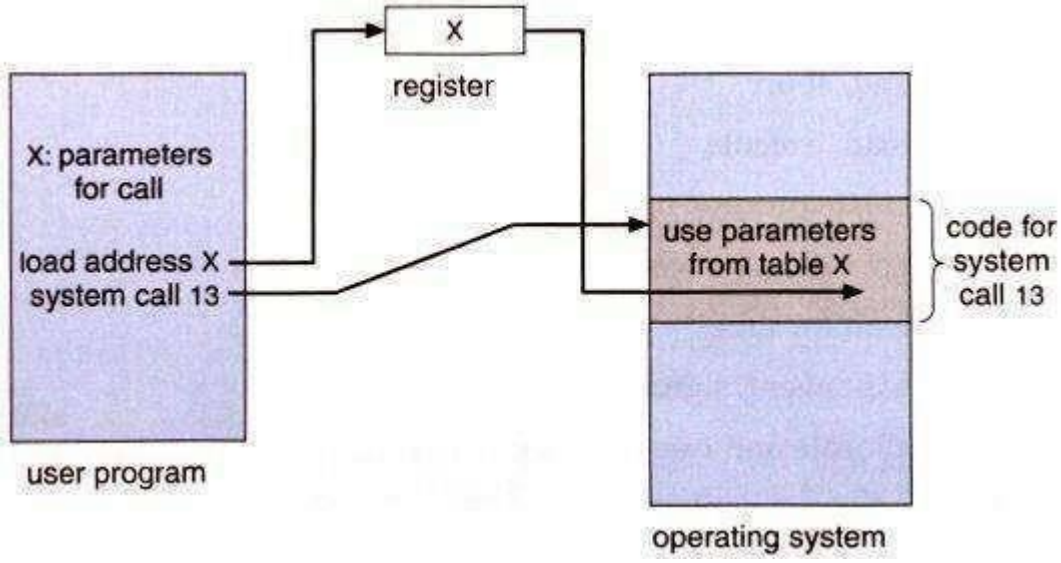


Fig. 5.17 System Call

Types of System Calls

There are 5 different categories of system calls:

Process control, file manipulation, device manipulation, information maintenance and communication.

Process Control

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

File Management

Some common system calls are create, delete, read, write, reposition, or close. Also, there is a need to determine the file attributes – get and set file attribute. Many times the OS provides an API to make these system calls.

Device Management

Process usually requires several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs request the device, and when finished they release the device. Similar to files, we can read, write, and reposition the device.

Information Management

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is time, or date.

The OS also keeps information about all its processes and provides system calls to report this information.

Communication

There are two models of interposes communication, the message-passing model and the shared memory model.

- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.