1

Operating System Basics

An operating system is an integrated set of programs that acts as an intermediary between the user and computer hardware. The primary goal of an operating systems is to make the computer system **convenient** to use in an **efficient** manner. A secondary goal is to **manage the resources** of a computer system.

In this chapter, we will learn about operating systems, its need, its functions, and its types.

1.1 *What is an Operating System ?*

An operating system is an integrated set of specialised programs that are used to manage over all resources of and operations of the computer. It is specialised software that controls and monitors the execution of all other programs that reside in the computer, including application programs and other system software. It is the interface between the user and computer as shown in fig. 1.1

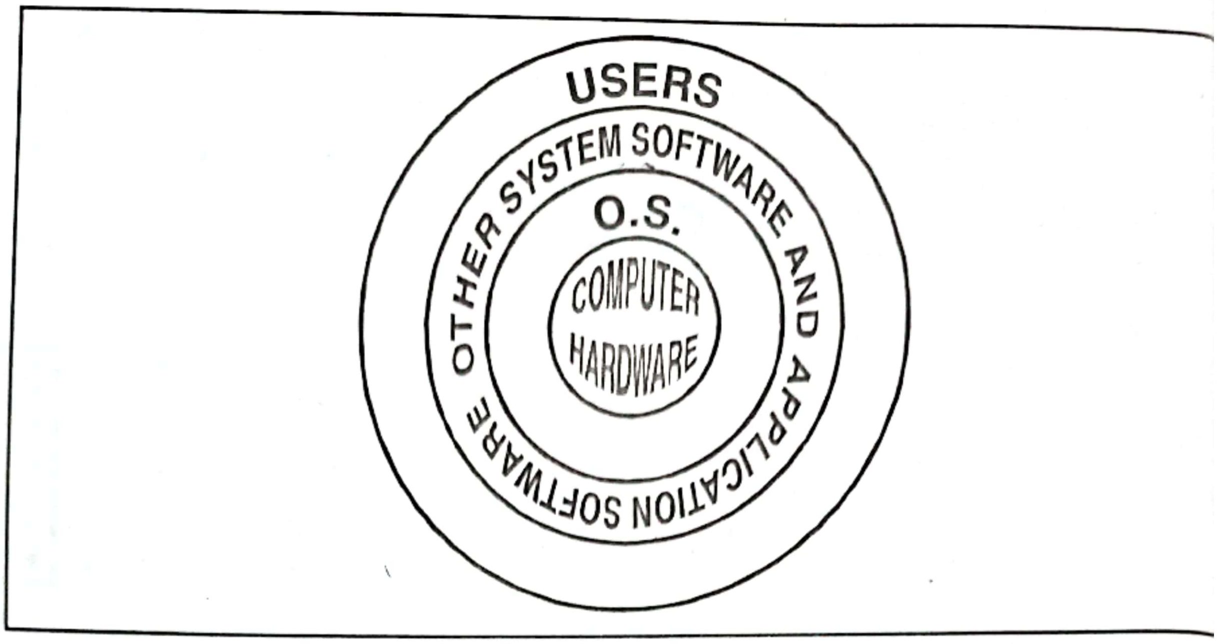


FIGURE 1.1

A simple way of defining the operating system can be :

An operating system is a program that acts as an interface between the use and the computer hardware and controls and manages the overall resource of Computer system.

A computer system has many resources (hardware and software) that may b required to solve a problem.

According to this definition, the two primary **objectives**, or **goals** of an operatin system are :

1. **Making a computer system convenient to use in an efficient manner**

   The operating systems hides the details of the hardware resources from the users and provides the users with a convenient interface for using the computer system. It acts as an intermediary between the hardware and its users and making it earsier for the users to access and use other resouces.

   As shown in figure 1.1, the hardware resources are surrounded by the operating system layer, which in turn is surrounded by a layer of other system software (such as assemblers, compilers, interpreters, etc.) and application softwares (suh as Word processors, spread sheets, Accounting software, etc.). Finally, the users view the computer system in terms of the user interfaces provided by the operating systems.

The **efficient** operation of the computer system is particularly important for large, shared multiuser systems. These systems are very expensive, so it is desirable to make them as efficient as possible.

2. **Managing the resources of a Computer System**

This objective involves performing such tasks as keeping track of who is using which resource, granting resource requests, accounting for resource usage and mediating conflicting requests from different programs and users. The efficient and fair sharing of resources among users and/or programs is a key goal of most operating systems.

1.1 The Operating System as an Extended Machine

An average programmer probably does not want to get too initmately involved with the programming of device like **floppy disks, hard disks** etc. Instead, what the programmer wants is a simple, high-level abstraction to deal with. In the case of disks, a typical abstraction would be that the disk contains a collection of named files. Each file can be opened for reading or writing, then read or written, and finally closed.

The program that hides the truth about the hardware from the programmer and presents a nice, simple view of named files that can be read and written is, of course, **the operating system**. Just as the operating system shields the programmer from the disk hardware and presents a simple file-oriented interface. Therefore, the abstraction offered by the operating system is simpler and easier to use than that offered by the underlying hardware.

In this view, the function of the operating system is to present the user with the equivalent of an **extended machine** or **virtual machine** that is easier to program than the underlying hardware.

Figure 1.1 shows the logical architecture of computer system. The computer hardware resoures are surrounded by the operating system layer which in turn is surrounded by a layer of other system softwares such as compilers, editors, assemblers, loaders etc. and by a set of application programs such data processing applications, entertainment and education applications etc. Finally, the end users view the computer system in terms of the user interfaces.

1.1.2 The Operating System as a Resource Manager

The job of an operating system is to provide an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs competing for them. The resource sharing includes sharing resources in two ways **in time**

and in space. Time management includes allocation of CPU, printer and o resources to various programs and space management deals with the mem allocation to multiple programs.

In addition, users often need to share not only hardware, but information (like fi database etc.) as well. Operating system keep track of who is using which resou to grant resource requests, to account for usage, and to mediate conflicting reque from different programs and users..

For example, in a multiuser networked environment, imagine what would happe three programs running on some computer all tried to print their output simultaneou on the same printer. The first few lines of printout might be from program 1, the n few from program 2, then some from program 3, and so forth. The result would chaos. The operating system can bring order to the potential chaos by buffering the output destined for the printer on the disk. When one program is finished, t operating system then copy its output from the disk file where it has been stored the printer, while at the same time the other program can continue generating mo output (not to the printer yet). As **Resource Manager** its basic functions are show in figure 1.2.

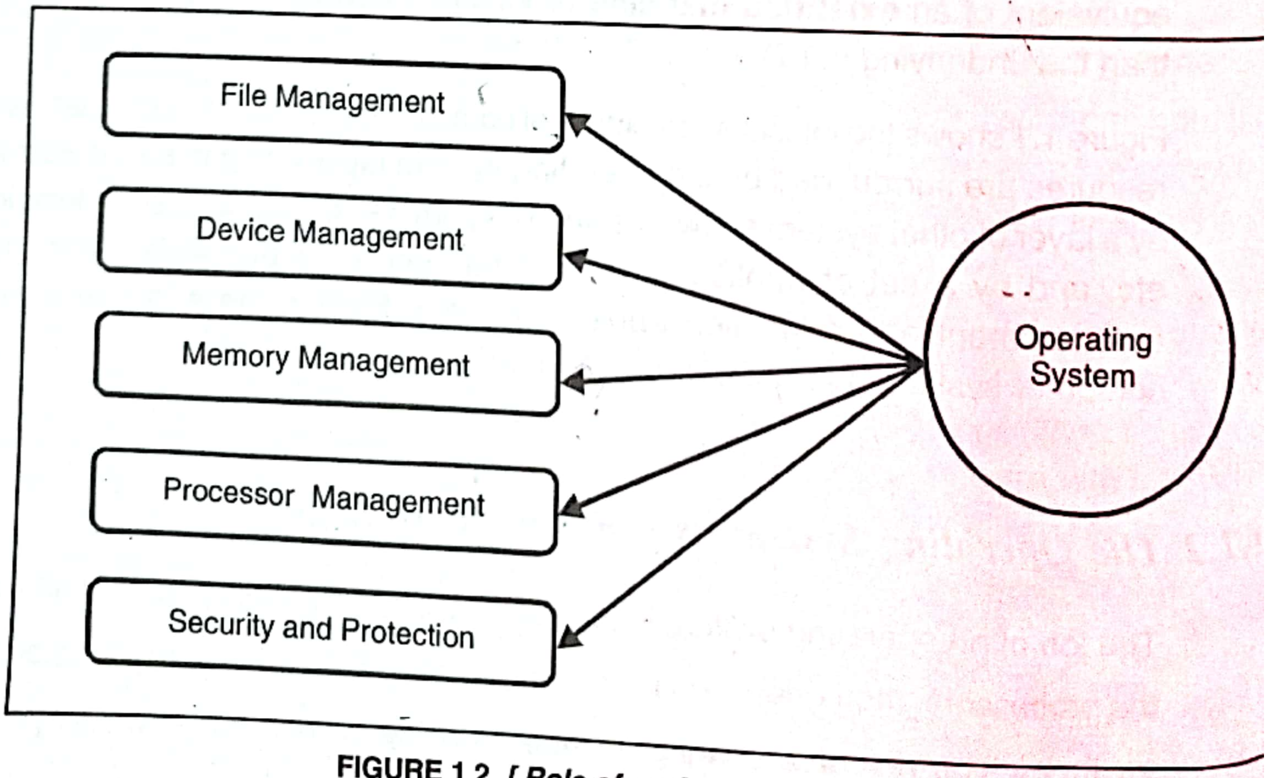


FIGURE 1.2 ***[Role of an Operating System]***

1.2 FUNCTIONS OF OPERATING SYSTEM

Functions of Operating Systems are :

1. **Memory (Storage) Management**
   - (a) It keeps tracks of primary memory i.e. what part of it are in use by whom, what part are not in use etc.
   - (b) In multiprogramming it decides which process will get memory when and how much.
   - (c) Allocates the memory when the process or program request it to do so.
   - (d) Declaims (deallocate) the memory when the process no longer needs it or has been terminated.
2. **Processor Management**
   - (a) Keep tracks of processor and status of process. Program that does this is called **traffic controller.**
   - (b) In multiprogramming it decides which process gets the processor when & how much time. This function is called **Process Scheduling.**
   - (c) Allocate the processor (CPU) to a process.
   - (d) Deallocate processor when processor is no longer required.
3. **Device Management**
   - (a) Keeps tracks of all devices (peripherals). This is also typically called the **I/O controller.**
   - (b) Decides which process gets the device when & for how much time.
   - (c) Allocate the device in the efficient way.
   - (d) Deallocate devices.
4. **File Management**
   - (a) It keeps track of information, its location, uses, status etc. The collective facilities are often known as **file system.**
   - (b) Decides who gets the resources.
   - (c) Allocates the resources.
   - (d) Deallocates the resources.

5. **Secondary Storage Management**

Main memory as a limited size and it is volatile i.e. it looses its contents soon as power turns off. The operating system performs following functior regard to secondary storage management.

(a) Free space management i.e., manages free space on the second storage devices by reclaiming memory from used objects.

(b) Allocation of storage space when new files have to be written.

(c) Scheduling the requests for memory access i.e. Disk Scheduling.

(d) Creation and Deletion of files.

6. **Security and protection**

By means of passwords & similar other techniques, preventing unauthoriz access to programs & data.

Protection involves ensuring that all access to system resources is controlle

7. **Network Management**

An operating system works as a network resource manager when multipl computers are in a network or in a distributed architecture.

8. **Control over system performance**

Recording delays between request for a service & response from the system

9. **Job accounting**

Keeping track of time & resources used by various jobs and/or users.

10. **Interaction with the operators**

The interaction takes place via the console of the computer in the form of instructions from the operator acknowledging the same, action thereon, as well as informing the operation by means of a display screen of works & problem encountered.

11. **Error-detecting aids**

Production of dumps, traces, error messages and other debugging and error-detecting aids.

12. **Coordination between other softwares and users**

Coordination and assignment of compilers, interpretiers, assemblers and other software to the various users of the computer systems.

1.3 *Services of Operating System*

Commonly used services of operating systems are :

1. **Program execution**

   The system must be able to load a program into memory and to run it. The program must be able to end its execution, either normally or abnormally (indicating error).

2. **File-System manipulation**

   User can directly create, delete, rename, copy etc. files by using the appropriate commands as specified by the operating system.

3. **I/O Opeartions**

   A running program may require I/O. For specific devices, special functions may be desired (such as rewind a tape drive, or blank the screen on a CRT). Users usually cannot control I/O devices directly. Therefore, the operating system provides some means to do I/O.

4. **Communications**

   Operating Systems implements the communications between processes via **shared memory** or by the technique of **message passing**, in which packets of information are moved between processes.

5. **Resource allocation**

   In multiuser or multiprogramming, operating system decides which process will get the resource, when and how much time.

6. **Accounting**

   Sometime we want to keep track of which users use, how much and what kinds of computer resources. Operating system provides all details.

7. **Security and protection**

   For security, Login/Logout commands are used in multiuser environment for logging into or logging out of the system.

   Protection involves ensuring that all access to system resources is controlled.

8. **Error Detection**

   Errors may occur in the CPU, In memory, in I/O devices or in the user program. For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

1.4 Some Examples of Operating System

Some examples of operating systems are :

1. **MS-DOS (Micro - Soft Disk Operation system)**

   DOS was introduced in 1981 & has undergone many upgrades. The late (1995) DOS version is 6.22 which has advanced memory & file manageme capabilites, as well as virus protection facilities. DOS is a single-user, singl tasking O.S.

2. **CP/M (Control Program for Micro Processors)**

   This OS is single user & meant for 8-bit micros.

3. **Windows : Windows has two verison**

   - DOS With Windows &
   - Windows NT

   (a) **DOS with Windows :** It extends the capabilities of **DOS** by creating a operational environment. Actually, it is not an independent O.S. & require DOS to run it. Its capabilities include multi-tasking, enhanced memory & graphical user interface.

   (b) **Windows NT :** It is an OS by itself. It has get to gain adequate popularity Its capabilities include multiprocessing, multitasking, networking flexibility & graphical user interface. It has more memory. It is also a multiuser OS.

4. **OS/2**

   It was jointly developed by IBM & Microsoft Corportion. It requires at least the power of intel's 80386 processor. Its capabilities include multiasking, enhanced memory, networking flexibility & graphical user interface.

5. **Macintosh**

   The OS is contained in two primary files known as system file and finder. The O.S. procedures include such tasks as formatting disk, copying files, erasing files etc. & running application programs. These files also manage the user interface, displaying menus and activating tasks that are chosen from the menus. Its main advantage is ease of use, quaility graphics, multitasking & communication among programs.

6. **UNIX**

The programmes written on Unix OS are termed as portable. Unix OS can be used on microcomputers, mini computers & is multi user, multi processing & multi tasking OS.

The OS have very good networking facilities & is very popular on microcomputers called work stations.

1.5 *Measuring System Performance*

Following terms are used to measure the efficiency of an operating systems and the overall performances of a computer systems.

1. **Throughput**

Throughput is the amount of work that the system is able to do per unit time. For example, if *n* processes are completed in an interval of *t* seconds, then the throughput is taken as *n/t* processes per second.

2. **Turnaround time**

Turnaround time is the interval from the time of submission of a job to the system for processing to the time of completion of the job.

3. **Response time**

Response time is the interval from the time of submission of a job to the system for processing to the time the first response for the job is produced by the system. Response time is better measure as compared to turnarround time.

1.6 *Types of An Operating System (According to the number of users)*

There are two types of O.S.

1. **Single User Systems :** In this kind of system the processor or a computer does only one job at a time. That is at one point of time only one task can be performed.

   For Example : MS-DOS

   The **disadvantage** of single user operating system is that CPU sits idle for most of the time and is not fully utilized.

2. **Multi User System :** This type of system is used by more than one user as shown in figure 1.3. It allows the interaction of one user with the other. It also helps in executing more than one task at a particular time.

   For Example : Unix and Windows NT.

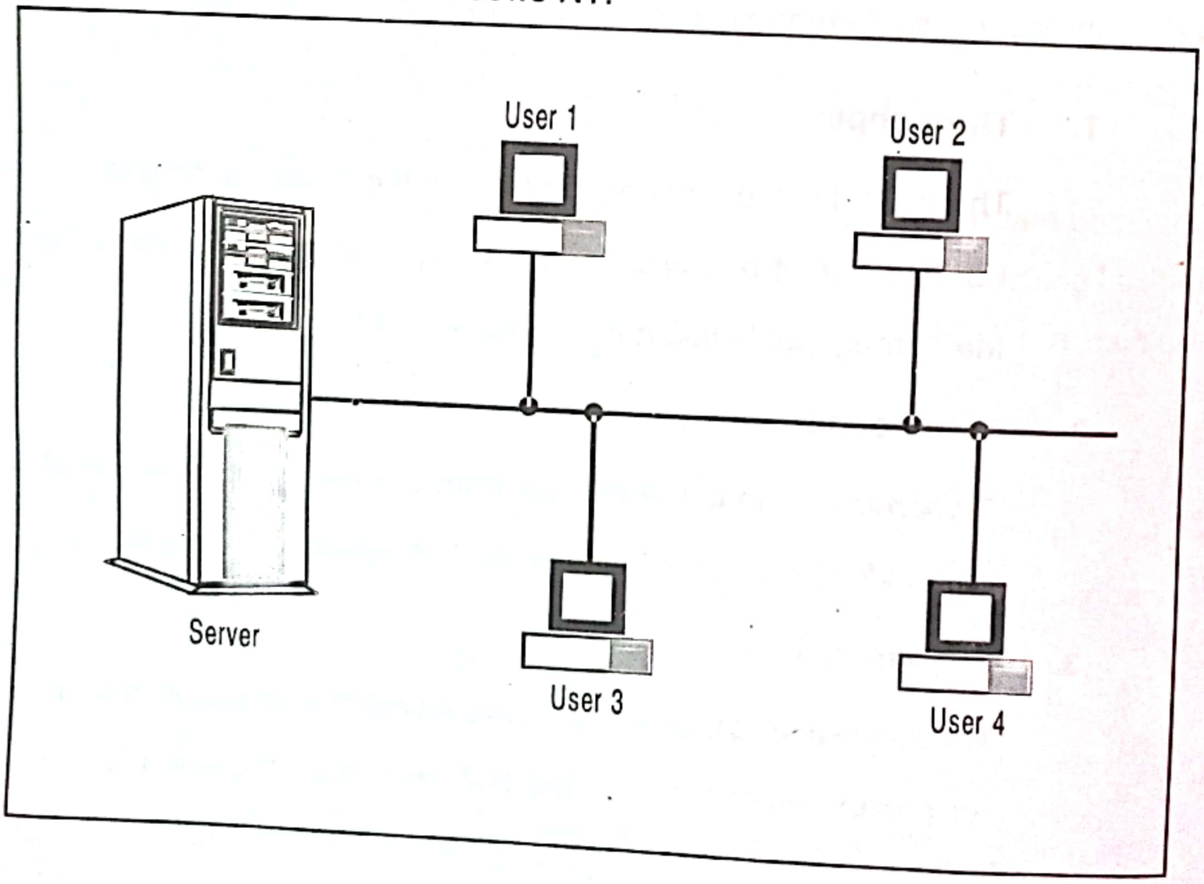


FIGURE 1.3

1.6.1 *Difference Between Single User & Multi User O.S.*

SINGLE USER (OS)	MULTI USER (OS)
1. Single person can use at a time	1. More than one user can use at a time
2. Standalone	2. Many systems are connected with one system
3. More secure	3. Less secure
4. No sharing of data	4. Sharing of data
5. Less ~~More~~ interactive	5. More ~~Less~~ interactive
6. Simple	6. Complex
7. Cheap	7. Costly
8. Less functions	8. More functions
9. Small in size	9. Large in size
10. Examples are DOS, Windows	10. Examples are Windows NT, Unix

1.7 *Historical evolution of Operating System*

Operating systems have evolved through a number of distinct generations which corresponds roughly to the decades.

1. **The 1940's - First Generations**

   The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). The individual user was responsible for all machine set up and operation. Programming languages were unknown. Operating systems were unheard of.

2. The 1950's - Second Generation

By the early 1950's, the routine had improved somewhat with the introductio of punch cards. The General Motors Research Laboratories implemented th first operating systems in early 1950's for their IBM 701. **Early Computer** could perform only one job at a time and the users often had to wait for lon time to complete their job. The operating system of the 50's generally ran on job at a time. This system was named as **batch processing systems.**

Note : Main operating system of this generation is Batch processing system.

3. The 1960's - Third Generation

In the year of 1960's, the operating system designers thought that the single user batch processing system was inefficient because most of the time was spent waiting for the slow I/O devices to complete their tasks. Therefore, operating system designers developed the concept of **multiprogramming** in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several job advancing while keeping the peripheral devices in use.

Another major feature in third-generation operating system was the technique called **spooling** (simulteneous peripheral operations on line).

Another feature present in this generation was **time-sharing technique.**

Note : Operating System of this generations are :

- Multiprogramming operating system
- Time sharing operating system
- Multitasking operating system

4. Fourth Generation and Present

With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it become possible to build desktop computers as powerful as the mainframes of the 1970's. Two operating systems have dominated the personal computer **MS-DOS,** and **UNIX.**

In this generation, the **distributed operating system** were developed for computers that could run tasks on several interconnected processors.

In the mid 1980's the **network operating system** was introduced for connecting the compuıers in a local area network.

Today's operating systems provide the fundamental services to the users so that the computer hardware can be accessed easily and the hardware resources can be shared among different users. The multitasking operating sytem such as UNIX and Windows, divide the job among multiple processes in the form of **threads**. Also on single user system, the multitasking enables the computer to perform multiple tasks at the same **multi-threading.**

Therefore, now-a-days we use the concept of multi-threading. We also use the concept of Parallel Processing operating system.

Note : Main Operating Systems of this generation are :

- Distributed operating system
- Network operating system
- Parallel operating system

We will discuss all these operating systems in detail in artical **Classification of operating system.**

1.8 CLASSIFICATION OF OPERATING SYSTEM

Operating system may be classified in the following categories :

(i) Early O.S. or Serial Processing O.S.

(ii) Batch O.S.

(iii) Interactive O.S.

(iv) Real Time O.S.

(v) Multiprogramming O.S.

(vi) Timesharing O.S.

(vii) Multiprocessing O.S.

(viii) Multitasking O.S.

(ix) Distributed O.S.

(x) Network O.S.

1.8.1 *Early O.S. or serial processing O.S.*

In serial processing operating system only one job resides in computer memor and it remains there till it is executed. After complition of job, next job is entered i serial fashion.

In this system, memory management is very simple because whole memory space is allocated to the job or program as shown in figure 1.4

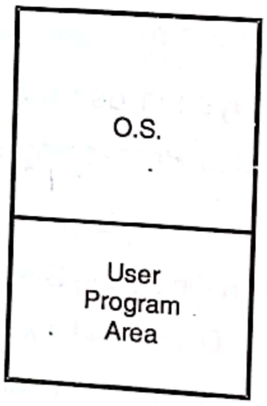


FIGURE 1.4

In serial processing operating system, the CPU remain idle for most of the time. because CPU is very fast as compare to I/O devices.

Advantages

1. Resource management is very easy.
2. Resource allocation is very easy.

Disadvantages

1. CPU remains ideal most of the time.
2. Very slow
3. Waiting time of jobs are more.

1.8.2 Batch O.S.

The users of batch Operating System do not allow to interact the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator.

The operator sorts jobs into batches with similar requirements and runs each batch on the computer. When the job is complete its output is sent back to the appropriate user. In batch system, major task of operation system is to transfer control automatically from one job to the next as shown in figure 1.5

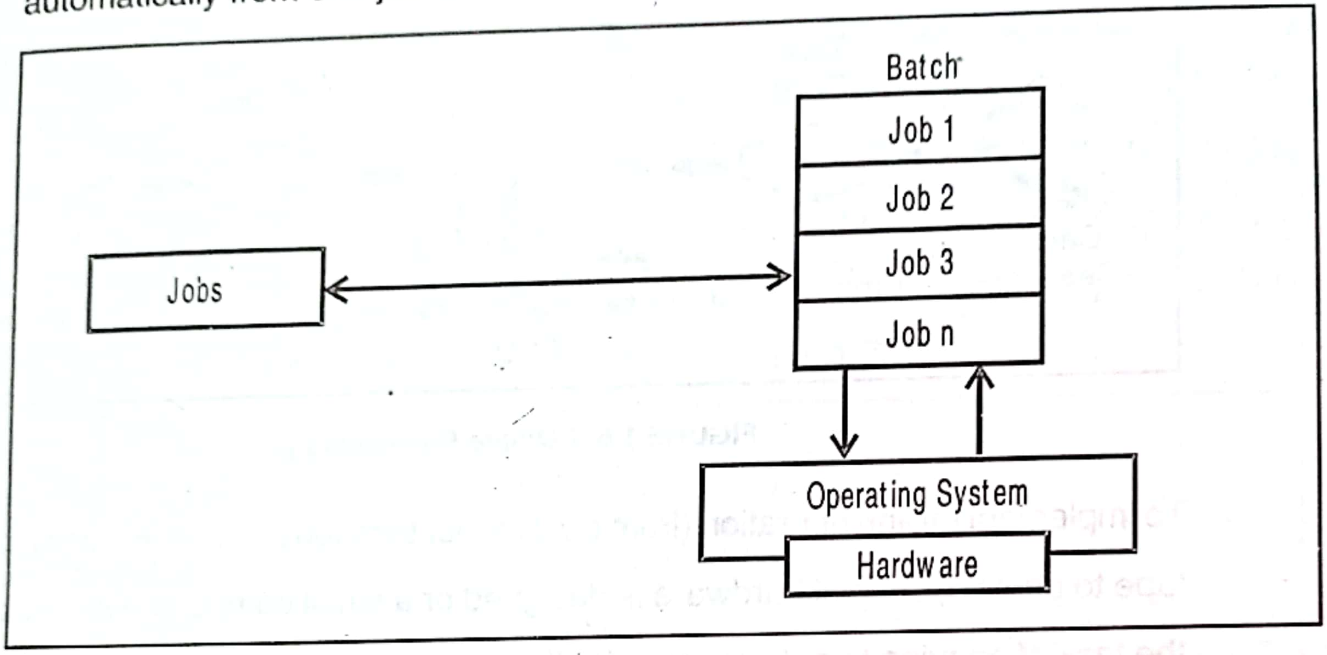


FIGURE 1.5

Advantages

1. Resource Management and allocation is very easy.
2. Faster than serial processing.

Disadvantages

1. Lack of interaction between the user and the job while job is executing.
2. Turnaround time is high. The delay between job submission and job completion is called turnaround time.
3. Utilisation of CPU is very poor. The CPU is often idle.
4. Difficult to provide the desired priority.

We know that the speed of I/O devices is much low as compared to fast processors. It means that CPU often waits for an I/O. Here we will discuss some techniques that overcome this problem.

1.8.2.1 *Offline Processing*

In this we replace the slow card readers and line printer with magnetic tapes. Earli cards were directly brought to the system but in this, cards were first copied magnetic tape and when the tape is sufficiently full, it is submitted to the compute Similarly, output was written to tape and the contents of tape would be printed late as shown in figure 1.6

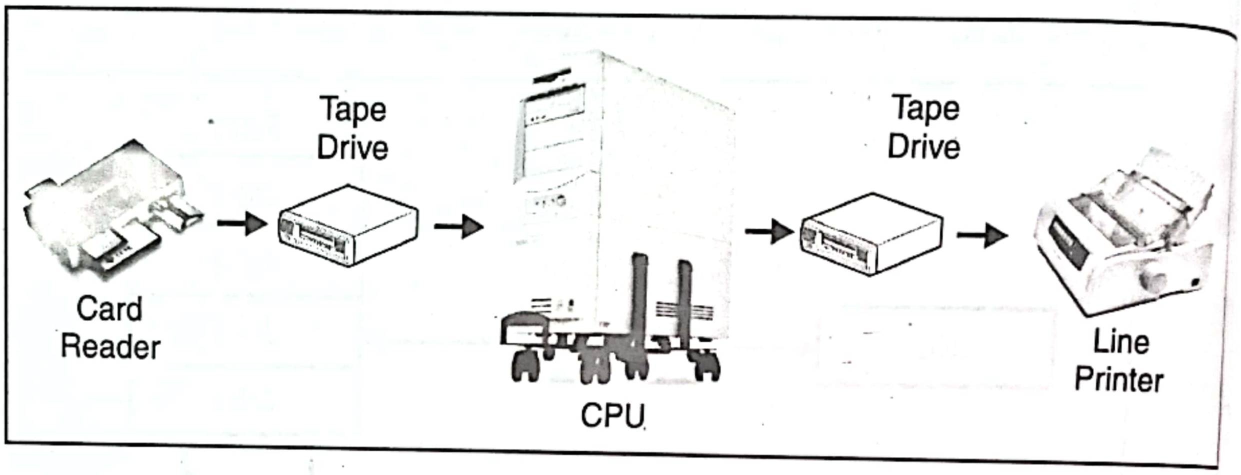


FIGURE 1.6 ***[Offline Processing]***

To implement off line operation (from card reader to magnetic tape and from magnetic tape to printer), special hardware is designed or a small computer was dedicated to the task of copying to or from tape. So the main advantage in offline is that it makes a bridge between the fast processors and slow card readers.

1.8.2.2 *Buffering*

Buffering is that method in which input, output and CPU act simultaneously. In this when CPU processes first record, the input buffer is instructed to accept the next input immediately. Similarly, buffering can be used for an output device. In this, the output is put onto a buffer until an output device can accept it.

Buffering increases the resource utilization but it can be difficult to code. The very first problem with buffering is that the next I/O operation can be started only when first I/O operation is finished. Interrputs are used to solve this problem.

1.8.2.3 Spooling

SPOOLING stands for **Simultaneous Peripheral Operation On Line.** It is most sophisticated form of buffering. In spooling the disk space is used as a very large buffer for reading, storing and output files.

In disk technology, rather than the cards being read from the card reader directly into memory, and then the job being processed, cards are read directly from the card reader onto the disk. The location of card images is recorded in a table kept by the operating system. When a job is executed, the operating system satisfies its requests for card-reader input by reading from the disk. Similarly, when the job requests the printer to output a line, that line is copied into a system buffer and is written to the disk. When the job is completed, the output is actually printed. This form of processing is called *spooling* as shown in fig. 1.7

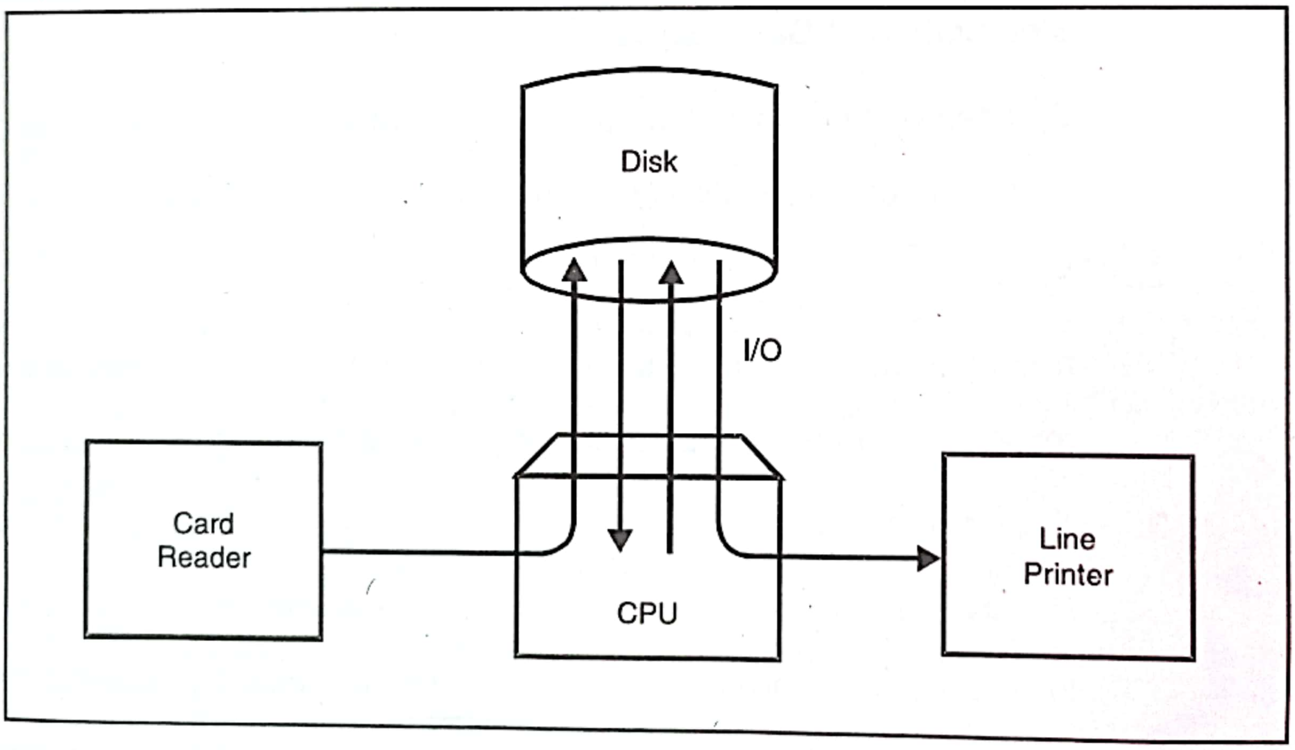


FIGURE 1.7 ***[Spooling]***

Thus, Spooling keeps busy both CPU and I/O devices at much high speeds.

The **main difference between spooling and buffering is** that spooling overlaps the I/O of a job with the computation of other job whereas buffering overlaps the I/O of a job with its own computation.

1.8.3 Interactive (Or on-line) O.S.

An interactive system provides on-line communication between the user and system. The user gives instructions to the operating system or to a program direc and receives an immediate response. Usually, a keyboard is used to provide inp and a monitor is used to provide output. When the operating system finishes execution of one instruction (or command), it seeks the next instruction from user's keyboard.

Applications of On-line O.S

Systems are being developed and are already in use for a wide range of applicatior in different types of industries discussed below:

1. **Electricity and Gas Boards**

   By means of terminals situated in showroom it is possible to inform perspectiv customers of the availability of the appliances in response to their enquiries

2. **Banking**

   It is possible to inform bank customers of the status of their accounts ir response to an enquiry by accessing relevant file using an online terminal.

3. **Tour operators**

   Reservation offices accept telephone inquiries from travel agents regarding the availability of holidays in respect of client's inquiries. By means of terminals, the availability of the required holidays can be checked and looked immediately.

4. **Stock Exchanges**

   Terminals located in major stock exchanges through out the country and the offices of participating brokerage firms enables the speedy processing of share dealings.

5. **Stock Control**

Terminals located in warehouse provide the means for automatic reordering of stocks, updating of stock records, reservations, follow up of outstanding orders and the printing of picking list etc.

Advantages

1. **Integration of clerical staff with the computer**

   Clerical staff can have access to information which they require for the efficient performance of their jobs in dealing with customer inquiries and order processing.

2. **Elimination of tedious tasks**

   Routine clerical tasks are replaced by terminal operations providing a greater degree of job interest, operating efficiency and job satisfaction.

3. **Reduction in paper work**

   Volume of paper work generate by normal clerical system and batch processing system is relatively high On line systems reduce the volume of print out required for management report since the information can be displayed on terminal screen on demand.

4. **Improved accuracy**

   As terminal messages are checked accuracy before being transmitted to the computer by data validation programs, the quality of information in a system increases as the input errors are reduced. Hence, information is more reliable.

5. **File updating improved**

   Master files are more easily updated by terminal keyboard with regard to transaction data.

6. **Management information more readily available**

Management information becomes more readily available by direct acce facilities which enables managers to obtain a greater degree of control to t operations for which they are responsible.

7. **Improved customer services**

Improvements in the level of customers service can be expected in the systems concerned with appliances sales, banking systems and accou inquiries.

8. **Reduced data preparation cost**

Online system dispense with the need to convert human sensible data int machine sensible data there by eliminating punching and verifying operations This saves time and the cost associated with such operations.

1.8.4 Real Time O.S.

Real time system is defines as a data-processing system in which the time interval required to process & respond to inputs is so small that it control the environment. Real time processing is always on-line whereas on line system need not be real time. In this method, data relating to each single transaction, received via the telecommunication line, is treated as batch and the processed results are communicated back instaneously, enabling decisions to be made on the basis of the latest information.

The time taken by the system to respond to an input i.e. the time between pressing of last key by the operator curser. & display of required updated information is termed as response time. So in this method response time is very less as compare to the online processing.

The essential requirements of this method are :

1. Large main memory for software & OS requirements.
2. Large disk memory.
3. Stand by facilities to take care of such events as system failure.
4. Complex communication system.
5. Maintainence of audit trails as well as security of programs and data.

Examples of real time processing are :

1. Air traffic contral system.
2. Reservation systems used by hotels and car rental agencies
3. Process control systems as in nuclear reactor plants.
4. Systems that provide up-to-the minute information on stock prices.
5. Systems that provide immediate updating of customer accounts in savings banks.

Advantages

1. Response time is very less.
2. Better throughput
3. Large memory
4. 24 hours service provider
5. Provide information up-to-minute

Disadvantages

1. Very costly
2. Large main memory and secondary storage required.
3. Comlex communication systems.
4. Stand by facilities required to take care of such events as systems failure.

1.8.4.1 Difference between Batch O.S. & Real Time O.S.

BATCH PROCESSING	REAL TIME PROCESSING (Including online processing)
1. Data collected for defined period of time & processed in batches.	1. Random data input at random time.
2. Most economical.	2. Costly
3. Simplest processing method.	3. Complex processing method
4. Requires sorting prior to processing. Sorting required	4. No sorting required
5. It is measurement oriented.	5. It is action-oriented.
6. Information of master file is up-to-date only up to last updating run.	6. Information on master files updated as events occur.
7. Files are online only during a processing run.	7. Files are permanently online.
8. Magnetic tape as well as magnetic disk are used.	8. Only direct access device are used like magnetic disk.
9. This mode of processing is particularly used for updating payroll file, stock ledger, customer billing, invoicing etc.	9. This is used for railway reservation, airline reservation, banks, order entry, inventory control, production scheduly etc.
10. Paper work is more (All transaction recorded on source documents.)	10. Minimum paper work.
11. Output reports are printed in detail.	11. Most information is usually displayed on terminals.

1.8.5 Multi programming O.S.

The mismatch between the speed of Input/Output device and CPU leaves some resources of the computer system under-utilized. However, if the computer system is working in the multiprogramming mode, better utilization of the available equipment can be realized.

Multiprogramming refers to keeping several programs in different parts of the main memory at the same time as shown in figure 1.8 and executing them concurrently by the CPU.

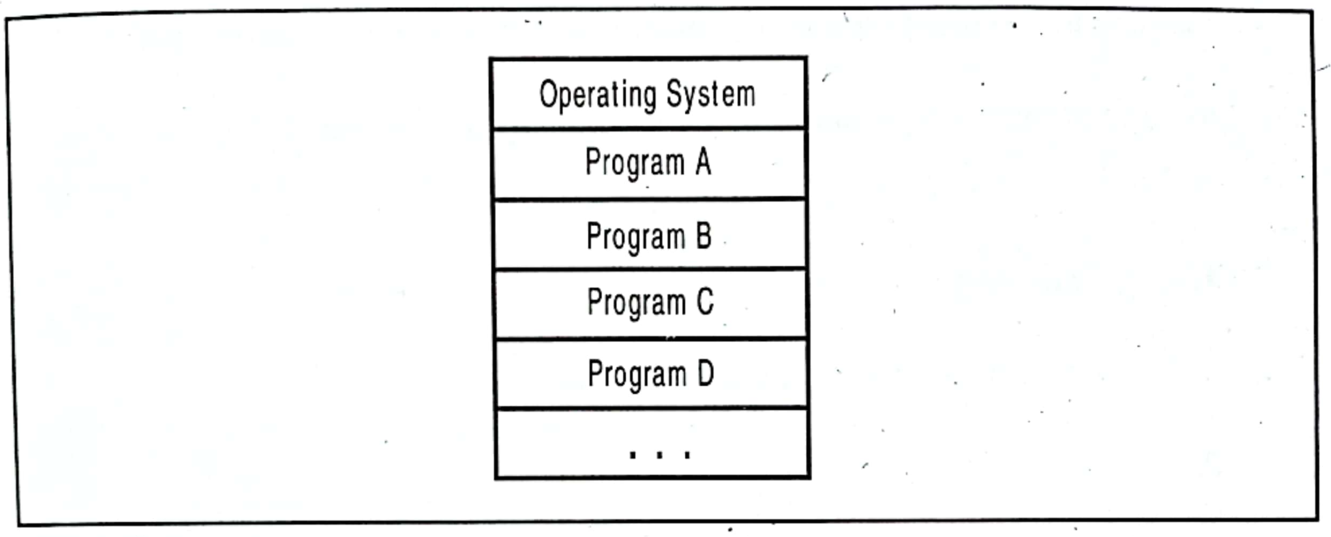


FIGURE 1.8

The CPU switches from one program to another almost instantaneously. Since the operating speed of CPU is much faster than that of I/O operations, the CPU can allocate time to several programs instead of remaining idle when one is busy with I/O operations. Hence, In multiprogramming system, when one program is waiting for I/O transfer, there is another program ready to use the CPU. A simple example of multiprogramming is given in the figure 1.9

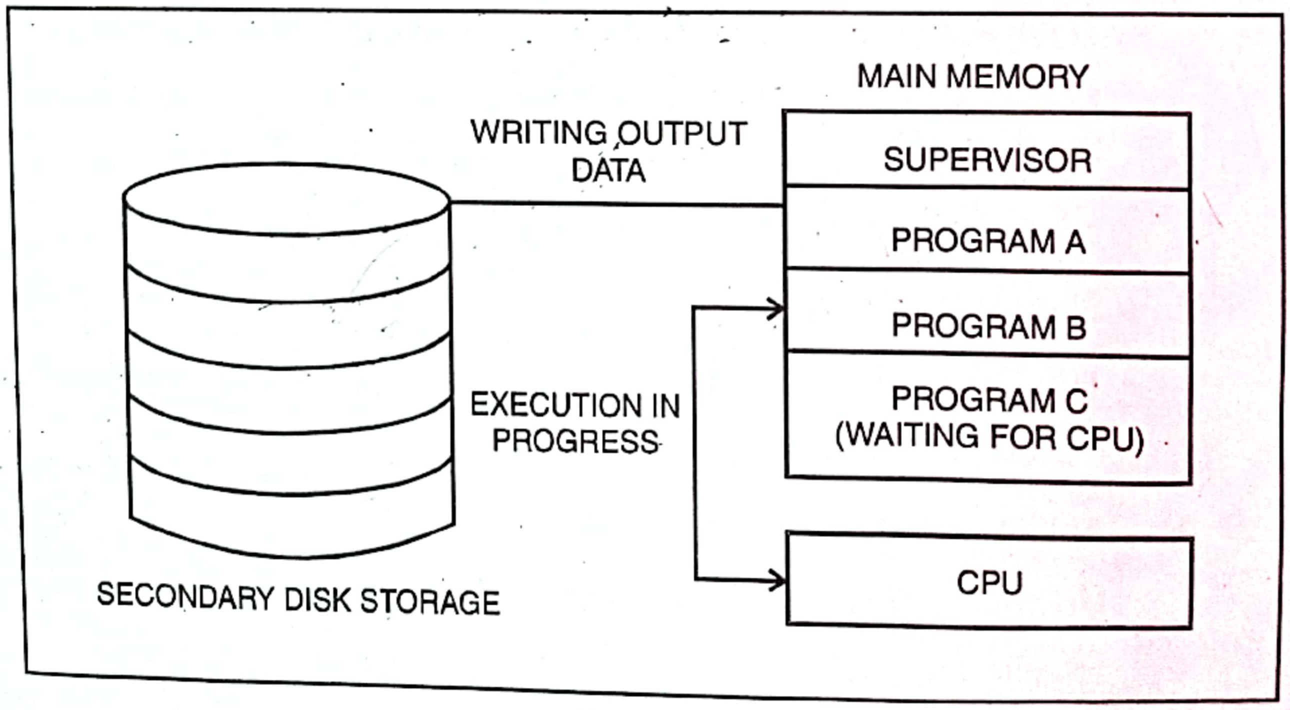


FIGURE 1.9

At the particular time instance shown in the figure, program A is not utilizing t CPU since it is busy in writing output data on to the disk. The CPU is being used execute program B which is also present in the main memory. Another program residing in the main memory, is waiting for the CPU to become free.

In case of multiprogramming the various programs stored in the main memory, ca be in one of the following three stages :

1. **Running**

   CPU is being used by the program.

2. **Ready**

   Waiting for CPU to be assigned to it.

3. **Blocked**

   Performing I/O opeation.

Advantages of Multiprogramming :

1. **Increased Throughput**

   Throughput is a measure of the total amount of processing that a computer system can complete over a fixed period of time. Total through put is significantly increased in multiprogramming because the CPU is not waiting for I/O for the program it is executing .

2. **Shorter Response Time**

   Turn around time for short jobs can be greatly improved under multiprogramming.

3. **Ability to Assign priorities of Jobs**

   Most multiprogramming systems have schemes for setting priorities for rotating programs.

4. **Improved Primary : Storage Allocation**

The greater the number of programs that primary storage can hold, the greater the probability that the CPU will be able to execute at least one program while waiting for I/O for the others.

Requirements or Disadvantages of Multiprogramming Systems

1. **Large Main Memory**

   Large main memory is required to accomodate many user programs along with operating system.

2. **Memory Protection**

   Computers designed for multiprogramming must provide some type of memory protection mechanism to prevent a program in one memory partition, from changing information or instruction of a program in another memory partition.

3. **Proper Job Mix**

   The main memory should contain some CPU-bound programs and some I/o-bound programs in its varioius partitions so that atleast one of the program which does not need I/o is always available to the CPU for processing.

4. **Program Status Preservation**

   In multiprogramming, a portion of one program is executed, then a segment of another, and so on. Before a program is suspended and the control is passed to another program, the values of all CPU registers should be stored in the memory area of that program and then restored when the control is ultimately returned to the first program. This is known as program status preservation.

5. **CPU Scheduling**

   There will be situations which two or more programs will be in the ready state, waiting for CPU to be allocated for execution. In such a case, the operating system ust decide to which program should CPU be allocated.

1.8.5.1 Difference between serial execution and Multiprogramm Execution

Difference between serial (or sequential) execution and Multiprogramming execu is shown in figure 1.10

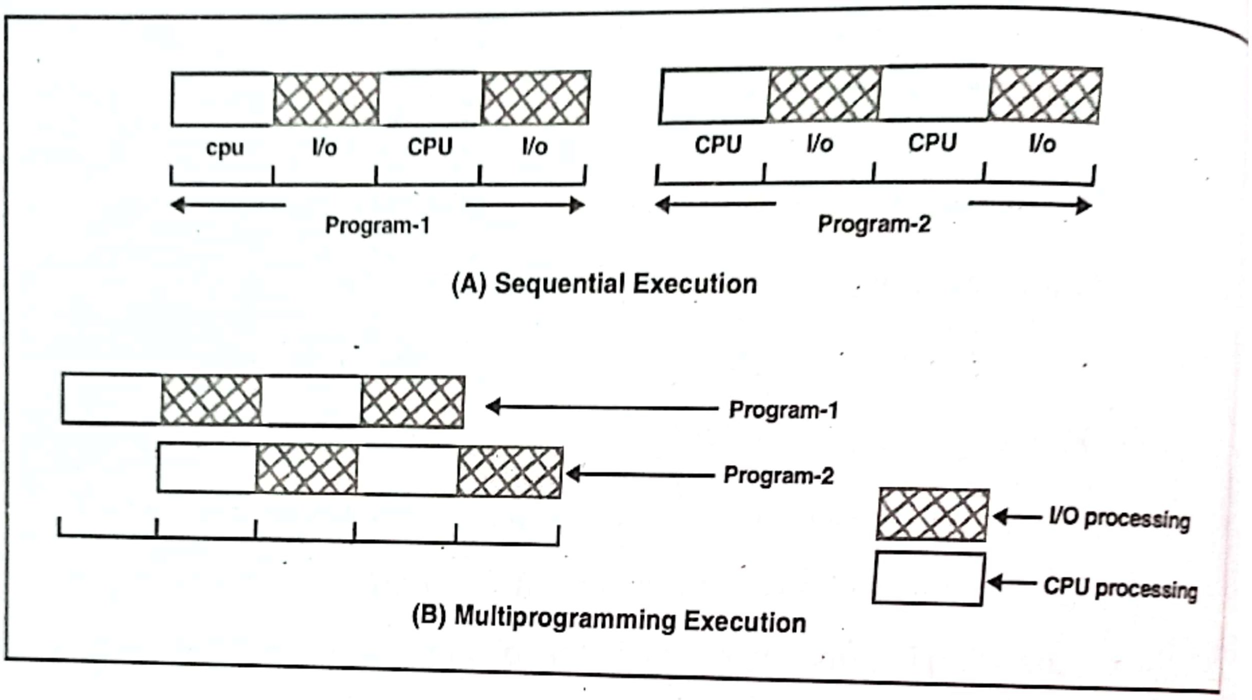


FIGURE 1.10

figure 1.10 (a) shows that at any time either the processor (CPU) is busy or I/O devices are active. Both are not active simultaneously. Figure 1.10(b) shows the possible concurrent execution of two programs. In this we see that when CPU sits idle for Program-1 during I/O activity, it simply switches to CPU intensive phase of the Program-2 or vice-versa. The number of programs actively competing for resources of a multiprogramming computer system is often called as degree of multiprogramming. So higher the degree of multiprogramming, higher will be the resouce utilization.

1.8.6 Timesharing Operating System

This is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Each user is actually away from the computer and other uses. Each user has it own set of programs. Under this scheme specific time slice are allocated to different user in rotation. Time-sharing system, allocates a very short period of CPU time one-by-one to each user process, beginning from the first user process and proceeding through the last one, and then again beginning from the first one. This short period of time during which a user process gets the attention of the CPU is known as ***a time slice, time slot, or quantum,*** *and* is typically of the order of 10 to 100 millisecond. It appears to the user that the computer is working for him exclusively, though the fact is that computer serves him periodically. Thus time sharing may be viewed as a multi-use multiprogramming technique. The time sharing process is shown in figure 1.11

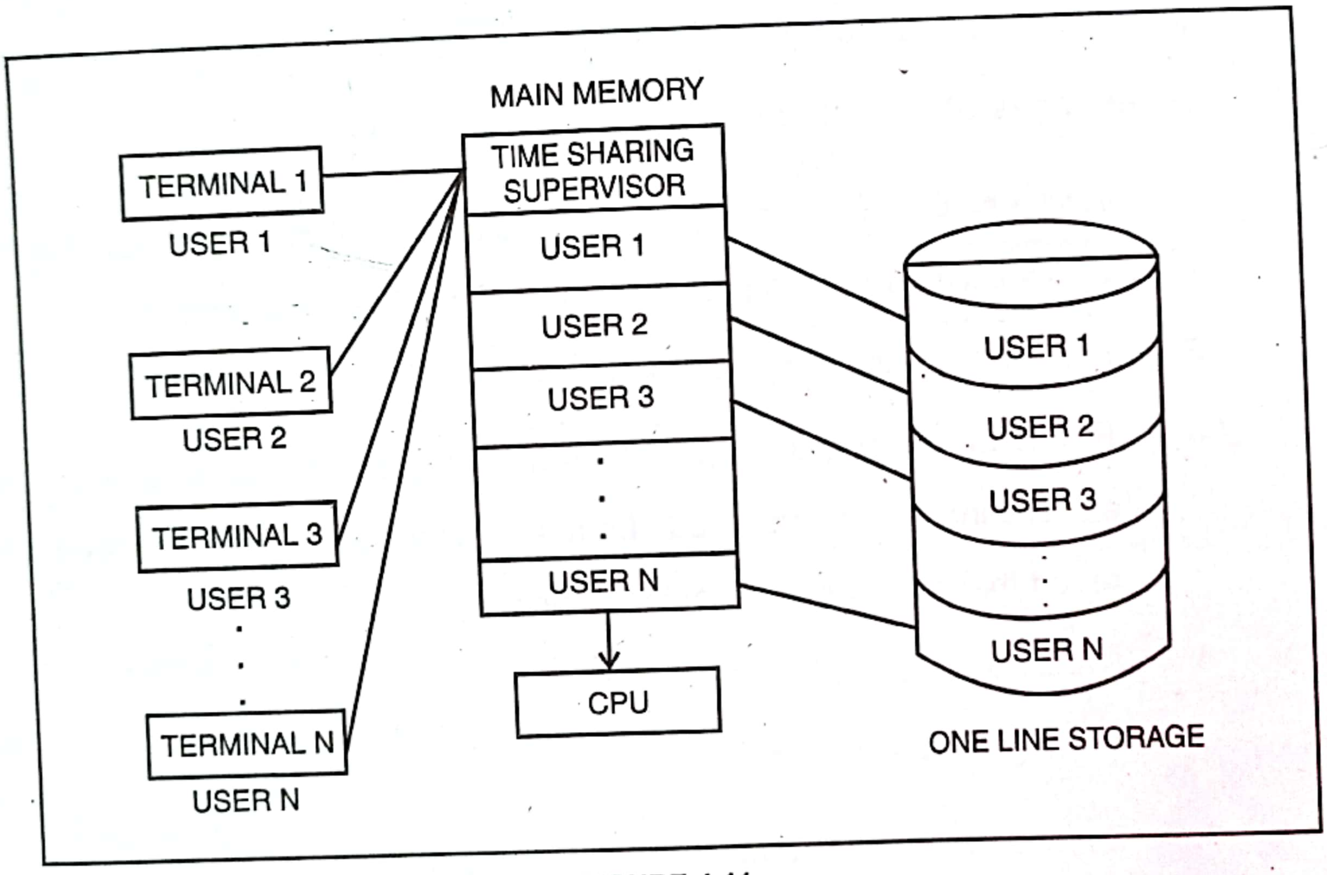


FIGURE 1.11

In case of time-sharing system the various programs can be in one of the following three stages :

1. **Running :** CPU is being used by the program.
2. **Ready :** Waiting for CPU to be assigned to it.
3. **Blocked :** Performing I/O opeation.

Requirements of Timesharing System

Requirements of Timesharing system are exactly same as we have discussed multiprogramming system i.e.

1. Large Main Memory
2. Memory Protection
3. Proper Job Mix
4. Program Status Preservation
5. CPU Scheduling

Advantages of Timesharing

1. **Reduces CPU idle time :** Time sharing significantly increases CPU utilization by switching from one program to another in rapid succession.
2. Provides advantages of quick response.
3. **Reduces the output of paper :** If a manager can retrieve at any time the specific information he needs from an online file, he does not need a bulky report that contains much of the file information.
4. Avoids duplication of software.

Disadvantages of Timesharing

1. Question of Security and integrity of user programs and data.
2. Problem of data communication.
3. Problem of reliability
4. Question of overhead involved : The timesharing system with its control functions such as switching from user to user and swapping programs in and out takes up an appreciable amount of CPU time. This is termed overhead and must be minimized.

1.8.7 *Multiprocessing*

This technique consists of two or more CPUs connected to common peripherals. Instruction from different programs may either be processed by different CPUs or one, or more, processors may execute instructions from the same program simultaneously. This technique can serve a purpose only in large computer Installations, where same jobs are too large for one CPU, or where a large number of problems need to be solved simultaneously.

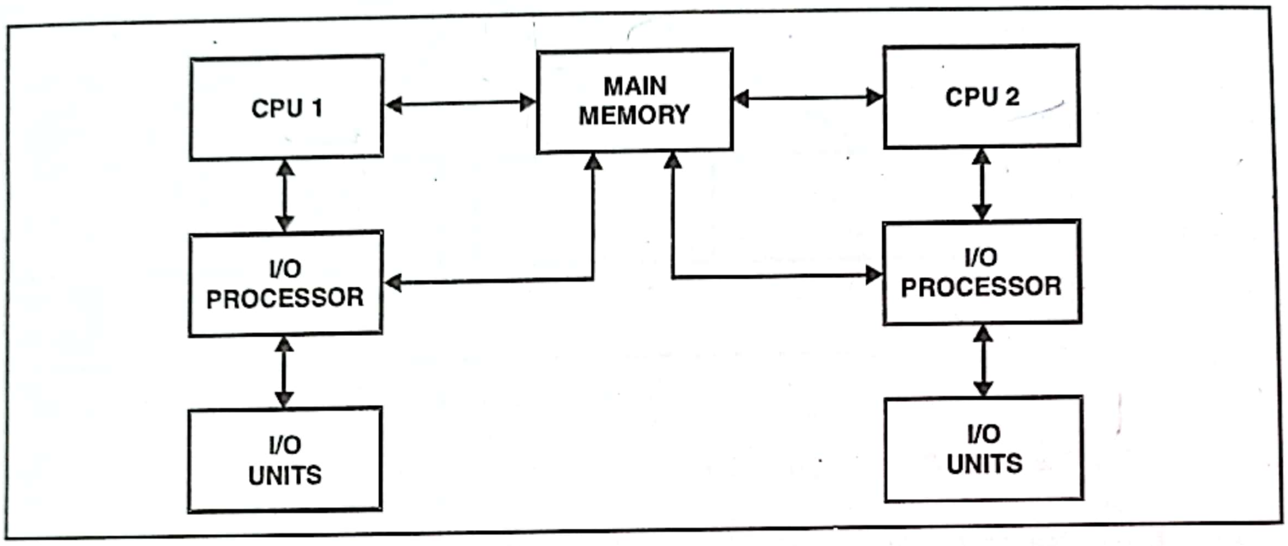


FIGURE 1.12

Generally the multiple processors are used in two ways :

1. **Symmetric Multiprocessor :** In Symmetric multiprocessor, each processor runs an identical copy of the operating system and these copies communicate with one another when needed. Figure 1.13 shows fthe structure of Symmetric Multiprocessor.

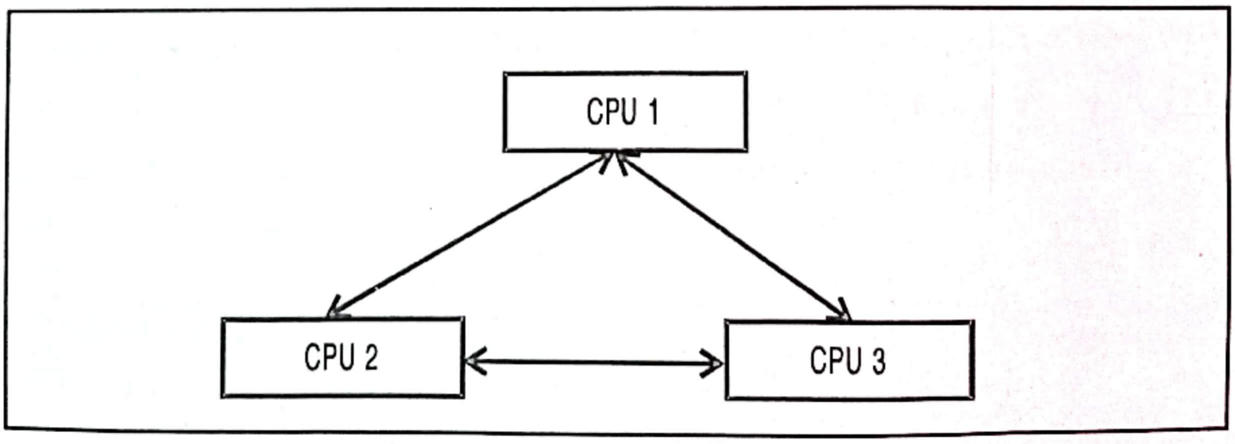


FIGURE 1.13

2. **Asymmetric Multiprocessor :** In Asymmetric multiprocessor, each proces is assigned a special task. In this scheme there is a master processor and remaining processors known as slave processors are controlled by the mast Such scheme is called as master-slave relationship as shown in figure 1.1 Hence, it is the master who allocates the work to the slave processors

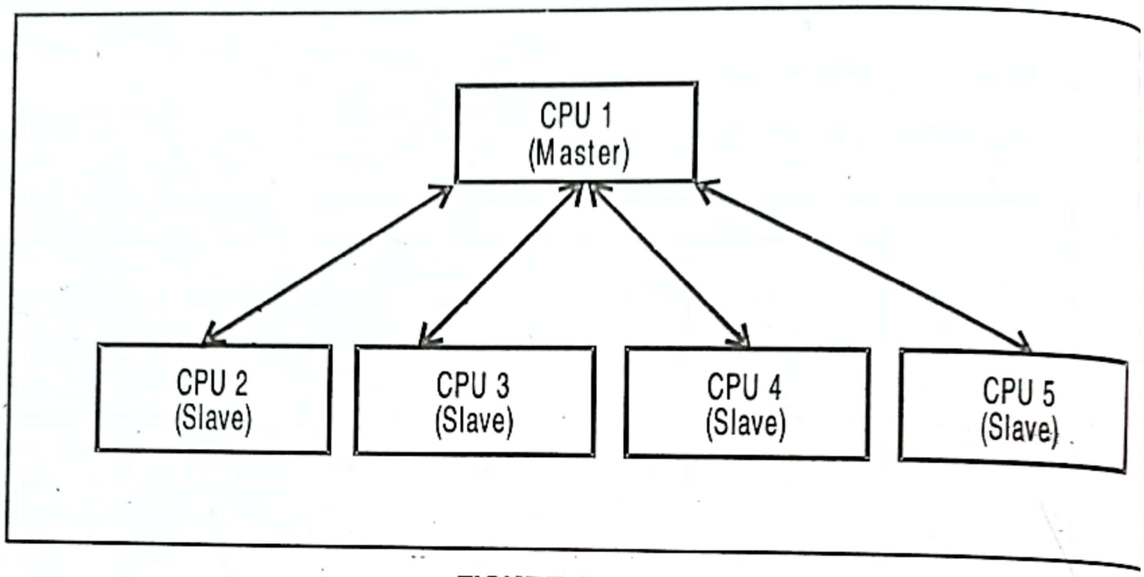


FIGURE 1.14

Advantages of Multi processing

1. **Better throughput :** It improves the performance of computer systems b allowing parallel processing of segments of programs.
2. **Better Reliability :** It provides a built-in backup. If one of the CPUs break down, the other CPUs automatically takes over the complete workload unti repairs are made. Hence, Multiprocessor system have **better reliability**.
3. **Better utilization of resouce :** In additions to the CPUs, it also facilitates more efficient utilization of all the other devices of the computer system.
4. **Saving Cost :** Such system shares the memory, buses, clock etc. so it reduces the cost of the system.

Disadvantages of Multi processing

1. A large main memory is required.
2. A very sophisticated operating system is required to schedule, balance and co-ordinate the input, output and processing activities of multiple CPUs.
3. Such system are very expensive.

1.8.7.1 Distinguish between Multiprogramming and Multiprocessing

Multiprogramming is the interleaved execution of two or more processes by a single-CPU computer system. On the other hand, multiprocessing is the simultaneous execution of two or more processes by a computer system having more than one CPU.

1.8.8 Multitasking

In multiuser systems, multiasking is the same as multiprogramming. In a single-user system, it is not necessary that the system work only on one job at a time. In fact, a user of a single-user system often has multiple tasks concurrently processed by the system.

In multitasking several programs reside in RAM together and are executed simultaneously. In multitasking, the computer might be printing out a data file while at the same time the user is running a word processing program to enter a document.

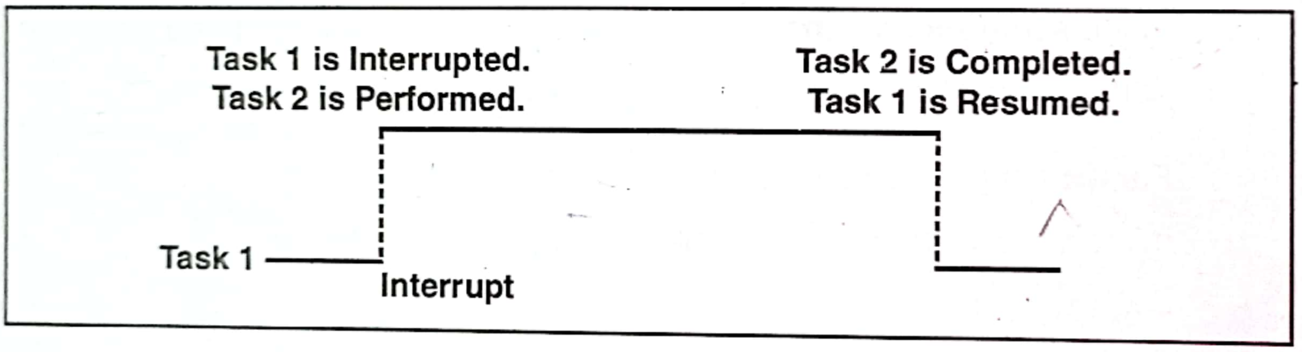


FIGURE 1.15

The basic idea of multitasking is to allow devices to perform much of their own processing, independent of the CPU. Whenever a device requires the attention of the CPU, it issues a signal to the CPU. This signal is called an interrupt. In response to an interrupt, the CPU puts aside its current activity and tends to the device. Then the CPU returns to whatever it was doing when the interrupt occurred. In this way, the CPU can carry out several simultaneous, or concurrent tasks. Multitasking systems divide the screen into a series of windows. A window environment usually includes provisions for moving data between windows, allowing programs to exchange information.

The user can run a separate program in each window. This can be extremely use
The programs in the various windows can keep a number of system devices work
at the same time. For example, the program in Window 1 could be printing a docum
and the program in Window 2 could be transmitting a document to another compu
via modum. Once the programs in Windows 1 is started, it can run without a
attention from the user. The user can than concentrate on the sale projection
Window 2.

1.8.9 Distributed Operating System

A distributed system uses multiple central processors to serve multiple real-tim
applications and/or multiple classes of users. Each Central Processing Unit (CP
specializes in performing particular functions or serving a particular class of user
Data processing jobs are distributed among the processors accordingly to whic
one can perform each job most efficiently. The Central Processors may all be locate
at the same site, but more typically are geographically scattered and connected b
data communication lines, forming a distributed network.

Far distant places may be linked through satellite transmission channels or groun
microwave systems and within same city through telephone lines or special coaxia
cables.

Thus, In distributed operating systems, each processor has its own memory and executed its own jobs and shared jobs. The processors communicate with one another through various communication media. Such systems are also known as **loosely-coupled systems**.

A typical application is in banks where all the branches have intelligent terminals (usually micro-computers) linked to a big computer at the Head Office. Data from the branches is sent to the master where it is processed.

The design of distributed operating system is based on two models :

(i) Client-server model

(ii) Peer-to-peer model

(i) Client Server Model

In this model, a specific computer is known as server which is specially dedicated to provide various services to other computers (called Clients). In simple words, server can be defined as a provider of services and client can be as a requester to services. Client makes a request for any kind of information from the server and server in turn respond to the client request as shown in figure 1.16.

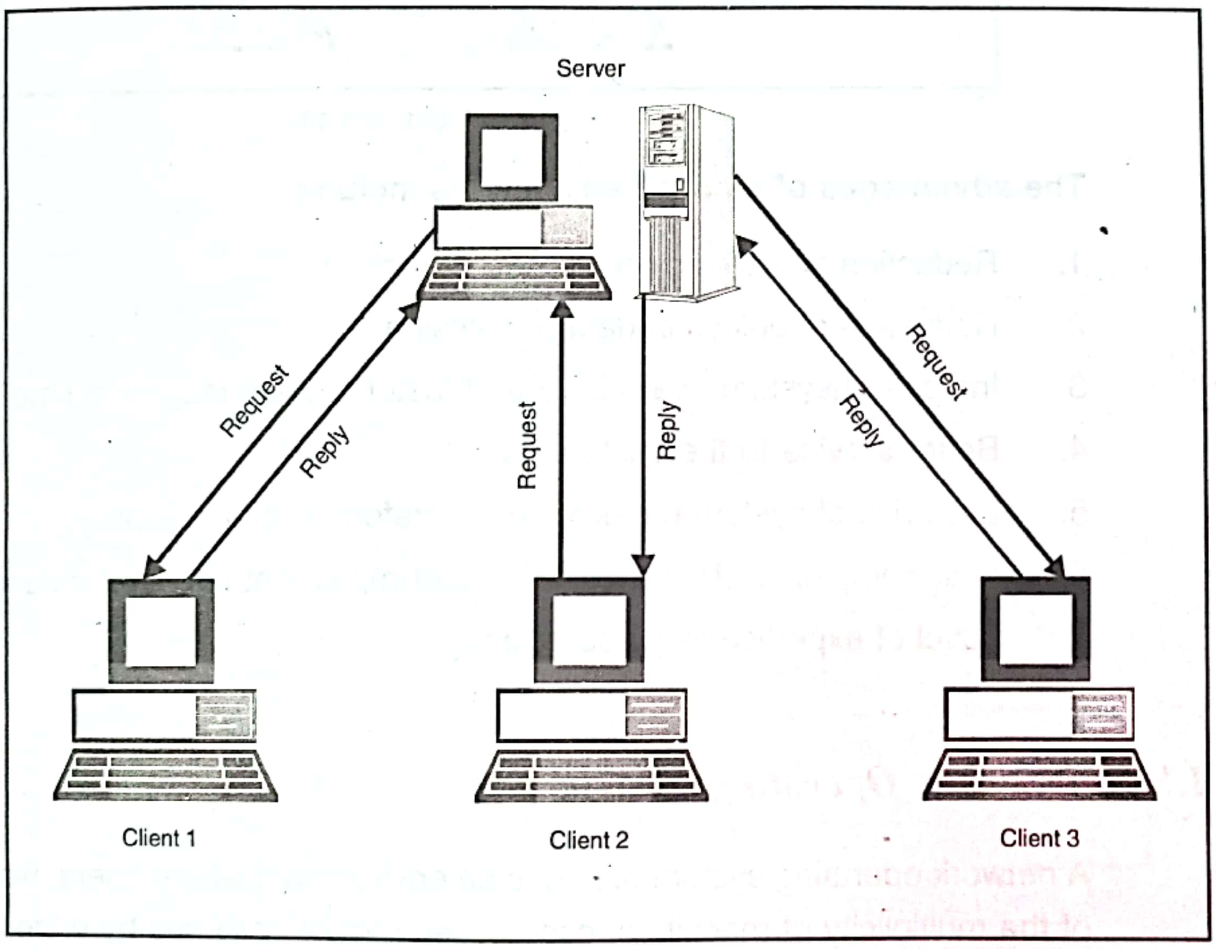


FIGURE 1.16 (CLIENT SERVER MODEL)

(ii) Peer to Peer Model

In peer to peer model, there are no dedicated servers. All computers are equal and, therefore, are termed as peer. Normally, each of these machines functions both as a client and a server. This arrangement is suitable for environments with a limited number of users (usually ten or less). In this models users need to freely access data and programs that reside on other computers across the network.

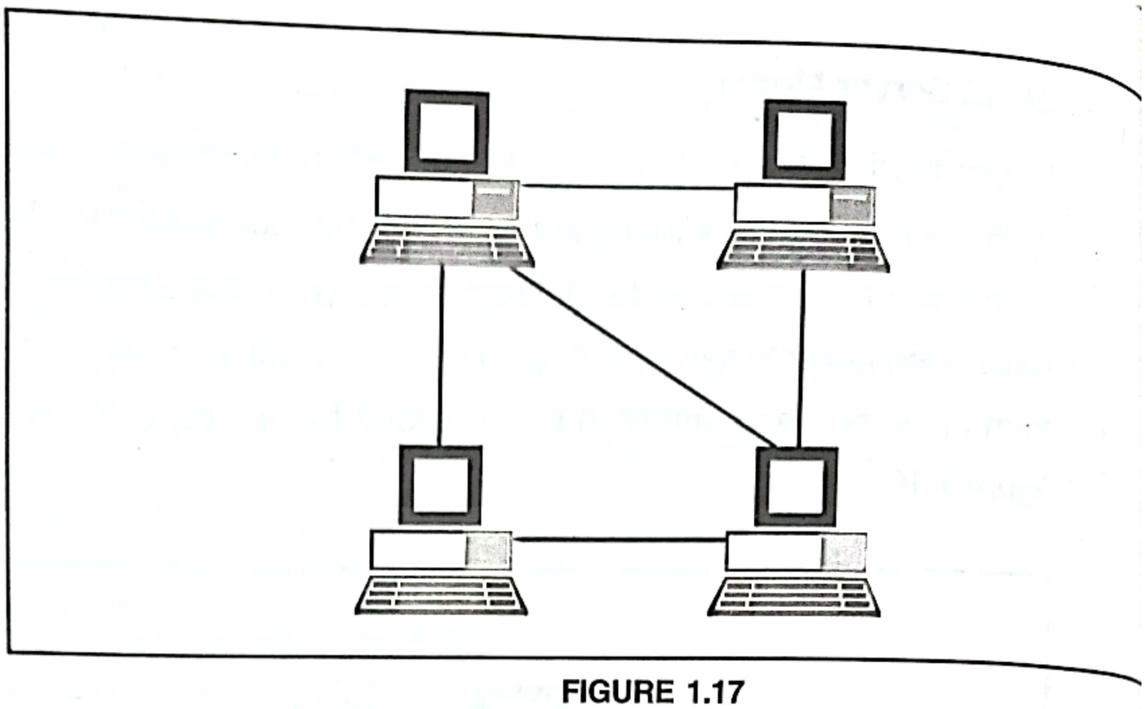


FIGURE 1.17

The advantages of distributed systems include :

1. Reduction of the load on the host comptuer.
2. Reduction of delays in data processing.
3. Increased system availiability and faster system response time.
4. Better service to the customers.
5. Less risk of system breakdown. Therefore, better reliability.
6. Less complexity of system design and implementaiton due to decentralisat
7. Level of expertise required is less.

1.8.10 Network Operating Systems

A network operating system provides an environment where users, who are aw of the multiplicity of machines, can access remote resoures by either logging i the appropriate remote machine, or transferring data from the remote machine their own machines.

A network operating system can be called as a special type of distributed operati system which allows to connect many computers in a single channel or circuit w them. Thus in case of a network operating system two or more operating syster are in use at a time. One is the operating system on one or more machines a other is the operating system managing the connection of these systems. Th network operating system is that operating system which manages the working of computer network.

1.8.11 Parallel Processing System

In parallel processing systems, multiple CPUs work in parallel to improve performance through parallel implementation of various operations such as loading data, building indexes and evaluating queries. Parallel processing divides a large task into many smaller tasks and executes the smaller tasks concurrently on several CPUs. As a result, the larger tasks completes more quickly. Parallel processing systems improve processing and input/output (I/O) speeds by using multiple CPUs and disks working in parallel.

There are three commonly used architectural models for parallel machines :

1. Shared-memory multiple CPU
2. Shared-disk multiple CPU
3. Shared-nothing multiple CPU

1. **Shared-memory Multiple CPU**

   In a shared-memory system, a computer has several simultaneously active CPUs that are attached to an interconnection network and can share a single (or global) main memory and a common array of disk storage as shown in fig. 1.17

   The shared-memory architecture of parallel system is closest to the traditional single-CPU processor of centralised systems, but much faster in performance as compared to the single-CPU of the same power.

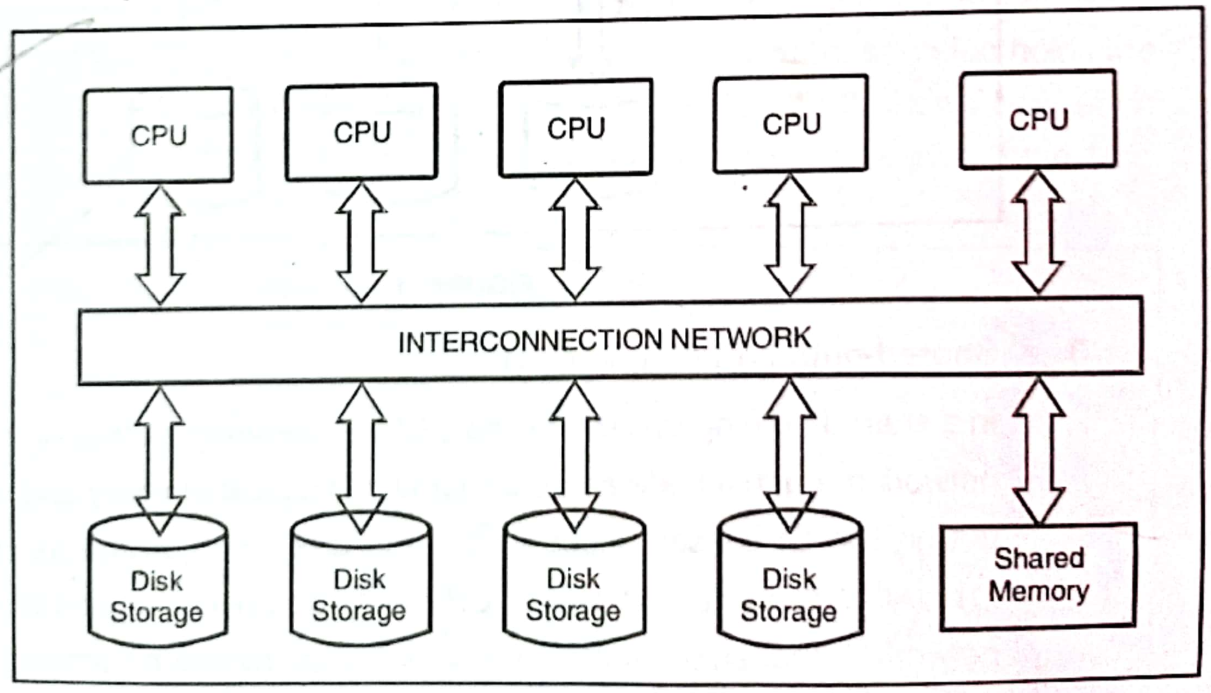


FIGURE 1.17

2. Shared-disk Multiple CPU

In a shared disk system, multiple CPUs are attached to an interconnec network and each CPU has its own memory but all of them have acces the same disk storage or, more commonly, to a shared array of disks shown in fig. 1.18

Since memory is not shared among CPUs, each node has its own copy of operating system. It is possible that, with the same data accessible to all no two or more nodes may want to read or write the same data at the same tim

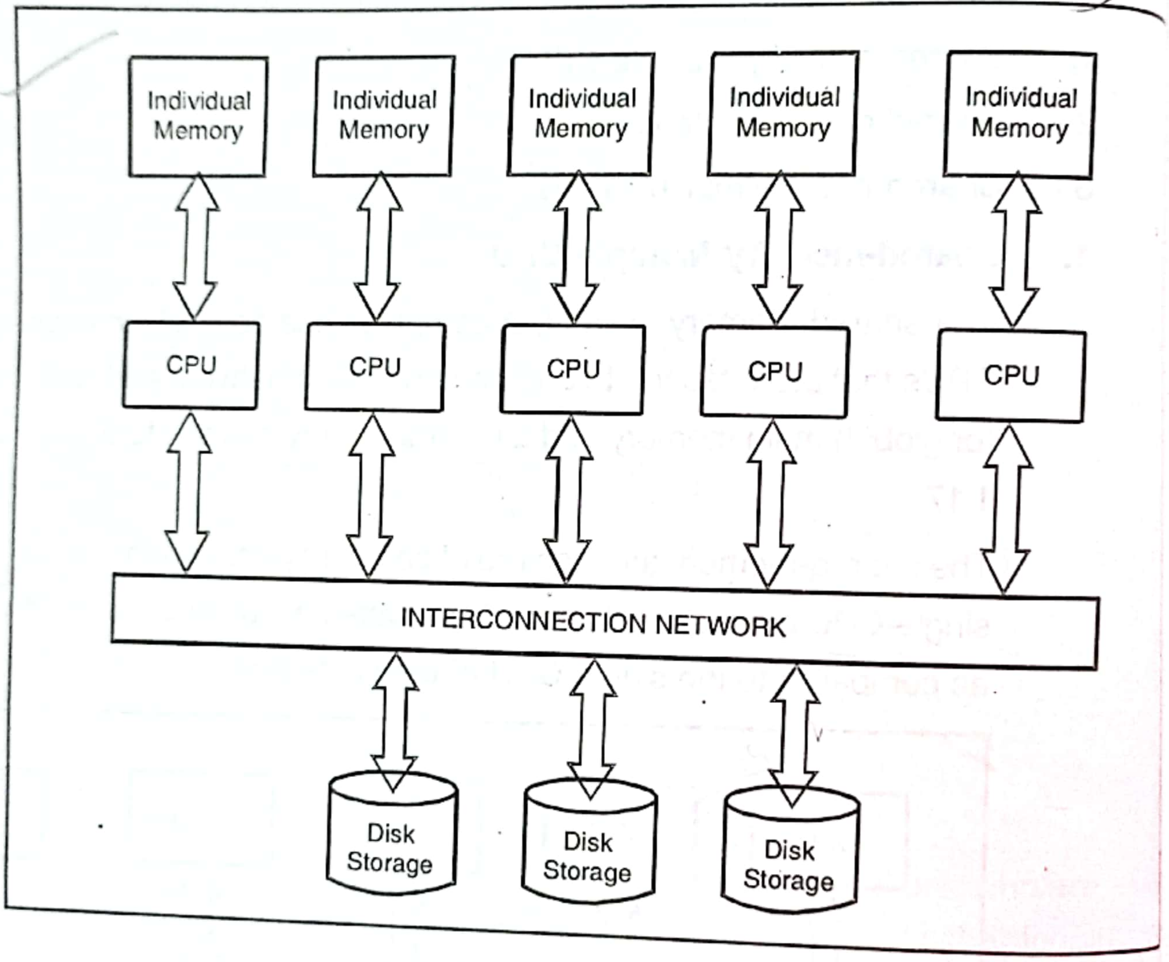


FIGURE 1.18

3. Shared-nothing Multiple CPU

In a shared-nothing system, multiple CPUs are attached to an interconnection network through a node and each CPU has a local memory and disk storage, but no two CPUs can access the same disk storage area as shown in fig. 1.19. All communication between CPUs is through a high-speed interconnection network. Thus, shared-nothing environments involve no sharing of memory or disk resources.

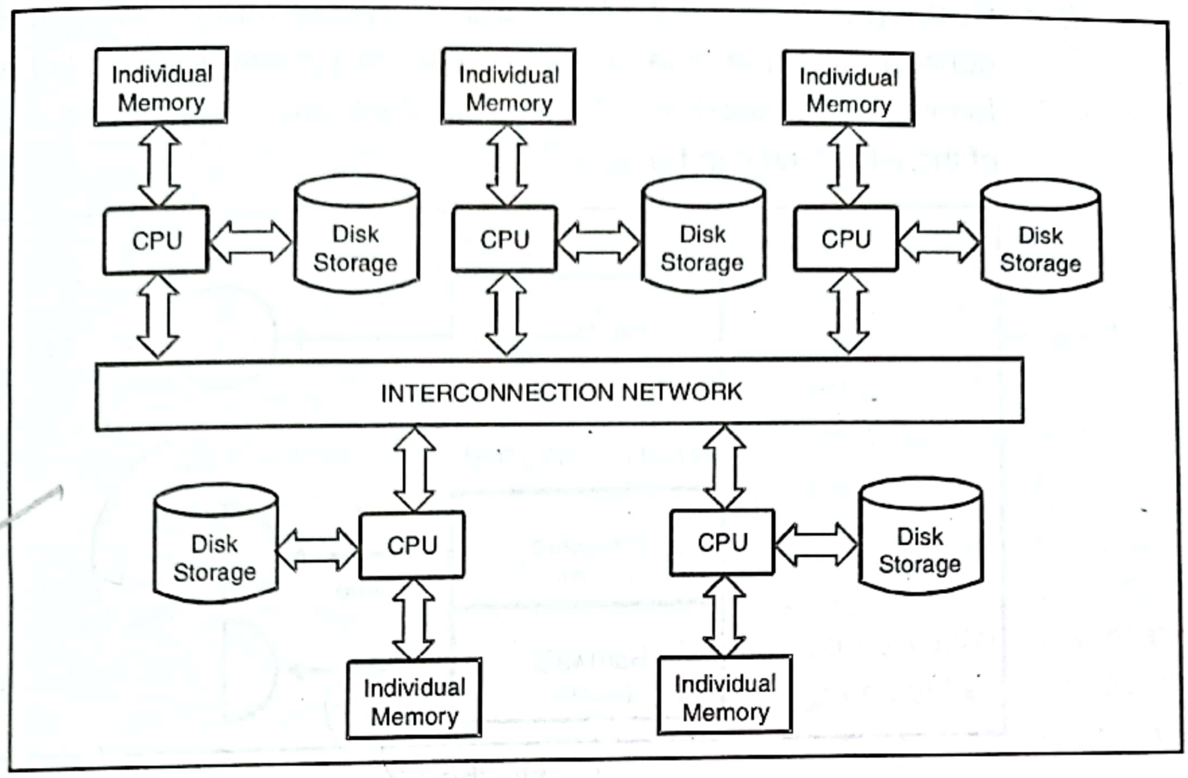


FIGURE 1.19

Advantages :

Main advantages are :

1. **Speedup**

   Speedup is the extent to which more hardware can perform the same task in less time than the original system. With added hardware, speedup holds the task constant and measures time savings. Figure 1.20 shows how each parallel hardware system performs half of the original task in half the time required to perform it on a single system.

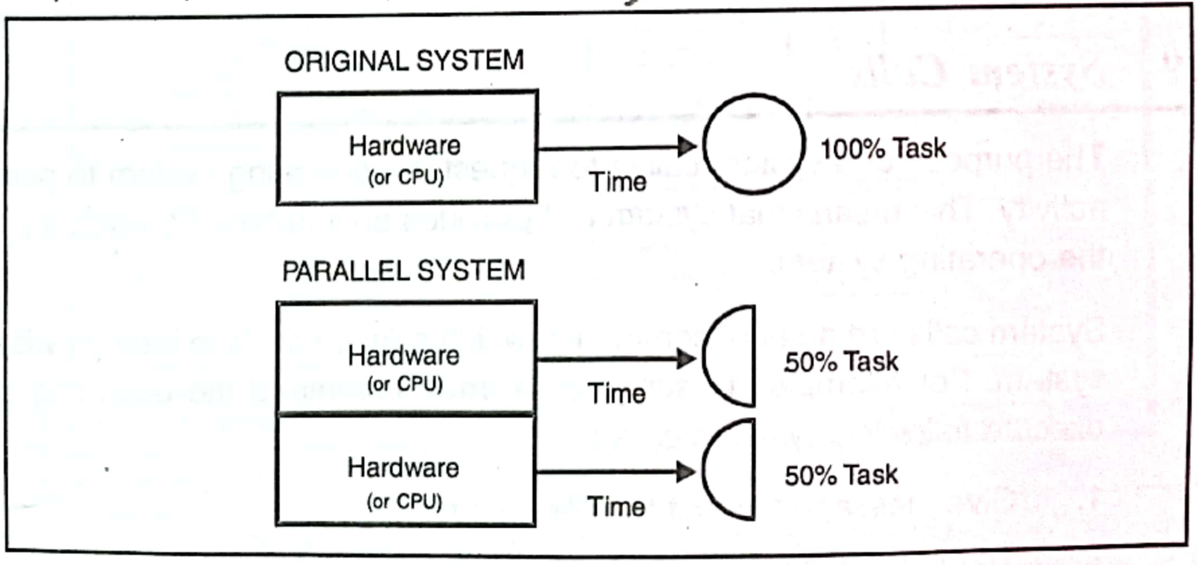


FIGURE 1.20

2. **Scaleup :** *Scaleup* is the factor that expresses how much more work ca done in the same time period by a larger system. With added hardwa formula for scaleup holds the time constant, and measures the increased of the job which can be done.

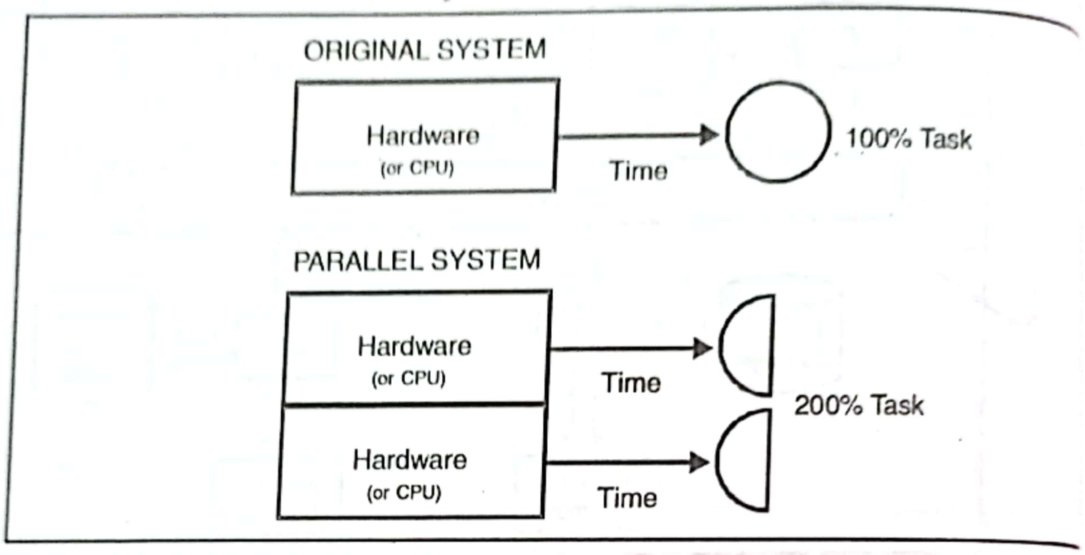


FIGURE 1.21

It transaction volumes grow and you have good scale-up, you can ke response time constant by adding hardware resources such as CPUs.

3. **High Avaiability :** Nodes are isolated from each other, so a failure at o node does not bring the entire system down. This means data is much mo available than it would be with a single node upon node failure. This al amounts to significantly higher database availability.
4. **Greater Flexibility :** A Parallel processing environment is extremely flexib
5. **More Users :** Parallel processing technology can make it possible to overco memory limits, enabling a single system to serve thousands of users.

1.9 System Calls

The purpose of a system call is to request the operating system to perform som activity. This means that system call provides an interface between a process an the operating system.

System calls are a set of commands which a user needs to interact with operatin system. For example, if user gives a **copy** command the operating system wi execute following systems calls :

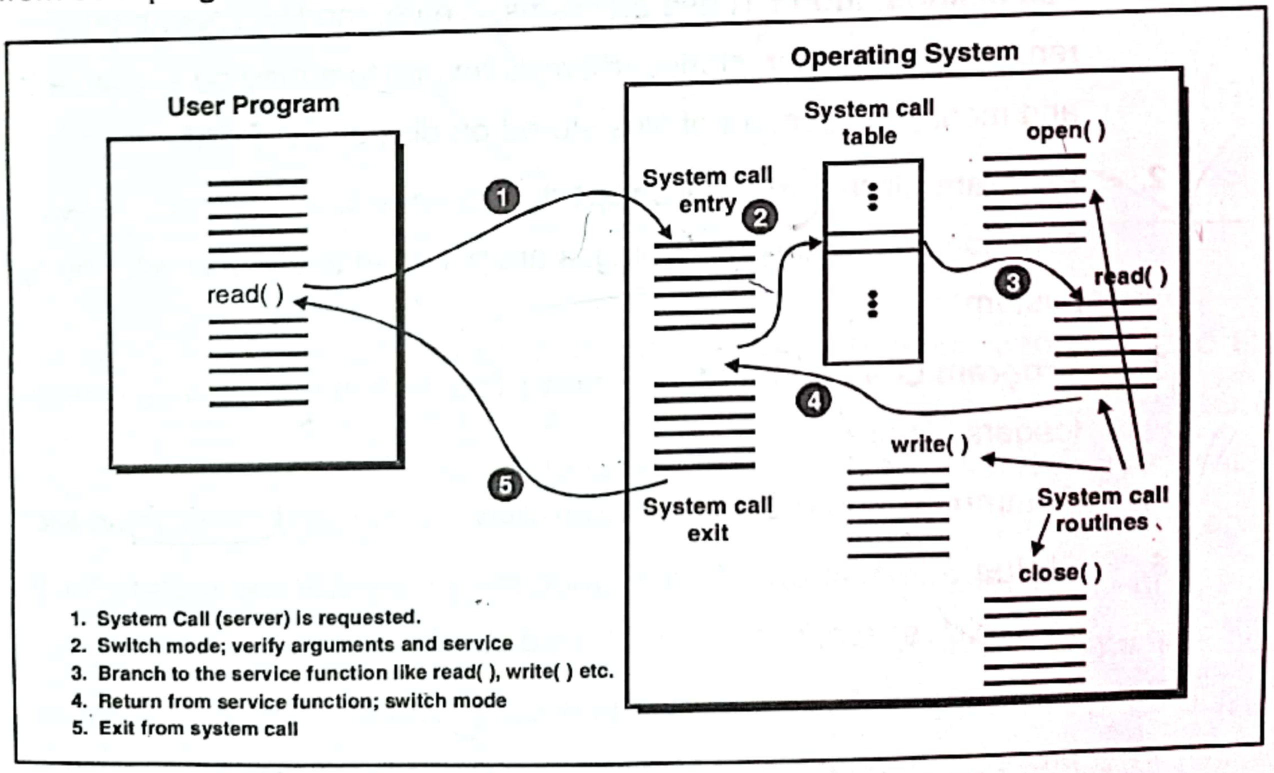
1. Give message to input two file names.
2. To open source and destination files.

3. Give error, if any
4. Read source file
5. Write in destination file
6. Display reading or writing error, if any
7. Close both files after execution of job

System calls are generally available as assembly - language instructions. Certain systems allow system calls to be made directly from a high - level language program. So to a high - level language programmer, invoking a system call is very similar in nature to calling any other procedure or function. The essential difference, however is that **in case of a convential subroutine, the object code is part of the calling program, while the system call code is within the operating system.**

An Operating System will only have a limited number of available system calls and in general, these will be suppliemented by standard subroutine libraries. Many of these standard subroutines, which provide additional higher level facilities for the programmer, will themselves use system calls. These higher level subroutines are generally organised into **Application Programming Interfaces** or APIs. An API would provide functions for all aspects of system activity, such as memory, file and process management.

Fig 1.22 shows how a system call serves in achieving protection of operating system from user program.



1. System Call (server) is requested.
2. Switch mode; verify arguments and service
3. Branch to the service function like read( ), write( ) etc.
4. Return from service function; switch mode
5. Exit from system call

FIGURE 1.22 ***[Protection Provided by System Cell]***

In unix, there are different system calls for performing different kinds of tasks.

- File manipulation system calls like open( ), close( ), read( ), write( ) etc.
- Process control system calls like abort( ), exec( ), wait( ), fork( ) etc.
- Device manipulation system calls like select( ), open( ), close( ) etc.
- Information maintenance system calls like time( ), acct( ) etc.
- Communications system calls like send( ), recv( ), accept( ) etc.

1.10 System Programs

System programs provide basic functioning to users so that they do not need write their own environment for program development (editors, compliers) an program execution (shells). In some sense, they are bundles of useful system cal

A most important systems program for an operating system is the comman **interpreter,** the main function of which is to get and execute the next user specifie command.

System programs can be divided into several categories.

1. **File Manipulation :** These programs, create, modify, copy, print, delete, list rename files and directories. Several text editors may be available to create and modify the content of files stored on disk or CD-ROM.
2. **Programming-language Support :** Compilers, interpreters, assemblers for common programming languages are provided to the user with the operating system.
3. **Program Loading and Execution :** The system may provide various type of loaders, linkers.
4. **Communications :** Communication allow users to send messages to each others.
5. **Status Information :** Some programs simply ask the system for the date, time, disk space, number of users etc.
6. **Application Programs :** Such programs include web browsers, word processors, spreadsheets, database systems, games etc.

1.11 *Operating System Structure*

Before we discuss the structure of UNIX, DOS and Window NT Operating systems, let us understand what does actually an operating system comprise of? It contains mainly a Kernel, command processor or shell and graphical user interface to make a user's life simple.

(a) **The Kernel :** The Kernel provides the **file systems, CPU Scheduling, Memory Management** and other operating system functions through system calls. Programs such as shell (sh) and editors (vi) interact with the Kernel by invoking a well defined set of system calls. The system calls instruct the Kernel to do various operations for the calling programs and exchange data between Kernel and program.

1. resource allocation
2. disk management
3. Process management
4. Memory management
5. I/O device

editors

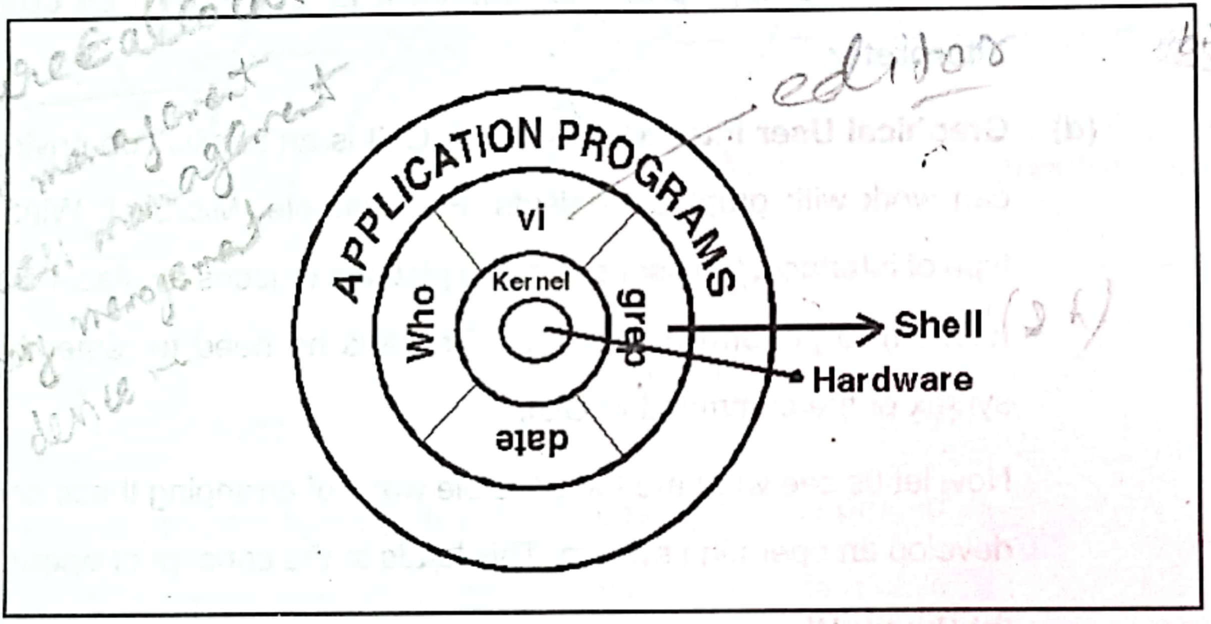


FIGURE 1.23

(b) **The Shell :** The Shell provides an easy inteface between user and the computer. The shell is a program that interprets the commands you type. If the command is valid, the shelll directs the Kernel to carry out your request. If the command is invalid, you will see and error message, and then the shell program will give you another chance to type a valid command. In unix, at a particular point in time, there may be several shells running in memory but only one Kernel. This is because, at any instance Unix is capable of executing only one program as the other programs wait for their turn.

(c) **Command Processor :** The Command Processor is an interface betwee the user and the operating systems. Command Processor provides a standar set of commands that gives users access to file-management, devic management, configuration and miscellaneous functions, such as maintainin and verifying the time and date.

Whenever a command prompt is displayed the command processor waits fo a command from the user. After user enters a command, it (comman processor) makes sure that command is valid and executes it. If command is not valid, it gives an error message. For operating systems with GUI (Graphical User Interface), the command processor interprets mouse operations and executes the appropriate command. It is also known as **command line interpreter.**

(d) **Graphical User Interface (GUI) :** A GUI is an application environment that can work with graphical objects. For example, Microsoft Windows. In this type of interface, the user select the pictures or icons available on the desktop (screen) to perform his/her task. There is no need to remember the exact syntax of the command in GUI.

Now let us see what are the possible ways of arranging these components to develop an operating system. This leads to the concept of **operating system architecture.**

1.11.1 Monolithic Architecture for Operating System

Monolithic Architecture is the oldest architecture used for developing an operating system. In this architecture, operating systems is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. Therefore, in this technique, each procedure has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if requires.

In this approach, to construct the actual object program of the operating system, one first compiles all the individual procedures, or files containing the procedures, and then binds them all together into a single object file using the system linker.

Fig 1.24 shows the monolithic architecture of an operating system.

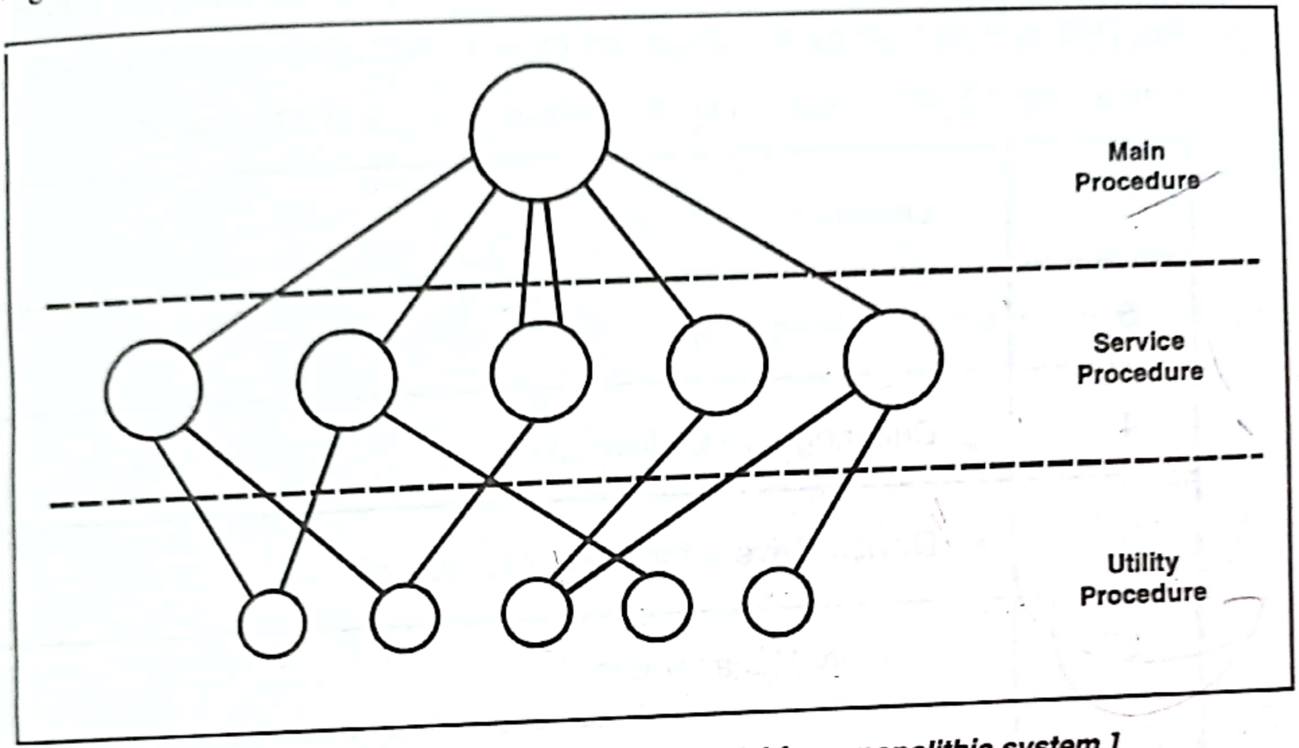


FIGURE 1.24 ***[A simple structuring model for a monolithic system]***

This organization suggests a basic structure for the operating system :

1. A main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

In this model, for each system call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs as shown in fig.

Disadvantages

1. No information hiding as every procedure is visible to every other procedure.
2. Monolithic kernel is not very portable.

1.11.2 Layered Architecture for OS

The operating system architecture based on layered approach consists of numbe of layer (levels), each built on top of lower layers. The bottom layer is the hardware; the highest layer is the user interface. The first system constructed in this way was the THE system built by E.W.Dijkestra (1968) and his students. The THE system was a simple batch operating system which had 32k of 27 bit words.

6	Operator
5	User Programs
4	Buffering for I/O devices
3	Device Drivers for operator's console
2	Memory Management
1	CPU Scheduling + Multiprogramming
0	Hardware

As shown in figure, layer 0 dealt with hardware; the higher layer 1 handled allocation of jobs to processor. The next layer implemented memory management. The memory management scheme was virtual memory. Level 3 contained the device driver for the operator's console. By placing it, as well as I/O buffering at level 4, above memory management, the device buffers could be placed in virtual memory. The I/O buffering was also above the operator's console, so that I/O error conditions could be output to the operator's console.

Layer 5 was where the user programs were found. The did not have to worry about process, memory, console or I/O management. The system operator process was located in layer 6.

OS/2 operating system has a layered architecture. Earlier various of Windows NT had a layer - oriented organization.

Disadvantages

1. It requires appropriate definition of the various layers and a careful planning for the proper placement of the layers. For example the disk driver should be normally above the CPU scheduler as the driver may have to wait for the I/O and the CPU can be assigned to some other process by the scheduler, during this time.
2. Sometimes, it is felt that each layer adds an overhead to the system call executed by a user program for a specific purpose. As the result, system call can consume much more time as compared to the one executed in a non-layered system.

Advantages

1. The main advantage of the layered approach is modularity which helps in debugging and verification of the system easily.
2. The layers are designed in such a way that it uses operation and services only if a layer below it. A higher layer need not know how these operations are implemented, only what these operations do. Hence each layer hides implementation details from higher level layers. Any layer can be debugged without any concern about the rest of the layer.

1.11.3 *Virtual Machine Architecture For OS*

(The virtual machine operating system for IBM systems is the best example of the virtual-machine concept, because IBM pioneered the work in this area.)

(By using CPU scheduling and virtual-memory techniques, an operating system can create the illusion of multiple processor, each executing on its own processor with its own virtual/memory.)

That is, VM uses software techniques to make a single computer appear to be multiple smaller computer systems. Each of these small computers, simulated in software, acts as an independent and complete computer system. It manages a real computing system so that all of its resources - CPU, storage and I/O devices are available to many users at the same time. Each user has at his disposal the functional eqivalent of a real, dedicated computing system. Figure 1.25 shows the virtual machine architecture of an operating system. Because this functional equivalent is simulated by virtual machine and does not really exist, it is called a "virtual" machine.

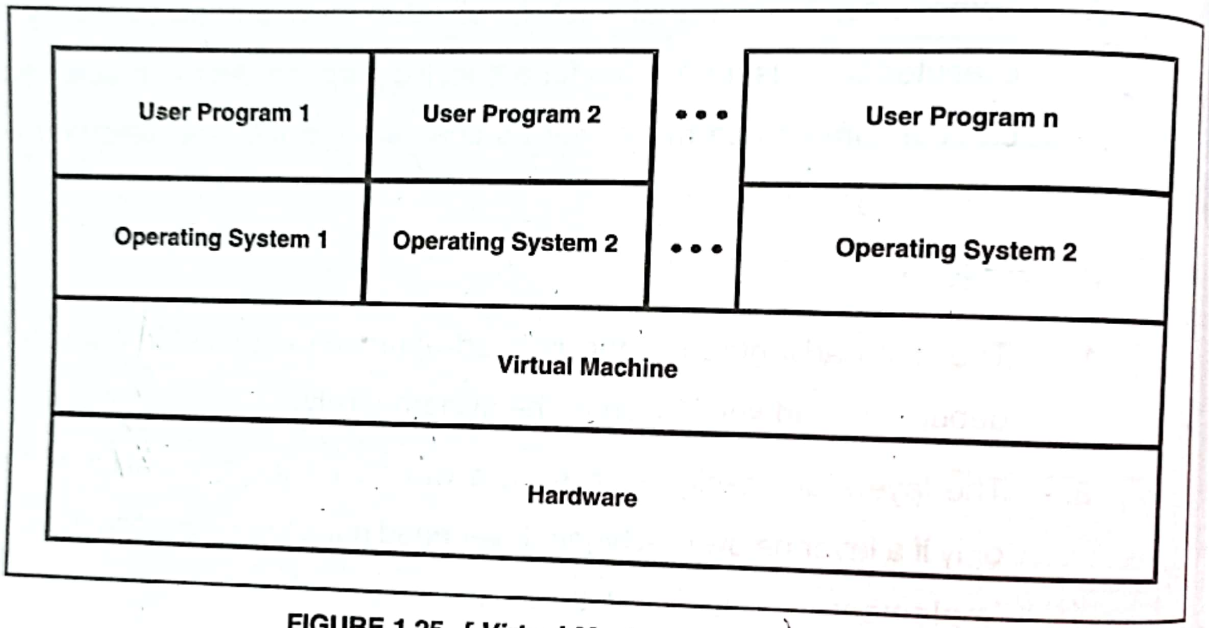


FIGURE 1.25 ***[Virtual Machine Architecture]***

Virtual machine has two main components :

Virtual Machine

- The control program (CP) controls the real machine
- The conversational monitor system (CMS) controls virtual machine

Hardware

The CP creates a virtual machine image in 'software' for each user of the computer system. By giving a virtual machine to each user, the Control Program provides effective protection to prevent one user from accidentally interfering with another. All real devices are controlled by CP and are made available to virtual machines on a more or less shared basis. Terminals connected to CP can logon on virtual machines.

The CMS is a single-user interactive operating system that runs on top of CP and interacts with the user and any application program running that virtual machine.

(The idea of virtual machine is used nowadays in a different context.

- Running MS-DOS programs on a Pentium
- The VM concept is used by Windows and other operating systems for running MS-DOS programs.
- Java is a very popular language designed by Sun Microsystems. Java is implemented by a complier that generates **bytecode** output. These bytecodes are the instructions that run on the **Java Virtual Machine** (JVM). Therefore, for Java programs to run on a platform, that platform must have a JVM running on it.

Disadvantages

This architecture is difficult to implement, as it requires lot of effort to provide an exact duplicate of the underlying machine.

Advantages

1. In this concept, there is complete protection of the various system resources.
2. Each Virtual Machine (VM) is completely isolated from all other virtual machines, so there are no security problems.
3. It is possible to define a network of VMs, each of which can send information over the virtual communications network.
4. A new field for OS research and development.
5. Virtual Machine Concept can solve the system compatibility problems.

1.11.4 *Client-Server Architecture for O.S. (or Micro-Kernel architecture)*

(A tend in modern operating systems is to take the idea of moving code up into higher layers even further and remove as much as possible from kernel mode, leaving a minimal **microkernel.**) The usual approach is to implement most of the operating system in user processes. To request a service, such as reading a bloc of a file, a user process (now known as the **client process**) sends the request to a **server process**, which then does the work and sends back the answer as shown in fig. 1.26.

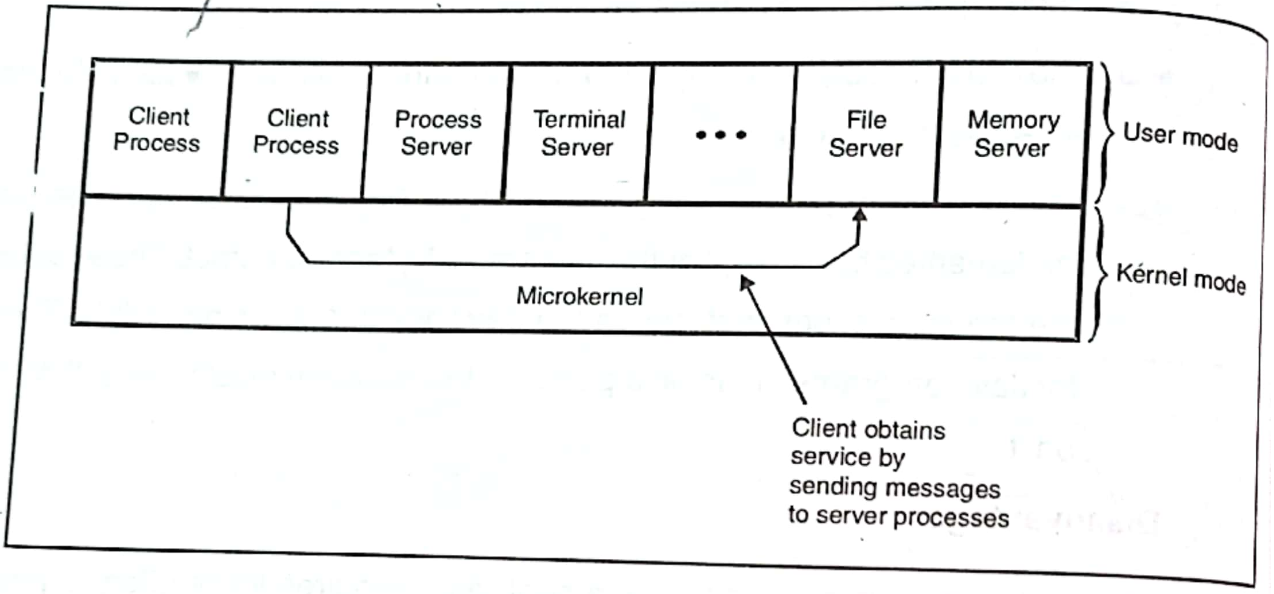


FIGURE 1.26 *[The client-server model]*

In client server model the operating system is divided into two kernels called user process (also called client-process) sends the request to a server process, which then does the processing and sent the user request-back with answer. By splitting the operating system up into parts each handles one facet of the system.

The main difference between client server model and all other models is that all the servers run as user-mode processes and not in kernel mode, they do not have direct access to the hardware.

The outcome of this feature is that if a long in the file server is triggered the file service may crash, but this will not usually bring the whole machine down.

Advantages

1. Client Server can be used in distributed system.
2. In client server only work kernel does is to communicate between client and server.
3. In client server the likelihood to system break-down is minimal.

2

Proces Managemen

A process can be viewed as a program in execution, and is used as a unit of wo for a processor. To put this another way, you can say that the process simultaneously has to manage several activities at one time and each activi corresponds to one process. A process will need certain resources such as CP time, memory, files and I/O devices to accomplish its task. These resources a allocated to the process either when it is created, or while it is executing.

Process Management manages the allocation of processes (tasks or jobs) to processor. The concept of process is central to the understanding of operatir system's functioning. Therefore, it is very important to understand and apprecial this concept fully.

2.1 *Operating System Terminology*

Let us consider few terms that are commonly used in relation to Operating System are defined as follows :

(i) **User :** A user is anybody who desires work to be done by a computer syster based on his instructions.

(ii) **Job :** Job is a collection of activities needed to accomplish the required task.

(iii) **Process (task) :** It is a computation that may be done concurrently with other computations.

(iv) **Address space :** The collection of programmes and data that are accessed during execution of a process forms an address space. Address space can be for I/O processes, CPU processes etc.

The relationship between these is shown in Fig. 2.1

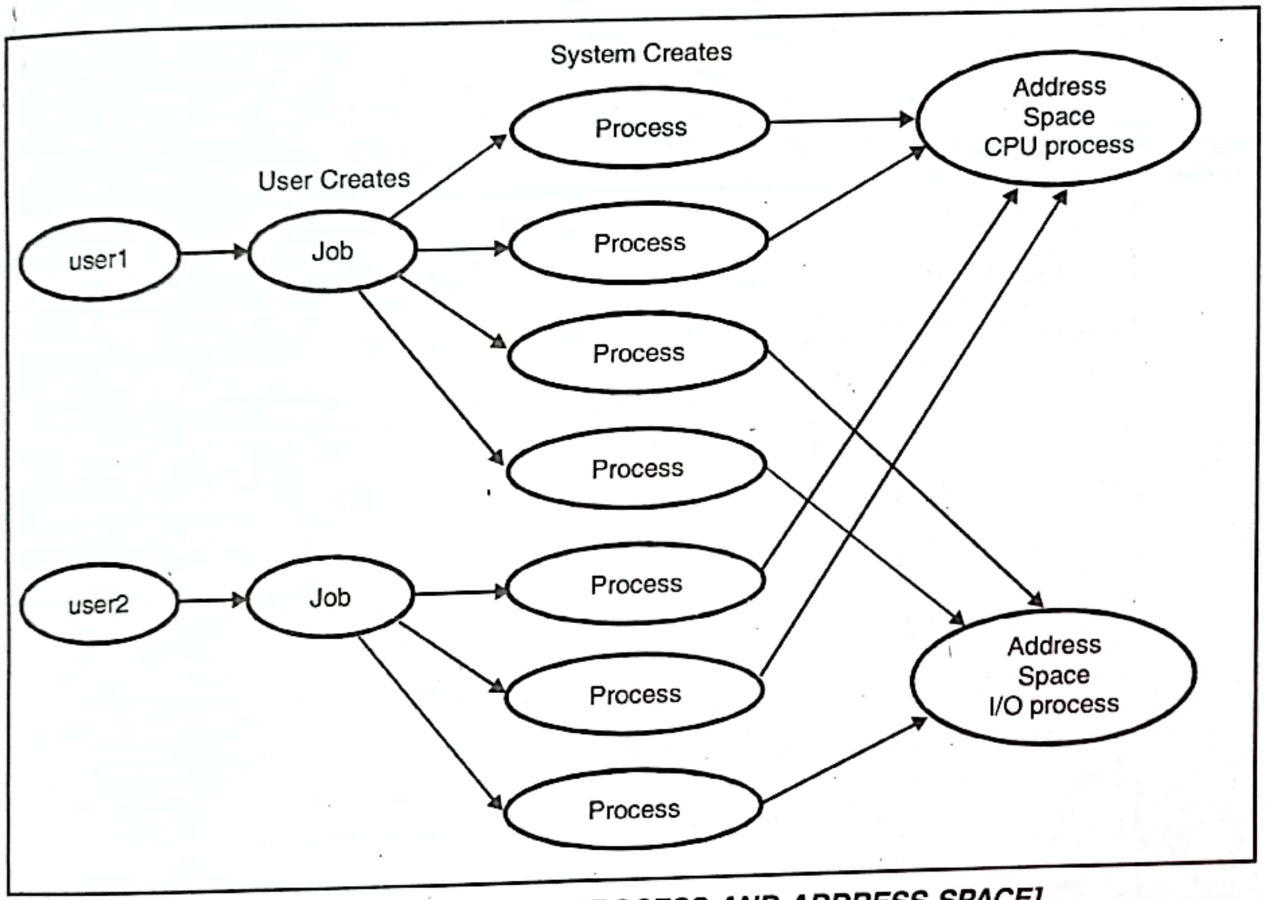


FIGURE 2.1 *[USER, JOB, PROCESS AND ADDRESS SPACE]*

2.2 *Definitions of "Process"*

In a multiple program environment, a single program does not have complete control of the system. This gave rise to the concept of process. The term "Process" has been given many definitions. Some of these definitions follows :

(i) A process is a program in execution.

(ii) A process is the "animated spirit" of a procedure.

(iii) A process is that entity to which processors are assigned.

(iv) A process is defined as an entity which represents the basic unit of work implemented in the system.

For example, a text editor program running on a computer is a process. A prog may cause several other processes to begin. For instance, a text editing prog can furnish a request for printing while ending the document. Here, the text edi a program that initiates two processes - one for editing the text and secon printing the document. There are numerous processes initiated by the system run without giving any direct evidence that they ever exist. Hence, process is initia by the program to perform an action, which can be controlled by a user, or by operating system.

2.3 *Difference Between a Program and a Process*

A **program** is defined as sequenced set of instructions whereas a **Process** is m than the program code, which is sometimes known as the **text section.** It a includes the current activity, as represented by the value of the **program coun** and the contents of the processor's registers. In addition, a process gener includes the process stack which contains temporary data such as functi parameters, return addresses and local variables and a data section, which conta global variables.

A program by itself is not a process. A program is a **passive entity** such as contents of a file stored on disk, whereas a process is an **active entity,** with a progr counter specifying the next instruction to execute & a set of associated resources.

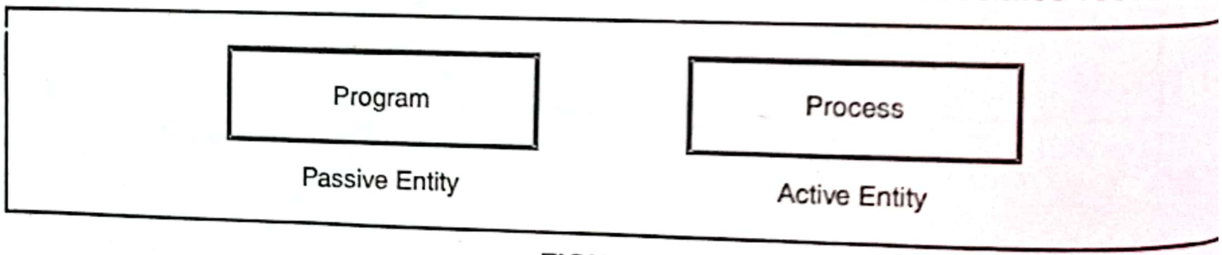


FIGURE 2.2

Note : Beside including instructions to be executed a process also includes

- Program counter value.
- Temporary data such as subroutine parameters.
- Return address
- Local and Global variables.

2.4 *Need of the "Process" Concept*

The concept of "process" is needed because the concept of a program is much less abstract. It is not sufficient to talk simply of 'programs' when considering the job of the operating system in supervising the execution of shared or re-entrant programs (Re-entrant means that the program can be executed two or more times simultaneously.)

When there are several users sharing a program in a multi-user environment, there are mainly two difficulties arising in font of O/S to execute these programs :

(i) The shared code in memory must not be altered during execution.

(ii) Separate data area must be maintained for each 'execution'.

The process concept is introduced to facilitate representation and control of executing programs in the face of these complixities. Each copy of a program and each separate execution of the one program are identified by a unique process within the operating system.

Fig. 2.3 shows the sharing of a program and shows the need for the concept of process.

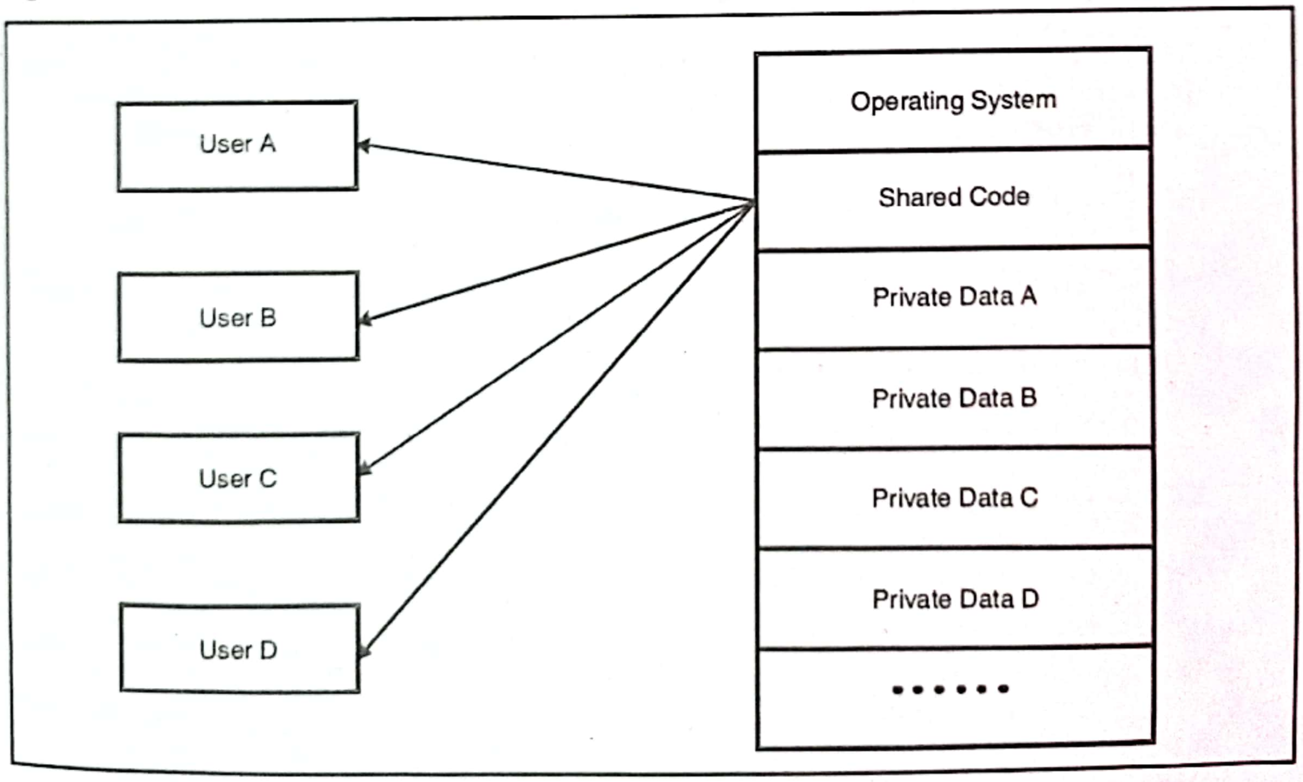


FIGURE 2.3

2.5 *Process States and Their Transitions*

When a process is born, its life in the system begins. During its existence, a pro goes through a series of discrete states or we can say that as a process exe it changes state.

Each process may be in one of the following states :

(i) New

(The process is being created.)

Wherever a user types a program to execute, the O/S attempt to find program and, if successful, the program code will be loaded and a system used to generate a process corresponding to the execution of the progra

(ii) Ready

(The process is waiting to be assigned to a processor.)

There are many process in the memory which may want to execute but th is one processor which can execute only one process at a time. So ot processes must have, to wait for the allocation of processer. A process wh is ready to execute by the processor is said to be in ready state.

(iii) Running

(Instructions are being executed.)

The C.P.U. is currently allocated to the process and the process is in executio

(iv) Waiting/Blocked State

(The process is waiting for some event to occur (such as an I/O completion reception of a signal)). A process comes in the blocked state if the last eve of interest to the process was a request made by it to the system. The reque is yet to be fulfilled, hence the process cannot execute even if the C.P.U. available to it.

(v) Terminated : The process has finished execution.

Fig. 3.4 shows state transitions of a process.

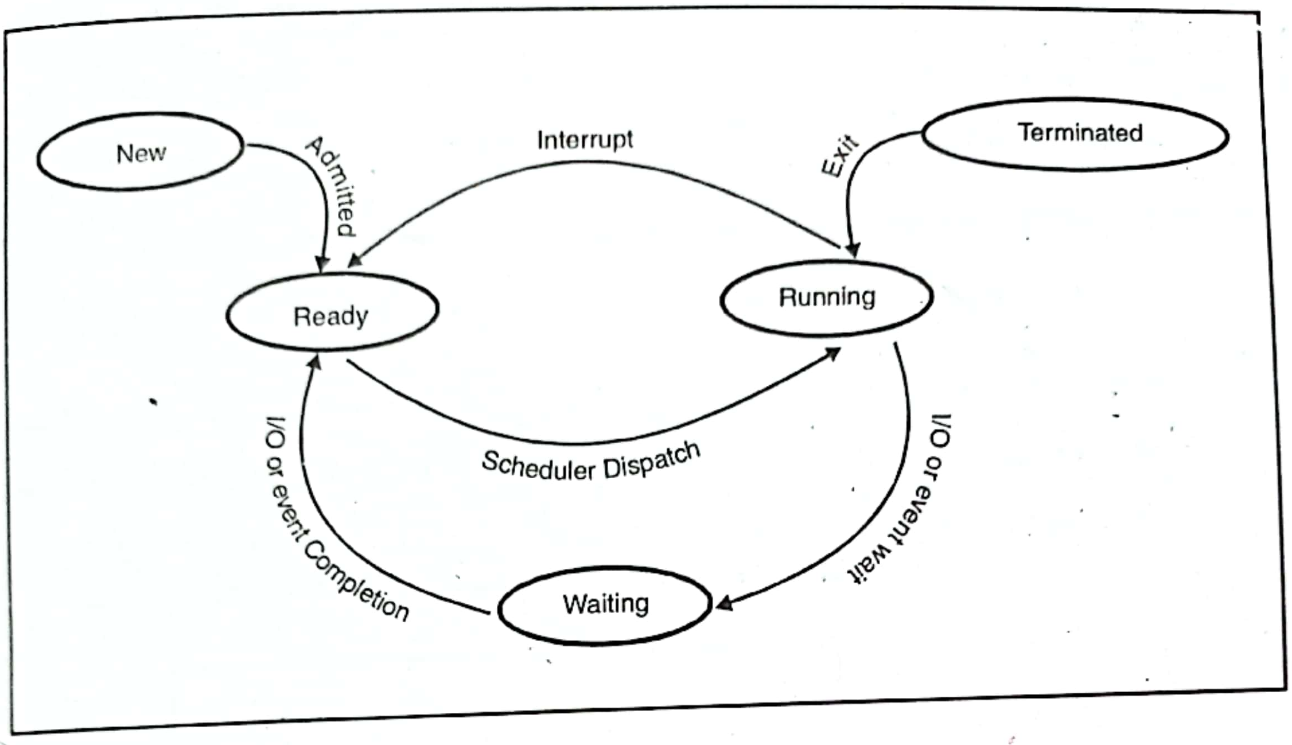


FIGURE 2.4

Some possible reasons for these transitions are :

(i) **Ready to Running :** A process is selected out of the ready queue (i.e. scheduled to be run) and is dispatched to the running state.

(ii) **Running to Waiting :** I/O operation is infact the most common reason for this type of transition. Here, since a process is waiting for I/O completion and thus do not need CPU for executing any other instructions, CPU is freed by it and CPU in the meantime takes up some other ready process and starts its execution.

(iii) **Waiting to Ready :** The reason for a process to remain in waiting state is no longer applicable e.g. I/O completes, timer goes off etc. Thus, the process becomes ready for its execution.

(iv) **Running to Ready :** The running process is pre-empted because some other process of higher priority has become ready.

2.6 *Process Control Block (PCB)*

The PCB is data structure containing certain important information about the proce
Each process has its own PCB to represent it in the operating system.

The PCB is a central store of information that allows the operating system to loca
all key information about a process.

A PCB may contain several types of informations depending upon the process
which PCB belongs. The information stored in PCB of any process may vary fro
process to process. But a general view of P.C.B. is shown in fig. 2.5.

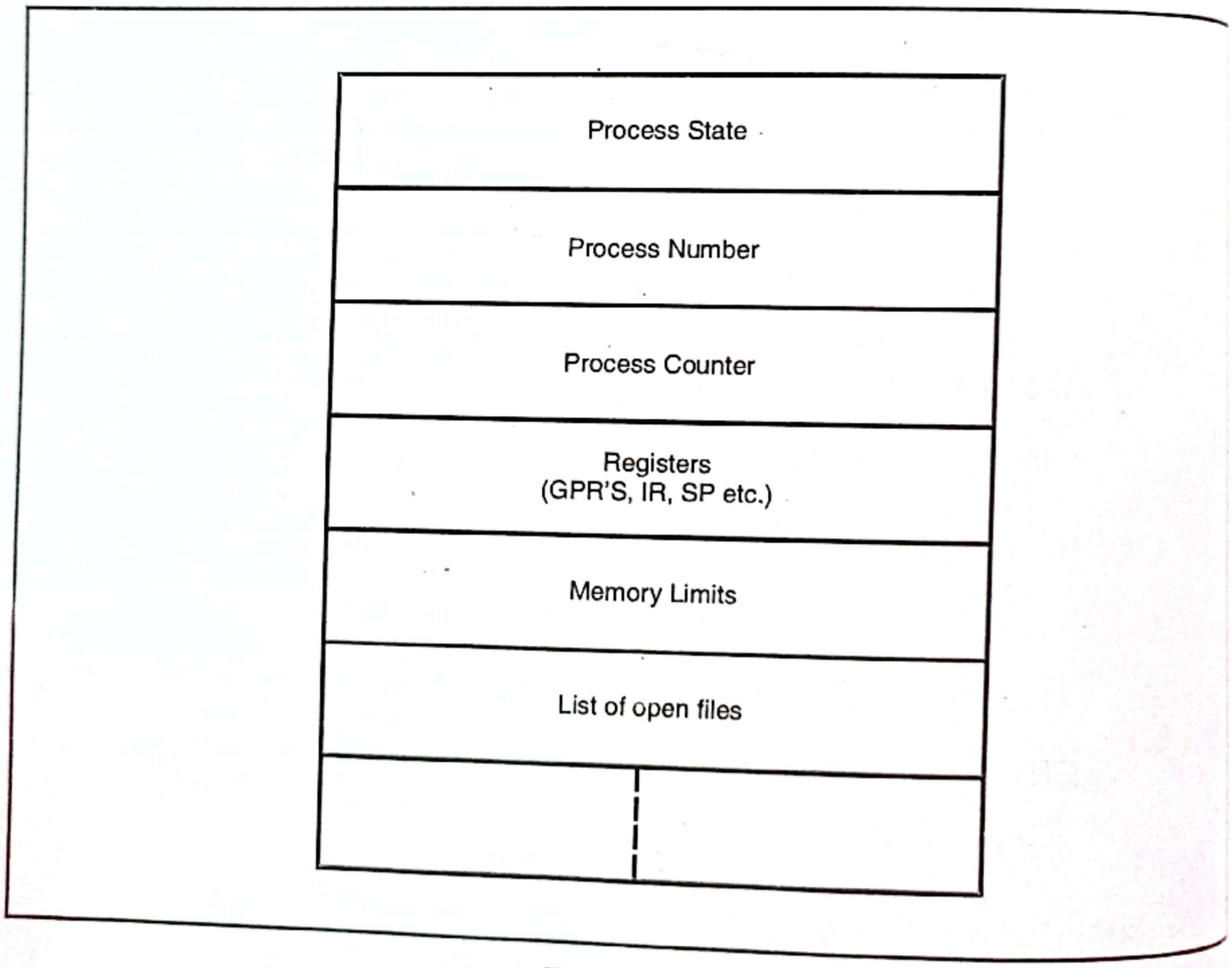


FIGURE 2.5

In general, a PCB may contain information regarding :

(i) **Process number :** Each process is identified by its process number, calle
process identification number (PID).

(ii) **Priority :** Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services.

(iii) **Process State :** This information is about the current state of the process i.e. whether process is in new, ready, running, waiting or terminated state.

(iv) **Program Counter :** This contains the address of the next instruction to be executed for this process.

(v) **CPU registers :** CPU registers vary is number and type, depending upon the computer architectures. These include index register, stack pointers and general purpose registers etc. When an interrupt occurred, information about the current status of the old process is saved in the registers along with the program counter. This information is necessary to allow the process to be continued correctly after the completion of an interrupted process.

(vi) **CPU Scheduling information :** This information includes a process priority, pointers to scheduling queues and any other scheduling parameters.

(vii) **Memory - management information :** This information may include such information as the value of base and limit registers, the page tables or the segment tables depending upon the memory system used by the O/S.

(viii) **Accounting :** This includes actual CPU time used in executing a process in order to charge individual users for processor time.

(ix) **I/O Status :** It includes outstanding I/O request, allocated devices information, pending operations, and so on.

(x) **File Management :** It includes information about all open files, access rights, etc.

2.7 *Operations On Processes*

Operating System that manage processes must be able to perform certain operations on processes. The set of operations on processes includes :

- Create a process
- Destroy a process
- Run a process
- Suspend a process
- Get process information
- Set process information

The operating system will supply mechanisms for at least some of these. Let us discuss process creation and process termination mechanisms provided by the operating systems.

2.7.1 Process Creation

Creating a process involves many operations including :

(i) name the process

(ii) insert it in the system's list of known processes

(iii) determine the process's initial priority

(iv) create the process control block.

(v) allocate the process's initial resources.

A process can create several new processes, via a create-process system call, during the course of execution. If it does, the creating process is called the **parent process** and the created process is called the **child process**. The act of creating a new process is often called **spawning** a process. Only one parent is needed to create a child. Such creation yields a **hierarchical process structure** like that shown in figure 2.6 in which each child has only one parent, but each parent may have many children.

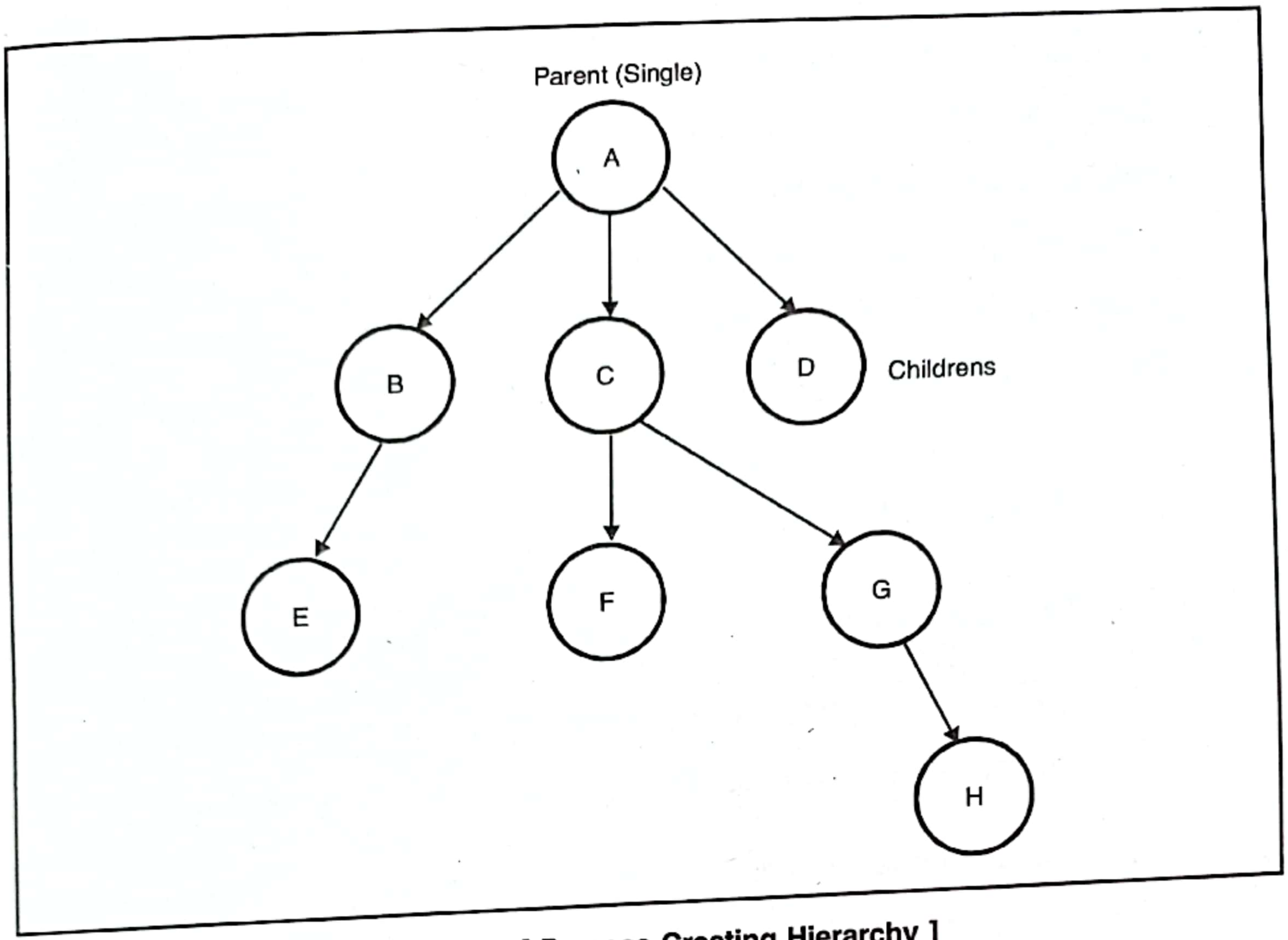


FIGURE 2.6 [Process Creating Hierarchy]

In general, a process & its sub-processes may need certain resources to accomplish its task. The subprocess may be also to obtain the desired resources in two ways :

(i) directly from the operating system

(ii) the parent may have to partition its resources among its children, or it may be also to share some resources among several of its children. Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many subprocesses.

2.7.2 *Process Termination*

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the **exit** system call. This involves returning of data to its parent process. All the resources of the process are deallocated by the operating system.

Termination occurs under additional circumstances. A process can cause the termination of another process via an appropriate system call e.g. abort . Usually, only a parent of the process that is to be terminated can invoke such a system call. Otherwise, users could arbitrarily kill each other's jobs. A parent therefore needs to know the identities of its children. Thus, when one process creates a new process, the identity of the newly created process is passed to the parent.

A parent may terminate the execution of one of its children due to the following reasons

(i) The task assigned to the child is no longer required.

(ii) The parent is exiting and the operating system does not allow a child to continue if its parent terminates. This is called **cascading termination**.

(iii) The child has exceeded its usage of some of the resources that it has been allocated. For this purpose, the parent must have a mechanism to inspect the state of its children.

2.8 *Process Hierarchies*

Operating Systems need ways to create and kill processes as discussed above. When a process is created, it further creates more processes and so on, thus forming a **process hierarchy** or **process tree** (as shown in above fig. 2.6. One of the processes acts as a parent to all other processes.

2.9 *Process Implementation*

Each process is represented in the operating system by a **process control block (PCB)** which contains information associated with a specific process. In Artical 2.6, PCB contents are already discussed.

Figure 2.7 shows how CPU passes control when multiple processes are running.

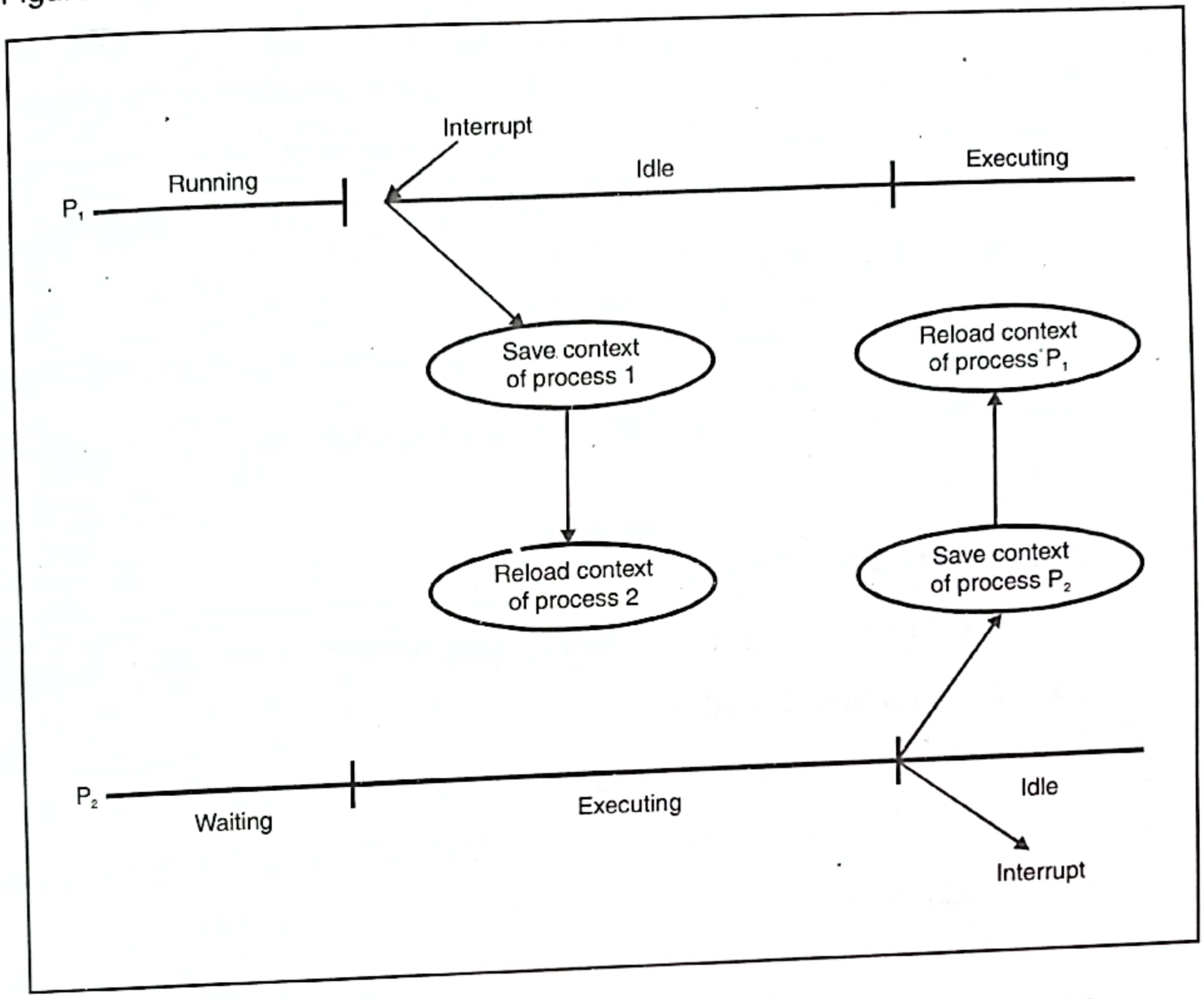

Interrupt
P₁
Running
Idle
Executing
Save context of process 1
Reload context of process P₁
Reload context of process 2
Save context of process P₂
P₂
Waiting
Executing
Idle
Interrupt

FIGURE 2.7 *[Passing of CPU control from one process to another process]*

Multitasking operating system is essentially *event-driven*, i.e. they perform operations in response to system events that can cause state changes and can lead to a resources reassignment. In an operating system it is essential to ensure that the system is informed about each event, and thus, enabled to process properly the changes of global system state.

Events may be *internal* such as sending and receiving signals or may be *external* such as completion of I/O operation. External events usually occur asynchronously and are signalled by means of **interrupts**. Internal events are usually synchronous and occur as a side effect of the execution of OS system calls invoked by the running process.

2.10 *Co-Operating Processes*

A process is said to be co-operating process if it can affect or be affected by the other processes executing in the system. On the other hand, if a process cannot affect or cannot be affected by the other processes in system, it is said to be an **independent process.**

Any process that shares data with other processes is a co-operating process.

There are several reasons why an environment for co-operating processes are provided :

(i) **Information Sharing**

Several users may want to share same piece of information.

(ii) **Computation Speed Up**

If we want a particular task run faster,we must divide it into subtasks, each of which will be executing in parallel with the others.

(iii) **Modularity**

We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.

(iv) **Convenience**

Even an individual user may have many tasks to work at one time. For example, a user may be editing, printing and compiling in parallel.

2.11 *Context Switch*

Switching the CPU to another process requires saving the state of the old process & loading the saved state for the new process. This task is known as a **Context Switch.**

The context of a process is represented in the PCB of a process; it includes the value of the CPU registers, the process state and memory - management information.

When a context switch occurs, the Kernel saves the context of the old process in its PCB & loads the saved context of the new process scheduled to run.

Context - Switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copies. Typical speeds range form 1 to 1000 microseconds.

Context - Switch times are highly dependent on hardware support. Also, the more complex the O/S, the more work must be done during a context switch.

2.12 *Threads*

Definition : A thread is a single sequential flow of control within a program.

A thread, also known as a **light-weight process**, is an independent sequence of execution within the context of a parent process. In the same way that one program may consist of several processes, so one such 'real' process may spawn several threads. Threads share the resources of the parent process (in particular, they share the same memory address space) but are separately & independently scheduled. Threads are created within, and belong to, processes.

A traditional (heavy weight) process has single thread of control. **If the process has multiple threads of control, it can do more than one task at a time.**

A thread, similar to a sequential program, has a beginning, an end, a sequence, and at any given time during the runtime of the thread, there is a single point of execution. However, a thread itself is not a program - it cannot run on its own - but runs within a program.

Like a process, a thread can be in any of several states (Running, Waiting, Ready or Terminated). Each thread has its own stack. A thread has a program counter (PC), a register set and stack space. Hence, threads provide a software approach to improving performance of operating systems by reducing the overhead of process switching.

2.13 Multithreading

The single thread of control allows the process to perform only one task at one time. An example of a single thread in a process is a **text editor** where a user can either edit the text or perform any other task like printing the document. In a multi-tasking operating system, a process may contain several threads, all running at the same time inside the same process. It means that one part of a process can be editing the text, while another part is printing the document.

Hence, **Multithreading** is the ability of an operating system to execute different parts of a program (threads) simultaneously. The programmer must carefully design the program in such a way that all the threads can run at the same time without interfering with each other.

While multithreading offers many benefits on single-processor machines, multithreaded applications are essential for taking full advantage of multiprocessor computers. Thus, to achieve maximum performance from multithreaded operating systems and multiprocessor machines, an application must be multithreaded.

Advantages

Windows NT, Windows 95, and some variants of UNIX operating system are multithreaded and run on either a single-processor machine or a multiprocessor computer. On single-processor machines, multithreading can help applications where :

(i) Some task takes much longer to execute than others

(ii) Some task needs a better deterministic outcome than others

(iii) Some user interface activity is to be run concurrently with hardware communications.

On multiprocessor systems, multithreading of an application takes advantage of the additional hardware and can result in greatly improved overall performance.

2.13.1 *Difference Between a traditional Single-threaded process & a multithreaded process*

In essence, threads can perform many of the funcitons of processes but are much **'cheaper'** in terms of system overheads. The operating system effort in creating a new process or switching from one process execution to another is much greater than the equivalent actions using threads.

However, the disadvantages of threads is that, since they share the same resources (such as main memory), they may conflict in their accessing of these resources; hence, it may be necessary for the application to mediate between competing threads.

Difference between a single-threaded process & a multiple-threaded process can be understood with the help of following example.

In certain situations, a single application may be required to perform several similar tasks. e.g. a web server accepts client requests for web pages, images, sound & so forth. A busy web server may have several of clients concurrent accessing it. If the web server ran as a traditional single - threaded process, it would be able to service only one client at a time. The amount of time that a client might have, to wait for its request to be serviced could be enomorous.

One solution is to have, the server run as a single process that accept requests. When the server receives a request, it creates a separate process to service that request. In fact, this process - creation method was in common use before threads became popular.

Process creation is very heavyweight, as was explained above. Also, if the new process will perform the same tasks as the existing process, why incur all that overhead? It is generally more efficient for one process that contains multiple threads to serve the same purpose. This approach would multithread the web - server process. The server would create a separate thread that would listen for client requests; when a request was made, rather than creating another process, it would create another thread to service the request.

2.13.2 Life - Cycle of Threads

Threads have a similar life-cycle to the processes & are mainly managed in the same way. Initially each process is created with a single thread. However, threads are usually allowed to create new once using particular system calls. Then, a thread tree is typically created for each process as shown in the figure 2.8.

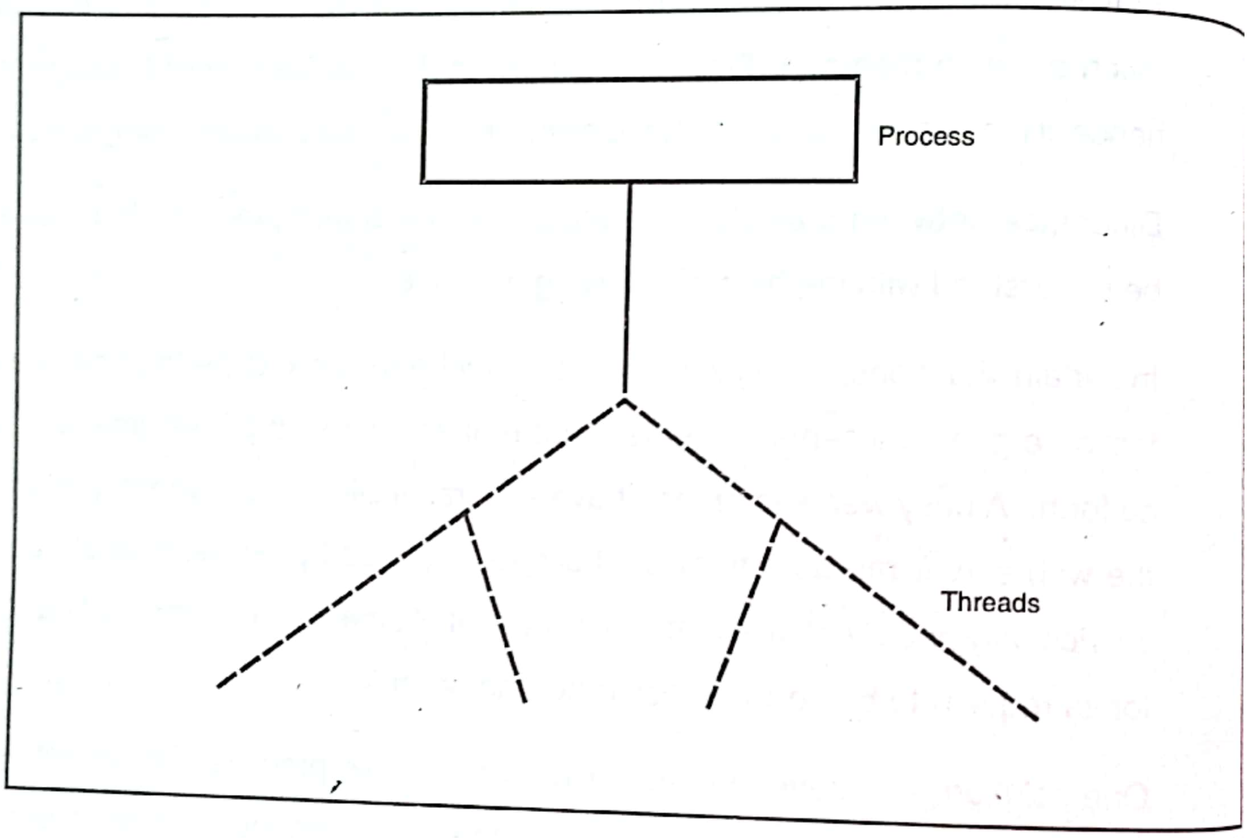


FIGURE 2.8 ***[Threads Tree]***

During creation, each thread is declared by a data structure usually called a **Thread Control Block** (TCB). Correspondingly, threads can be basically in one of three states: **running, ready to run or blocked.** Again, a state transition diagram describes possible state transfer.

Thread management is performed by setting up TCB queues for each state & performing state transitions according to the transition diagram & the scheduling policy.

2.14 *Processes vs Threads*

As we mentioned earlier that in many respect threads operate in the same way as that of processes. An idea of how threads and processes can be related to each other is shown in figure 2.9.

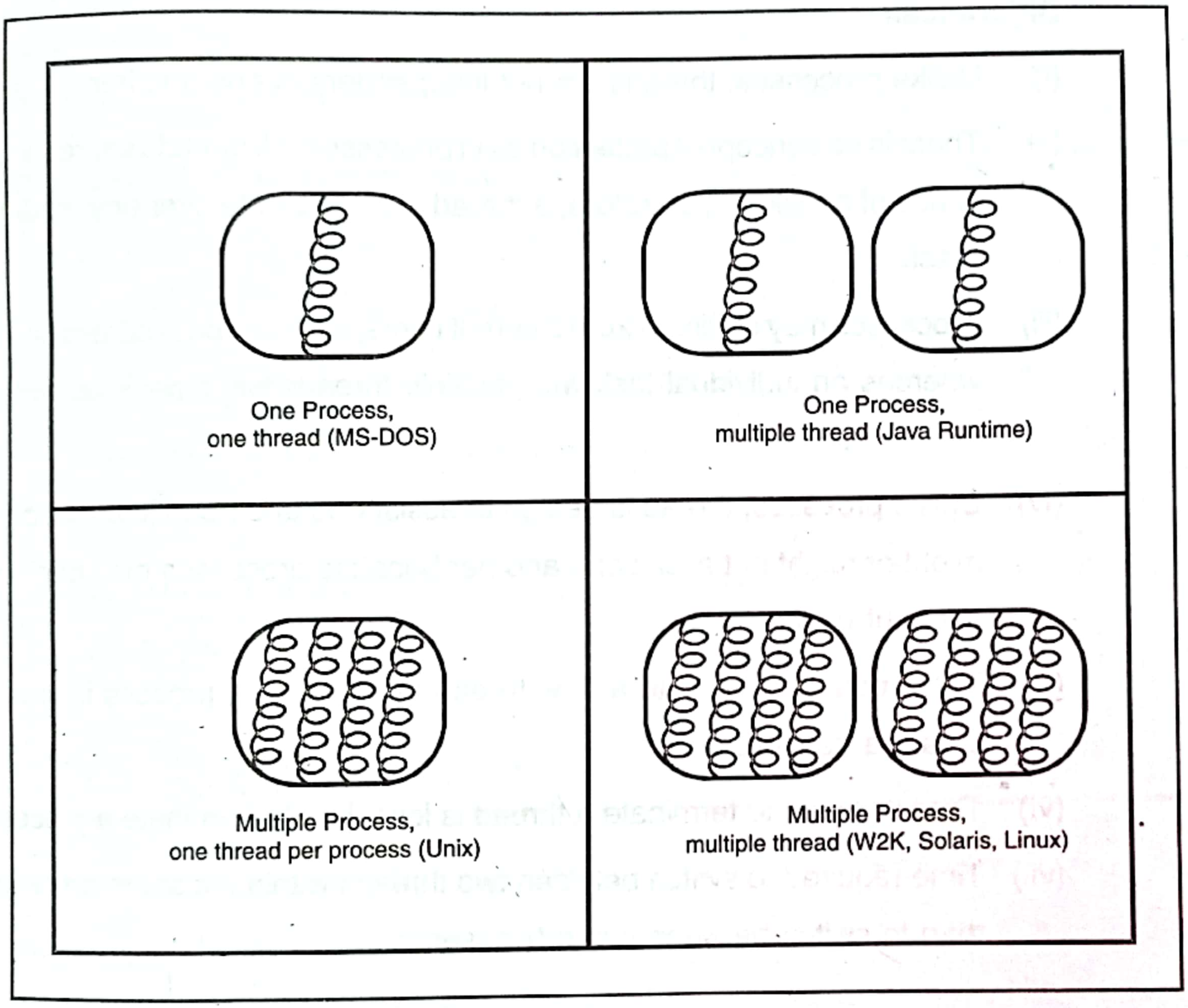

One Process,
one thread (MS-DOS)
One Process,
multiple thread (Java Runtime)
Multiple Process,
one thread per process (Unix)
Multiple Process,
multiple thread (W2K, Solaris, Linux)

FIGURE 2.9 ***[Threds and Process]***

Let us consider some similarities and differences between Threads and Processes.

Similarities

(i) Like processes threads share CPU and only one thread is active (running) at a time.

(ii) Threads also have almost the same states as processes - ready, running, waiting.

(iii) Like processes, thread can create children.

(iv) Each thread has its own stack and program counter.

(v) Like processes, if one thread is blocked/waiting, another thread can run.

(vi) Within a process, a thread executes sequentially.

Differences

(i) Unlike processes, threads are not independent of one another.

(ii) There is no concept of protection as in processes. All threads share the addre space of a task and therefore, a thread can read/write over any other threa stack.

(iii) Processes may originate from different users, and may be hostile to one anothe whereas an individual task with multiple threads will always be owned by single user.

(iv) Unlike proesses, thread is design to assist one another. Note that processe might or might not assist one another because processes may originate fro different users.

(v) Time required to create a new thread in an existing process is less than t create a new process.

(vi) Time required to terminate a thread is less than to terminate a process.

(vii) Time required to switch between two threads within the same process is les than to switch between two processes.

2.15 *CPU Scheduling*

In a multiprogramming system, at any given time, several processes will b competing for the CPU's time. At any instance, only one process will be in th running state while others will be in ready or waiting state. The operating syste determines how to allocate processor time among all the ready processes. Th allocation is referred as *scheduling*. Scheduling is a fundamental operating syste function.

Scheduling refers to a set of *policies* and machanisms built into the operating system that define the order in which the computer system works. Scheduler selects the next job to be admitted into the system for processing.

2.15.1 *Objectives of Scheduling*

The objective of scheduling are

(i) **Maximize the system throughput.**

(ii) **be 'fair' to all users.**

(iii) **provide tolerable response or turn-around time.**

(iv) **degrade performance gracefully.** If the system becomes overloaded, it should not 'collapse', but avoid further loading or temporarily reduce the level of service.

(v) **be consistent & predictable.** The response & turn around time should be relativly stable day to day.

2.16 *Scheduling Criteria* Imp.

Different CPU-scheduling algorithms have different properties and may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. Many criteria have been suggested for comparing CPU-scheduling algorithms. The characteristics used for comparison can make a substantial difference in the determination of the best algorithm.

Let us consider various scheduling criteria.

(i) **CPU Utilisation :** CPU should remain as busy as possible. In a real system, CPU utilisation should range from 40% to 90%.

(ii) **Throughput :** It refers to the amount of work completed in a unit of time. O way to measure throughput is by means of the number of processes that a completed in a unit of time. The higher the number of processes, the mo work apparently being done by the system. But this approach is not ve useful for comparison because this is dependent on the characteristics a resource requirement of the process being executed. For example, if all t processes are CPU bound, throughput will be less as compared to I/O bou processes. Therefore to compare throughput of several scheduling algorith it should be fed to the process with similar requirements.

(iii) **Turnaround Time :** It may be defined as interval from the time of submissio of a process to the time of its completion. It is the sum of the periods spe waiting to get into memory, waiting in the ready queue, CPU time and I/ operations. It should be as less as possible.

(iv) **Waiting Time :** In multiprogramming operating system several jobs reside a a time in memory, CPU executes only one job at a time. The rest of jobs wa for the CPU. Waiting time is the time spent in for waiting for resource allocatio due to contentions with others in multiprogramming system by the process. I is calculated by the following equation.

$$W(x) = T(x) - x$$

where $W(x)$ = waiting time

$T(x)$ = Turnaround time

x = units of services

(v) **Response Time :** In an interactive sytem, response time is the best metric. I is defined as the time interval between the job submission and the first response produced by the job. Response time should be minimized in an interactive system.

.17 *Scheduling Queues*

A Process when it is not executing, is placed in some waiting queue. There are two major classes of queue in an O/S: the ready queue and I/O request queue.

(i) **Ready queue :** The ready queue contains all the processes that are ready to execute and are waiting for the CPU. Each process is represented by a PCB.This queue is generally stored as a linked list. A ready - queue header contains pointers to the first and final PCBs in the list. Each PCB has a pointer field that points to the next PCB in the ready queue as shown in fig. 2.10.

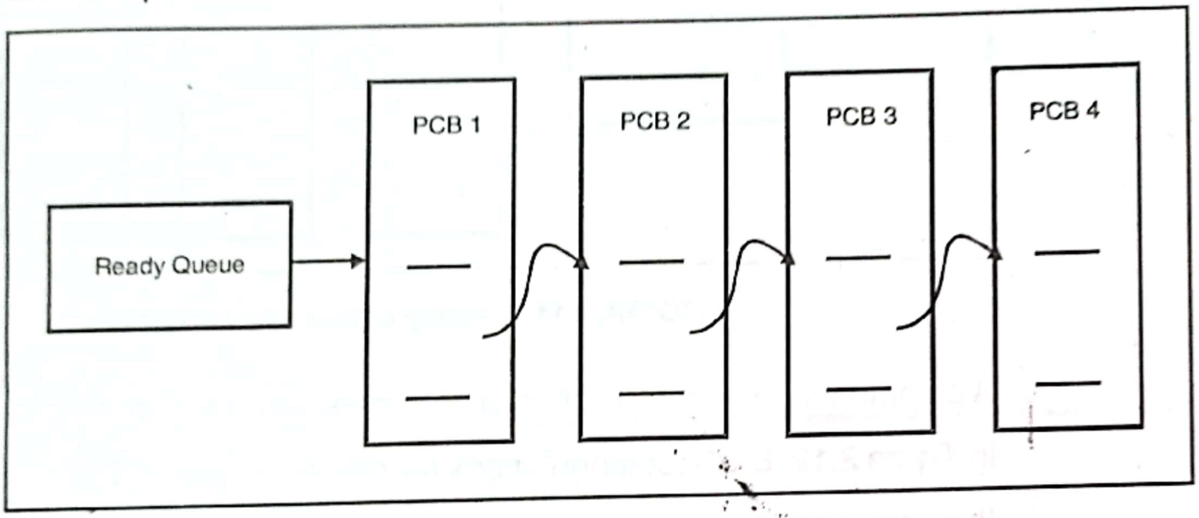


FIGURE 2.10 *[Ready Queue as a Linked List]*

(ii) **I/O request queue :** The operating system also has other queues. When a process is allocated the C.P.U., it executes for a while and eventually quite, is interrupted, or wait for the occurence of a particular event, such as the completion of an I/O request. In the case of an I/O request, such a request may be to a printer, or to a shared device, such as a disk. Since the system has many processes, the printer may be busy with the I/O request of some other process. The process therefore may have to wait for the printer. The list of processes waiting for a particular I/O device or another resource is called a **device queue** or **blocked queue**. Each resource has its own queue as shown in fig 2.11.

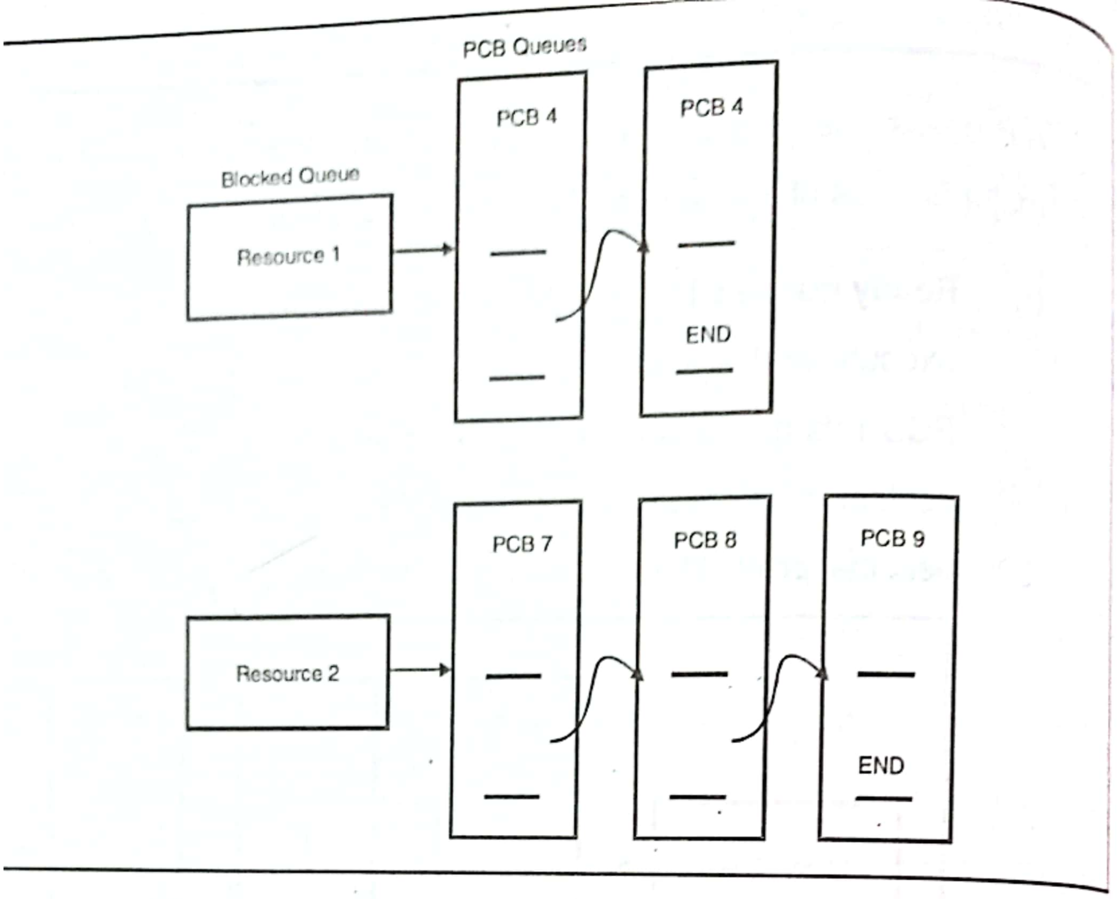


FIGURE 2.11 ***[Ready Queue as a Linked List]***

A common representaiton of process scheduling is a queueing diagram shown n figure 2.12. Each rectangular box represents a queue. The circles represent he resources that serve the queues, and the arrows indicate the flow of processes in the system.

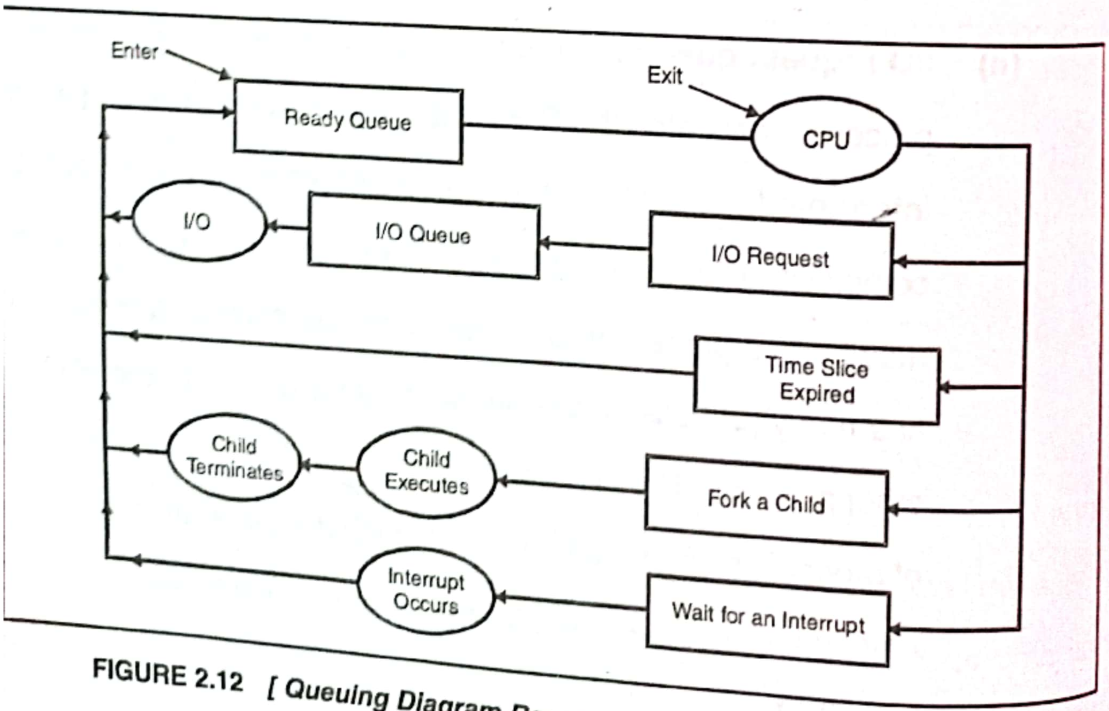


FIGURE 2.12 ***[Queuing Diagram Representation of Process Scheduling]***

A new process is initially put in the ready queue. It wait in this queue until it is selected for execution. Once the process is assigned to the CPU and is executing, one of the several events could occur :

(i) The process could issue as I/O request, and then be placed in an I/O queue.

(ii) The Process could be removed from the CPU due to expire of time slice in time sharing system.

(iii) The process could create a new subprocess and wait for its termination.

(iv) The process could be removed forcibly from the CPU, as a result of an interrupt and be put back in the ready queue.

A process continues this cycle until it terminated, at which time it is removed from all queues.

2.18 *Levels of Scheduler*

Scheduler is an operating system module that takes decision for admitting next job into system for execution.

Scheduling can be exercised at three distinct levels, which we will refer to as High-levels, Medium-level and low-level.

(i) **High - level Scheduler (or long - term or job scheduler)**

It deals with the decision as to whether to admit another new job to the system.

The High-Level Scheduler (HLS) controls the admissio[illegible] jobs into the system i.e. decides which newly submitted jobs are to be c[illegible]verted into processes and be put into the **ready queue** to compete for access to the processor. This activity is only really applicable to **batch systems**, since in as on-line environment, processes will be admitted immediately unless the system is fully loaded.

New jobs entered into the system will be put into a queue awaiting acceptan by the HLS. The principle control which the HLS exercise is ensuring that th computer is not overloaded, in the sense that the number of active processe (the degree of multiprogramming) is below a level consistant with efficien running of the system. If the loading level is maximum, new processes will no be admitted until a current process terminates.

It the loading level is below maximum, a waiting process will be selected from the queue on the basis of some selection algorithm e.g. First come first served (FCFS) or SJF (Shortest job first). These algorithms are also used within low-level scheduling by short-term schedular.

(ii) **Medium Level scheduler**

The key idea behind a medium term scheduler is that sometimes it can be advantageous to remove processes from memory, and thus to reduce the degree of multi programming. At some later time the process can be Swap in reintroduced into memory and its execution can be continued where it left off. This scheme is called **swapping** as shown in fig. 2.13.Time sharing system use medium level scheduling.

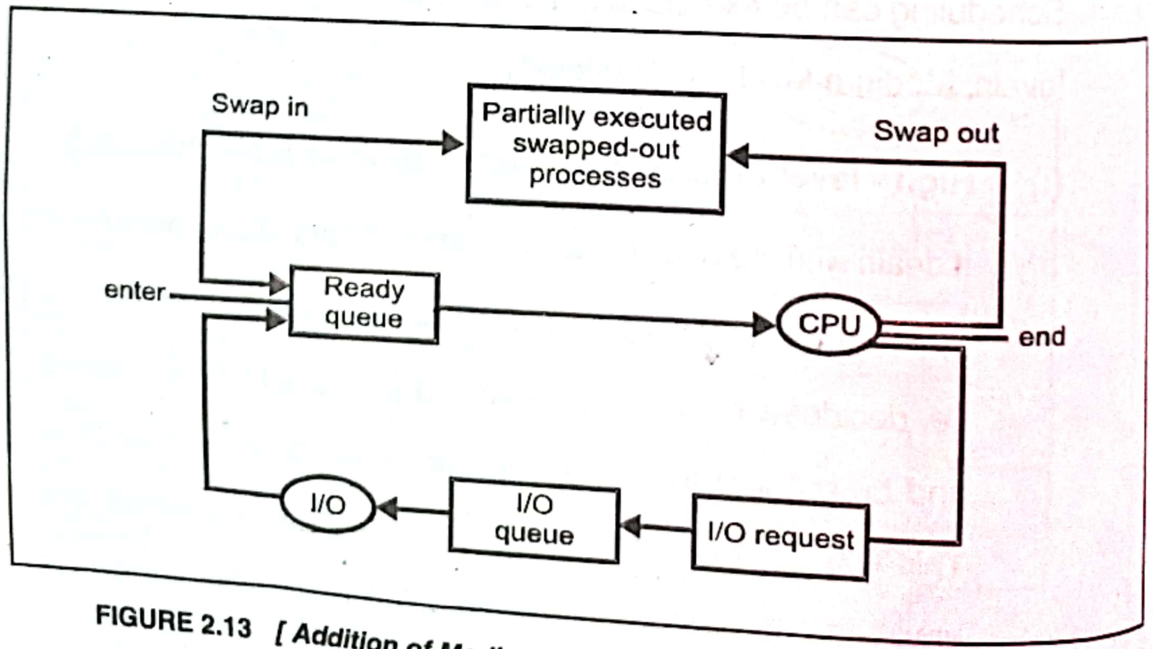


FIGURE 2.13 *[Addition of Medium Term Scheduling to Queueing Diagram]*

(iii) Low-level Scheduler (or Short-term scheduler or CPU scheduler)

The low level scheduler (LLS) is the most complex and significant of the scheduling levels. Whereas the high and medium level schedulers operate over time scales of seconds or minutes, the LLS is making critical decisions many times every second. The LLS (or CPU scheduler) selects from among the processes that are ready to execute, and allocates the CPU to one of them. A number of different policies have been devised for use in low level schedulers, each of which has its own advantage and disadvantages. LLS is performed by the dispatcher that operates many times per second.

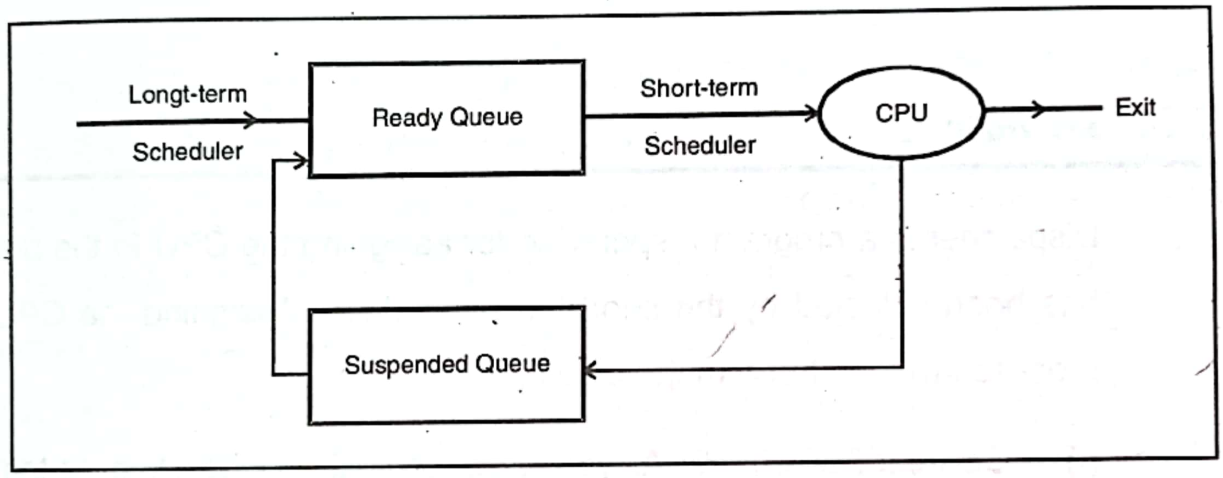


FIGURE 2.14

2.19 *Types of Scheduling*

There are two types of scheduling algorithms

(i) Non-preemptive scheduling

(ii) Preemptive scheduling

(i) **Non-preemptive scheduling :** In non preemptive scheduling, a scheduling decision is made every time some job in the system finishes its execution (and at system initialization time). It means once a process has been given the C.P.U, the CPU cannot be taken away from that process. In non-preemptive scheduling, short jobs are made to wait by longer jobs, but the treatment of all process is fairer. Response times are more predictable because incoming high priority jobs cannot displace waiting jobs.

(ii) **Preemptive scheduling :** On the other hand, in preemptive scheduling, a scheduling decision can be made even while the execution of a job is in progress. Consequently, a job in execution may be forced to release the processor so that execution of some other jobs can be undertaken. Preemptive scheduling is useful in systems in which high-priority processes require rapid attention. To make preemption effective, many processes must be kept in main storage so that the next process is normally read for CPU when it becomes available. Keeping non-running programs in main storage also involves overhead.

2.20 *Dispatcher*

Dispatcher is a program responsible for assigning the CPU to the process, which has been selected by the short-term scheduler. Assigning the CPU to a ready process involves three major steps :

(i) **Context Switching :** As you know, context switching implies the switching of CPU from one process to another. It requires saving the state of old process and loading the saved state, it any, for this ready process.

(ii) **Switching to user mode from monitor mode :** As we know, a user processe must be run in user mode therefore, mode must be changed from monitor mode to user mode.

(iii) **Jumping to the proper location (or instruction) in the user program to restart that program :** The execution of process should be restarted by jumping to the instruction that was supposed to be executed when this process was last interrupted or to the first instruction if this ready process is to be executed for the first time after its creation.

The dispatcher should be as fast as possible as it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

Unit - II

2.21 CPU Scheduling Algorithms

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated CPU first. The CPU is allocated to the selected process by dispatcher. For scheduling CPU and to make proper & maximum utilization of CPU, we have many algorithm as follows :

2.21.1 First-Come, First-Served Scheduling : (FCFS)

As the name implies, the FCFS policy simply assigns the processor to the process which is first in the ready queue. Key concept of this algorithm is "allocate the processor (CPU) in the order in which the processor arrive". It is also known as First in First out (FIFO). FCFS is non-scheme discipline. It is fair in formal sense but somewhat unfair in that long jobs make short jobs wait, and unimportant jobs make important jobs wait. The average waiting time under FCFS policy is quite long.

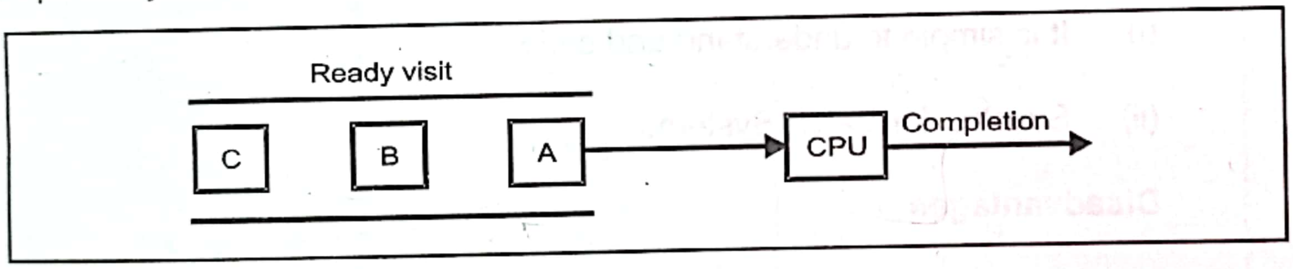


FIGURE 2.15 ***[First-In-Fist-Out Scheduling]***

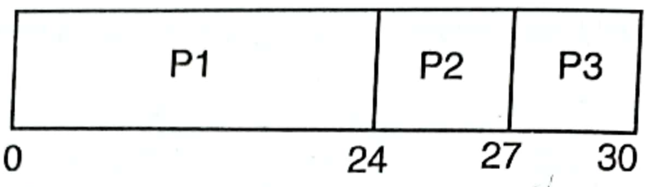
Example 1

Consider the following set of processes that arrive at time 0, with the length of CPU Burst time (or run time) given in milliseconds. CPU - burst time indicates that for how much time, the process needs the CPU.

Process	Burst Time	Waiting time
P1	24	0
P2	3	24
P3	3	27

If the processes arrive in the order P1, P2, P3 and are served in FCFS order, w
get the result in "GRANTT CHART".

The waiting time is 0 millisecond for process P1, 24 milliseconds for process p2
and 27 milliseconds for process p3.



Thus the Average Waiting Time

$$= \frac{0+24+27}{3} = 17\text{ms} \qquad = \frac{\text{Waiting Time}}{\text{No. of Processes}}$$

Thus the average waiting time under FCFS policy is generally very long. FIFO is rarely used on its own but it is often embedded within other schemes.

Advantages

(i) It is simple to understand and code.

(ii) Suitable for Batch Systems.

Disadvantages

(i) Waiting time can be large if short requests wait behind the long ones.

(ii) It is not suitable for time sharing systems where it is important that each user should get the CPU for an equal amount of time interval.

(iii) A proper mix of jobs (I/O based and CPU based jobs) is needed to achieve good results from FCFS scheduling.

2.21.2 Shortest-Job- First Scheduling (SJF)

Key concept of this algorithm is :

"CPU is allocated to the process with least CPU-burst time"

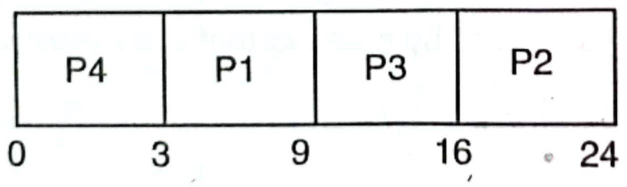
Amongst the processes in the ready queue, CPU is always assigned to the process with least CPU burst requirement. SJF reduces average waiting time over FCFS. When CPU is available it is assigned to the process that has the smallest run time. If two processes have the same run time, FCFS is used to break the tie. The shorter the job, the better service it will receive. This tends to reduce the number of waiting jobs, and also reduces the number of waiting jobs, and also reduces the number of jobs waiting behind large. As a result, SJF can minimize the average waiting time of Jobs. The obvious problem with SJF is that it require precise knowledge of how long a job or process will run.

Example 2

Consider the set of processes, with the length of CPU burst time given in milliseconds:

Process	Burst Time	Waiting time
P1	6ms	3
P2	8ms	16
P3	7ms	9
P4	3ms	0

"GRANTT CHART"



Waiting Time is 3 milliseconds for process P1, 16 milliseconds for process P2, 9 ms for process P3 and 0 millisecond for process P4

$$\text{Average Waiting Time} = \frac{3+16+9+0}{4} = \frac{28}{4} = 7\text{ms}$$

If we were using FCFS policy, Average waiting Time would be 10.25 ms.

Thus SJF algorithm is optimal because it gives the minimum average waiting time.

Advantages

(i) Minimum average waiting time.

Disadvantages

(i) The problem is to know the length of time for which CPU is needed by a process. A prediction formula can be used to predict the amount of time for which CPU may be required by a process.

2.21.3 *Priority Scheduling*

A priority is associated with each process and the CPU is allocated to the process with highest priority. Equal priorities again are scheduled in FCFS order. Priorities can be defined either internally or externally. Internally defined priorities are some measurable qunatities. e.g. Time Limits, memory requirement, The number of open files etc.

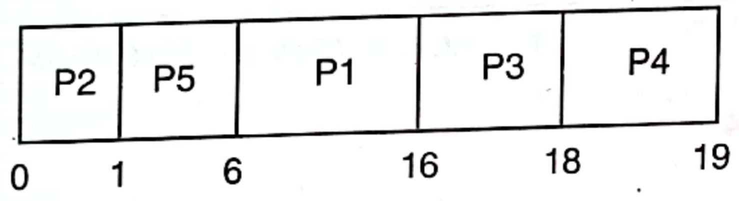
External priorities are set by crieteria that are external to Operating system, such as importance of process, type of process etc. Priority scheduling can be either preemptive or non-preemptive. When a process arrives at ready queue, its priority is compared with the priority of currently running process. The CPU will be allocated to the new process if the priority of the newly arrived process is higher than the priority of the currently running process. On the other hand non-preemptive priority scheduling will simply put the new process at the head of ready Queue.

Example 3

Consider the processes :

Process	Burst Time	Priority	Waiting time
P1	10	3	6
P2	1	1	0
P3	2	4	16
P4	1	5	18
P5	5	2	1

"GRANTT CHART"



The Average Waiting time under priority scheduling algorithm is

$$= \frac{6+0+16+18+1}{5} = \frac{41}{5} = 8.2\text{ms}$$

Problem with priority scheduling : Preemptive priority scheduling some times becomes the biggest cause of **indefinite blocking** or **starvation** of a process. If a process is in ready state but its execution is almost always preempted due to the arrival of higher priority processes, it will starve for its execution. Therefore, a mechanism like **aging** has to be built into the system so that almost every process should get the CPU in a fixed interval of time. This can be done by increasing the priority of a low-priority process after a fixed time interval so that at one moment of time it becomes a high priority job as compared to others and thus, finally gets CPU for its execution.

2.21.4 *Round Robin Scheduling (RR)*

In round robin scheduling, processes are dispatched FIFO but are given a lim amount of CPU time called a Time -slice or a quantum. If a process does not comp before its CPU time expires, the CPU is preempted & allocated to next wa process. The preempted process is then placed at the back of ready Queue scheduling is effective in time-sharing environments in which the system need guarantee reasonable response times for interactive users.

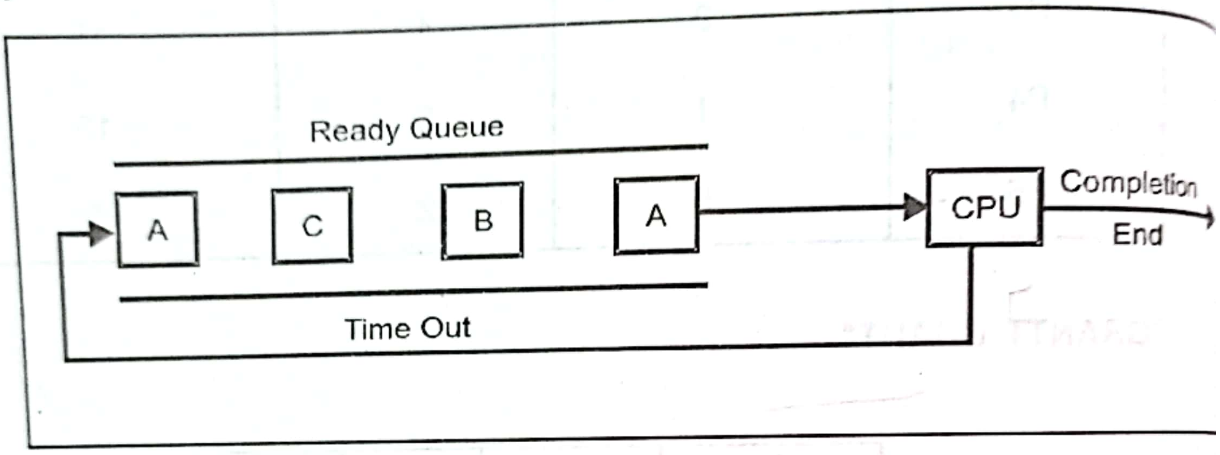


FIGURE 2.16 *[Round Robin Scheduling]*

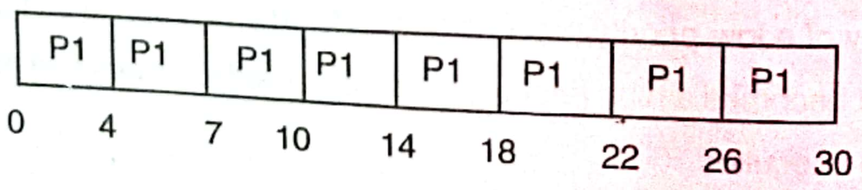
Example 4

Consider the processes :

Process	Burst Time
P1	~~24~~ ~~20~~ 16
P2	3
P3	3

Set Time slice = 4ms

then **"GRANTT CHART"**



Waiting time for P1 will be = 0 + (10 - 4) = 0 + 6 = 6ms

Waiting time for P2 will be = 4 ms

Waiting time for P3 will be = 7 ms

Average waiting time = 6 + 4 + 7 = 5.66 ms

Advantages

(i) It is simple to understand

(ii) Suitable for Interactive Systems or time sharing systems

Disadvantages

(i) Performance depends heavily on the size of the time quantum

(ii) Number of context switches - As mentioned above, the number of context-switches should not be too many to slow down the overall execution of all the processes. Time quantum should be large with respect to the context switch time. This is to ensure that a process keeps CPU for a considerable amount of time as compared the time spent in context switching.

2.22 *Multilevel Queue Scheduling*

Multilevel Queue scheduling was created for situation in which processes are easily classified into different groups. It has the following steps.

(i) A multi-level-queue scheduling algorithm partitions the ready queue into separate queues. Processes are permanently assigned to each queue, based upon properties such as, interactive jobs, batch jobs, memory sizes, and so on.

(ii) Each queue has its own scheduling algorithm. One may be using FCFS, the other round-robin and so on.

(iii) There must be scheduling algorithm between the queues. Usually this is a fixed priority preemptive scheduling. For example, the foreground queue may have absolute priority over the background queue. A percentage-based time sliced approach can also be used. Each queue gets a certain portion of the CPU time, which it can then schedule among the various processes in its queue.

Example 5

Consider a example of Multilevel Queue scheduling is : - having Five queues.

(i) System processes

(ii) Interactive processes.

(iii) Interactive editing processes.

(iv) Batch processes

(v) User processes

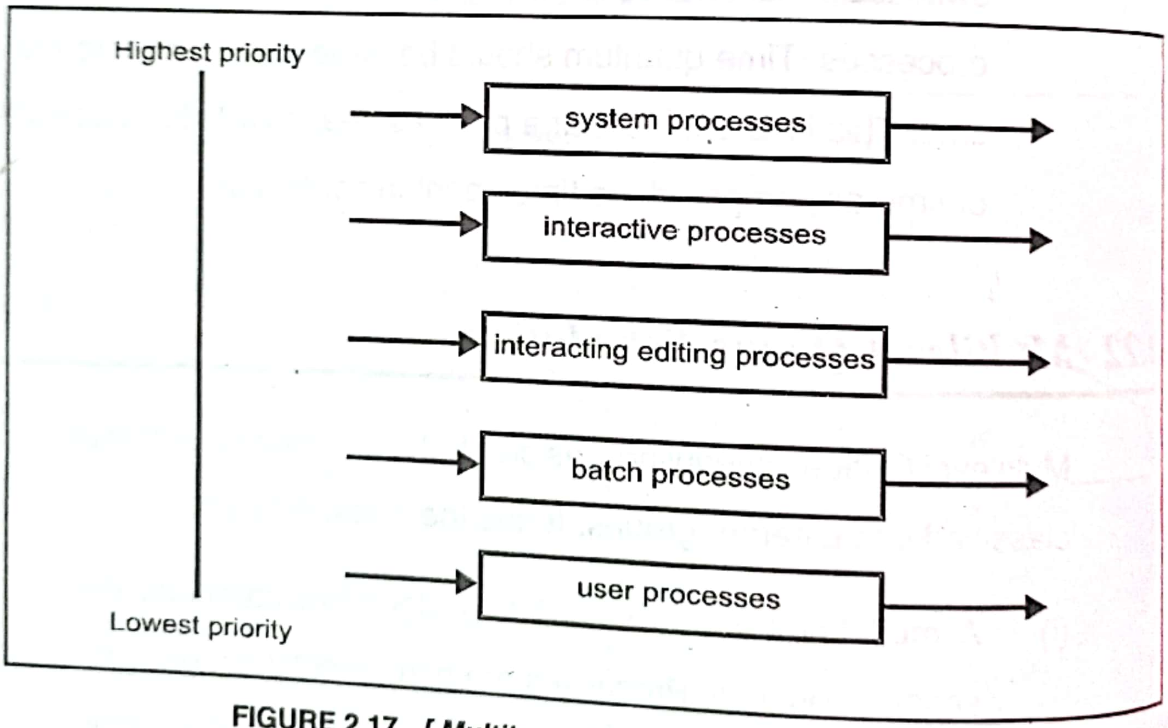


FIGURE 2.17 ***[Multilevel Queue Scheduling]***

Each queue has absolute priority over lower-priority queue. No process in batch queue can run before system process & so on.

Another possibility is to time slice between the queues. Each queue runs for a particular time slice.

Advantages

In a Multilevel Queue algorithm, processes are permanently assigned to a queue on entry to the system. Since processes do not change their interactive foreground or batch (background) nature, this set up has the advantage of low scheduling overhead.

Disadvantages

It is inflexible as the processes can never change their queues and thus may have to starve for the CPU if one or the other higher priority queues are never becoming empty.

2.23 *Multi-Level-Queues With Feedback Scheduling*

In a multiple-level-queue scheduling algorithm processes do not move between queues. Multi level feedback queue scheduling allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics.

If a process is using too much CPU time, it will be moved to a low-priority queue. This scheme leaves I/O bound and interactive processes in the higher-priority queues. Similarly, if a process waits too long in a lower priority queue, it may be moved to a higher-priority queue.

A technique called "**aging**" promotes lower processes to the next higher priority queue after a suitable interval of time.

How to define this algorithm?

The following things should be determined for the implementation of this type of algorithm :

- Number of queues
- Scheduling algorithm for each queue

- Method for upgrading a process to a higher-priority queue
- Method for downgrading a process to a lower-priority queue
- Method for assigning a process to a queue initially.

Example 6

Consider a multilevel feedback queue. Scheduler with three queuesm Q1, Q2 and Q3 as shown in fig. 2.18.

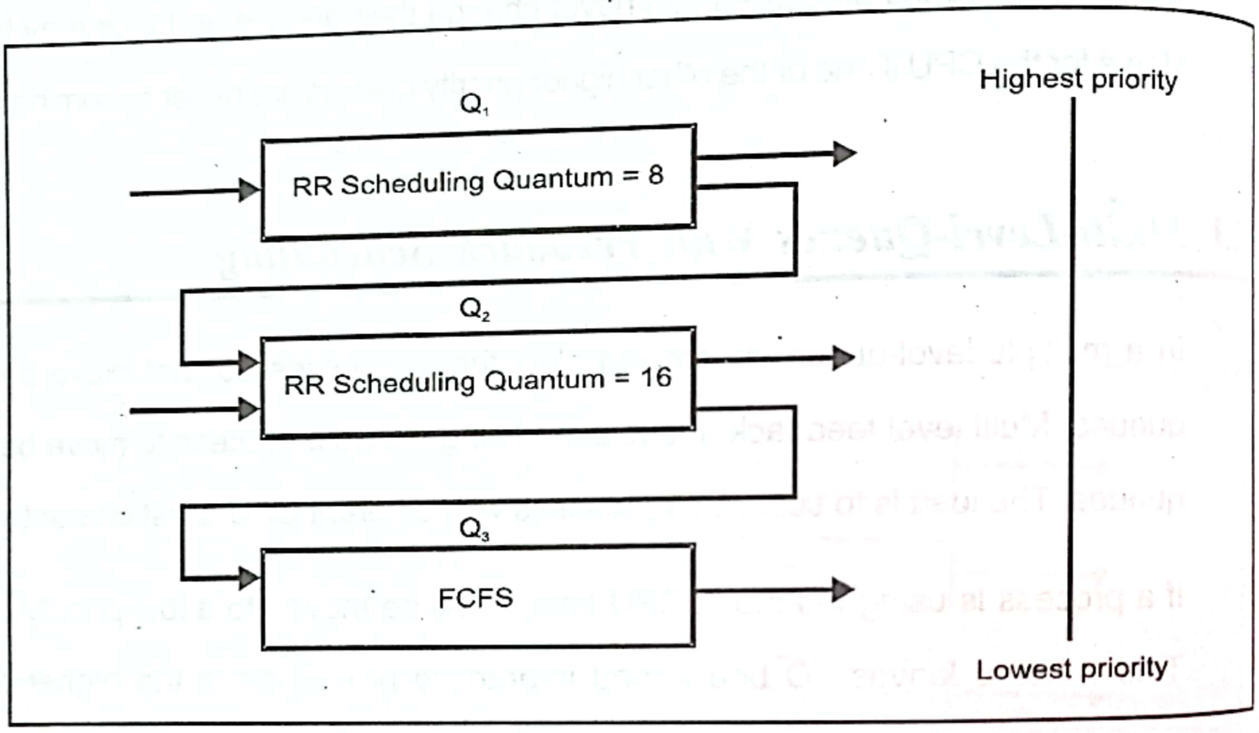


FIGURE 2.18

A process entering the ready queue is put in queue Q1. The scheduler first executes processes in Q1, which is given a time quantum of 8 milli-seconds. If a process does not finish within this time, it is moved to the tail of the Q2. The scheduler executes processes in Q2 only when Q1 is empty. The process at the head of Q2 is given a quantum of 16 milli-seconds. If it does not complete, it is preempted and is put into Q3. Processes in Q3 are run on an FCFS basis, only when Q1 and Q2 are empty.

A process that arrives in Q1 will preempt a process in Q2, a process that arrives in Q2 will preempt a process in Q3.

Consider the following set of processes with their CPU-burst times and arrival times.

Process	Arrival Time	Burst time
P1	0	17ms
P2	12	25ms
P3	28	8ms
P4	36	30ms

Process P1 first assigned to Q1. The scheduler first executes P1 in Q_1, which is given a time quantum of 8ms. But P1 burst time is 17ms, therefore, P1 is moved to the tail of the Q_2. Since Q_1 is empty, therefore scheduler executes P2 in Q_2. But at 12ms, new process P2 is assigned to Q1, therefore scheduler executes P2 in Q_1, which is also given a time quantum of 8ms. After exhausts the time quantum (8ms) of Q_1, the process P2 is moved to the tail of the Q_2 having proces P1. Now Q_1 is empty, therefore scheduler executes the process in Q2 as shown in Gantt Chart.

P1	P1	P2	P1	P2	P3	P4	P2	P4	P4
Q1	Q2	Q1	Q2	Q2	Q1	Q1	Q2	Q2	Q3

0 8 12 20 25 28 36 44 58 60 80

The waiting time for each process will be given below

Process	Waiting Time
P1	8ms
P2	21ms
P3	0ms
P4	14ms

$$\text{Average waiting time} = \frac{8+21+0+14}{4}$$
$$=10.75\text{m}$$