Women University Press
Jayoti Publication Desk
ISBN No. : 978-93-90892-25-9

# A SHORT GUIDE TOWARDS OPERATING SYSTEM

JV'n Muskan Kumari

JAYOTI VIDYAPEETH WOMEN'S UNIVERSITY, JAIPUR
UGC Approved Under 2(f) & 12(b) | NAAC Accredited | Recognized by Statutory Councils

Printed by :
JAYOTI PUBLICATION DESK

Published by :
Women University Press
Jayoti Vidyapeeth Women's University, Jaipur

Faculty of Education & Methodology

**Title:** A short guide towards operating system

**Author Name**Ms. Muskan

**Published By**: Women University Press

**Publisher's Address**: Jayoti Vidyapeeth Women's University, Jaipur
Vedaant Gyan Valley,
Village-Jharna, Mahala Jobner Link Road, NH-8
Jaipur Ajmer Express Way,
Jaipur-303122, Rajasthan (INDIA)

**Printer's Detail**: Jayoti Publication Desk

**Edition Detail**: I

**ISBN**: 978-93-90892-25-9

## Table of Contents

# CHAPTER 1
# Introduction to OS

## 1. Basic Introduction to Operating System

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

## 1.1 Operating System – Definition

- An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being application programs.
- An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information. The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.
- In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user.
- We need a system which can act as an intermediary and manage all the processes and resources present in the system.

The below figure1shows the operating system which creates the interface between hardware and application.

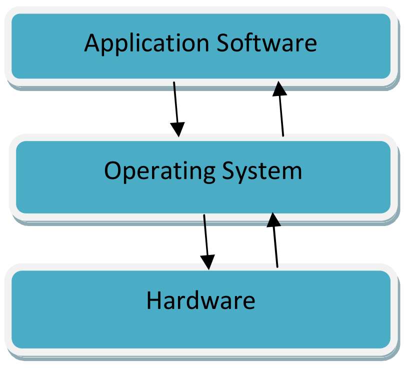


Figure1: Diagram of Operating System

- An **Operating System** can be defined as an **interface between user and hardware**. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.
- The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

## 1.2 Functions of Operating system -

Operating system performs three functions:

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions at the same time without interfering with service.

## 1.3 Operating system as User Interface –

1. User
2. System and application programs
3. Operating system
4. Hardware

Every general-purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and

I/O devices, peripheral device, and storage device. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs.

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work.

The operating system is a set of special programs that run on a computer system that allows it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling peripheral devices.

OS is designed to serve two basic purposes:

1. It controls the allocation and use of the computing System's resources among the various user and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs.

## 1.4 Operating system tasks

The tasks are given below:

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer's memory for execution.
4. Provide routines that handle the details of I/O programming.
5. I/O System Management – The module that keeps track of the status of devices is called the I/O traffic controller. Each I/O device has a device handler that

resides in a separate process associated with that device. A memory Management component that includes buffering caching and spooling.

6. A general device driver interface.

### 1.4.1 Drivers for specific hardware devices

**Assembler**

The input to an assembler is an assembly language program. The output is an object program plus information that enables the loader to prepare the object program for execution. At one time, the computer programmer had at his disposal a basic machine that interpreted, through hardware, certain fundamental instructions. He would program this computer by writing a series of ones and Zeros (Machine language), place them into the memory of the machine.

**Compiler**

The High-level languages- examples are FORTRAN, COBOL, ALGOL and PL/I are processed by compilers and interpreters. A compiler is a program that accepts a source program in a “high-level language “and produces a corresponding object program. An interpreter is a program that appears to execute a source program as if it was machine language. The same name (FORTRAN, COBOL, etc.) is often used to designate both a compiler and its associated language.

A Loader is a routine that loads an object program and prepares it for execution. There are various loading schemes: absolute, relocating and direct-linking. In general, the loader must load, relocate and link the object program. The loader is a program that places programs into memory and prepares them for execution. In a simple loading scheme, the assembler outputs the machine language translation of a program on a secondary device and a loader places it in the core. The loader places into memory the machine language version of the user’s program and transfers control to it. Since the loader program is much smaller than the assembler, those make more cores available to the user’s program.

## 1.5 History of Operating System

Operating system has been evolving through the years. Below figure shows the history of OS.

| Generation | Year | Electronic device used | Types of OS Device |
|---|---|---|---|
| First | 1945-55 | Vaccum Tubes | Plug Boards |
| Second | 1955-65 | Transistors | Batch Systems |
| Third | 1965-80 | Integrated Circuits(IC) | Multiprogramming |
| Fourth | Since 1980 | Large Scale Integration | PC |

Figure2 : history of Operating System

## 1.6 Applications of Operating System

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

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other software and users** − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.
- Operating System provides the basic and advanced concepts of operating system.
- Operating System can be defined as an interface between user and the hardware. It provides an environment to the user so that, the user can perform its task in convenient and efficient way.
- The Operating System is divided into various parts based on its functions such as Process Management, Process Synchronization, Deadlocks and File Management.

## 1.7 What does an Operating system do?

1. **Process Management** : A program does nothing except if their guidelines are executed by a CPU. A cycle is a program in execution. A period shared client program, for example, a complier is a cycle. A word preparing program being controlled by an individual client on a pc is a cycle. A framework assignment, for example, sending yield to a printer is likewise a cycle. A cycle needs certain assets including CPU time, memory records and I/O gadgets to achieve its undertaking. These assets are either given to the cycle when it is made or dispensed to it while it is running. The OS is liable for the accompanying exercises of cycle the board.

   • Creating and erasing both client and framework measures.

   • Suspending and continuing cycles.

   • Providing system for measure synchronization.

   • Providing system for measure correspondence.

   • Providing system for halt taking care of.

2. **Memory Management :** The primary memory is integral to the activity of a cutting edge PC framework. Fundamental memory is a huge cluster of words or bytes going in size from many thousand to billions. Principle memory stores the rapidly open information shared by the CPU and I/O gadget. The focal processor peruses guidance from fundamental memory during guidance get cycle and it the two peruses and composes information from principle memory during the information get cycle. The fundamental memory is by and large the just huge stockpiling gadget that the CPU can address and access straightforwardly. For instance, for the Central processor to deal with information from circle. That information should initially be moved to fundamental memory by CPU created E/O calls. Guidance should be in memory for the CPU to execute them. The OS is liable for the accompanying exercises regarding memory the executives.

   • Keeping track of which parts of memory are at present being utilized and by whom.

   • Deciding which cycles are to be stacked into memory when memory space becomes accessible.

• Allocating & de-allocating memory space varying.

3. **CPU Scheduling : In the uni-programmming systems** like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idol. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given.

   **In Multiprogramming systems**, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes.

4. **File Management :** File management is one of the most important components of an OS computer can store information on several different types of physical media magnetic tape, magnetic disk & optical disk are the most common media. Each medium is controlled by a device such as disk drive or tape drive those has unique characteristics. These characteristics include access speed, capacity, data transfer rate & access method (sequential or random).For convenient use of computer system the OS provides a uniform logical view of information storage. The OS abstracts from the physical properties of its storage devices to define a logical storage unit the file. A file is collection of related information defined by its creator.

   The OS is responsible for the following activities of file management.

   • Creating & deleting files.

   • Creating & deleting directories.

   • Supporting primitives for manipulating files & directories.

   • Mapping files into secondary storage.

   • Backing up files on non-volatile media

5. **Security :** If a computer system has multi users & allow the concurrent execution of multiple processes then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that files, memory

segments, CPU & other resources can be operated on by only those processes that have gained proper authorization from the OS.

## 1.8 Types of Operating System

- **Batch Operating System-** Sequence of jobs in a program on a computer without manual interventions.
- **Time sharing operating System-** allows many users to share the computer resources (Max utilization of the resources).
- **Distributed operating System-** Manages a group of different computers and makes appear to be a single computer.
- **Network operating system-** computers running in different operating system can participate in common network (It is used for security purpose).
- **Real time operating system** – meant applications to fix the deadlines.

Examples of Operating System are –

- Windows (GUI based, PC)
- GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
- macOS (Macintosh), used for Apple's personal computers and work stations (MacBook, iMac).
- Android (Google's Operating System for smart phones/tablets/smart watches)
- iOS (Apple's OS for iPhone, iPad and iPod Touch)

# CHAPTER 2
# Process Management & CPU Scheduling

## 2.1 Introduction to process management

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

### 2.1.1 Process

A process or task is an instance of a program in execution. The execution of a process must programs in a sequential manner. At any time at most one instruction is executed. The process includes the current activity as represented by the value of the program counter and the content of the processors registers. Also it includes the process stack which contain temporary data (such as method parameters return address and local variables) & a data section which contain global variables.

### 2.1.2 Difference between process & program

A program by itself is not a process. A program in execution is known as a process. A program is a passive entity, such as the contents of a file stored on disk whereas process is an active entity with a program counter specifying the next instruction to execute and a set of associated resources may be shared among several process with some scheduling algorithm being used to determinate when the stop work on one process and service a different one.

## 2.2 Process state

As a process executes, it changes state. The state of a process is defined by the correct activity of that process.

Each process may be in one of the following states.

• New: The process is being created.

• Ready: The process is waiting to be assigned to a processor.

• Running: Instructions are being executed.

• Waiting: The process is waiting for some event to occur.

• Terminated: The process has finished execution. Many processes may be in ready and waiting state at the same time. But only one process can be running on any processor at any instant.

## 2.3 Process Control Block

Each process is represented in the operating system by a process control block (PCB)—also called a task control block.

- Process state - The state may be new, ready, running, and waiting, halted, and so on.
- Program counter-The counter indicates the address of the next instruction to be executed for this process.
- CPU registers- The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general purpose registers, plus any condition code information.
- CPU-scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- Memory-management information- This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system
- Accounting information-This information includes the amount of CPU and real time used, time limits, account members, job or process numbers, and so on.
- I/O status information-This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

## 2.4 Process scheduling

The process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU. As processes enter the system, they are put into a job queue, which consists of all processes in the system.

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. 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 includes a pointer field that points to the next PCB in the ready queue.

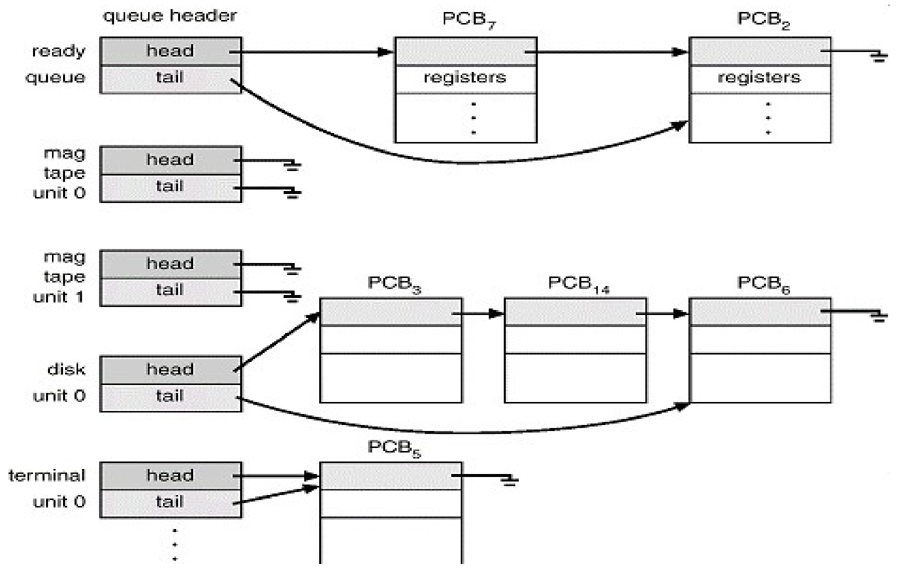


Figure 3 : The ready queue and various I/O device queue

In the above figure3 shows that each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system. A new process is initially put in the ready queue. It waits there till it is selected for execution, or is dispatched.

Once the process is allocated the CPU and is executing, one of several events could occur:

• The process could issue an I/O request and then be placed in an I/O queue.

• The process could create a new sub process and wait for the sub process's termination.

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

### 2.4.1 Working of Schedulers

A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler -

The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution.

The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them. The below figure4 shows the working of process with scheduling in queue.

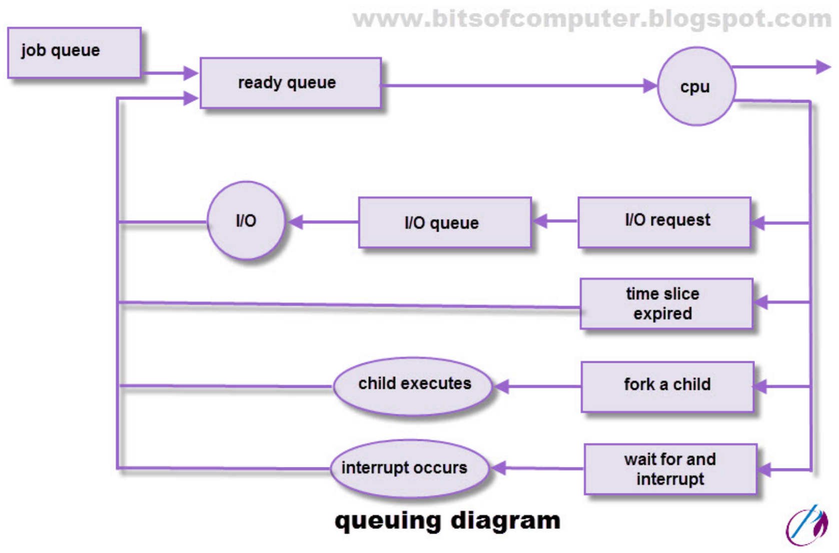


Figure 4: Queuing diagram representation of process scheduling

## 2.5 Operations on Processes

Process Creation A process may create several new processes, via a create-process system call, during the course of execution. The making cycle is known as a parent cycle, and the new cycles are known as the offspring of that cycle. Every one of these new cycles may thusly make different cycles, framing a tree of cycles. Most working frameworks recognize measures as indicated by a one of a kind process identifier (or pid), which is normally a number. These processes are liable for overseeing memory and document frameworks. The sched process additionally makes the init process, which fills in as the root parent measure for all client measures.

At the point when a cycle makes another cycle, two prospects exist regarding execution:

1. The parent keeps on executing simultaneously with its youngsters.
2. The parent holds up until a few or the entirety of its kids has ended.

There are likewise two prospects regarding the location space of the new cycle:

1. The kid cycle is a copy of the parent cycle (it has a similar program and information as the parent).
2. The youngster cycle has another program stacked into it.

```
#include <sys/types.h>
#include <stdio.h>
 #include <unistd.h>
 int main()
{
pid-t pid; /* fork a child process */
pid = fork();
 if (pid < 0)
{
/* error occurred */
 fprintf(stderr, "Fork Failed");
exit (-1) ;
}
 else if (pid == 0)
}
{
/* child process */
xeclpf"/bin/Is","Is",NULL);
```

```
}
else
{
/* parent process */
 /* parent will wait for the child to complete */
wait(NULL);
 printf("Child Complete");
 exit (0) ;
 }
}
```

In UNIX, as we've seen, each cycle is recognized by its cycle identifier, which is an interesting number. Another cycle is made by the fork() framework call. The new cycle comprises of a duplicate of the location space of the first cycle. This system permits the parent cycle to discuss effectively with its youngster cycle. The two cycles (the parent and the kid) proceed with execution at the guidance after the f ork(), with one distinction: The return code for the fork() is zero for the new (kid) measure, while the (nonzero) measure identifier of the youngster is gotten back to the parent. The executive() framework call is utilized after a fork() framework call by one of the two cycles to supplant the cycle's memory space with another program. The executive () framework consider loads a parallel record into memory (destroying the memory picture of the program containing the exe() framework call) and starts its execution. The below figure5 shows that how process creates.

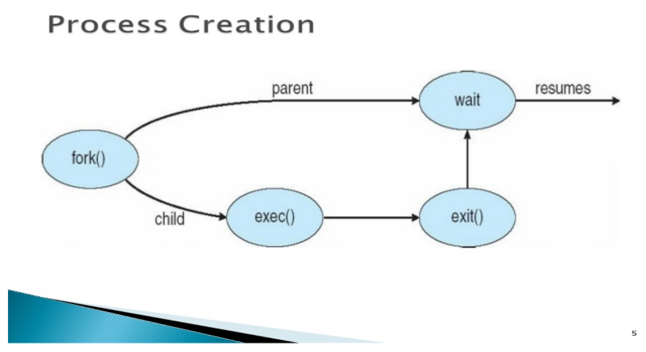


Figure 5 : Process Creation

**Process Termination**

A cycle ends when it wraps up executing its last assertion and requests that the working framework erase it by utilizing the exit () framework call. By then, the cycle may restore a status esteem (commonly a number) to its parent cycle (through the stand by() framework call). All the assets of the cycle—including physical and virtual memory, open records, and I/O cradles—are bargain situated by the working framework. End can happen in different conditions too. A cycle can cause the end of another cycle by means of a proper framework call (for instance, Terminate Proces() in Win32). Typically, such a framework call can be summoned exclusively by the parent of the cycle that will be ended. A parent may end the execution of one of its kids for an assortment of reasons, for example, these:

- The youngster has surpassed its use of a portion of the assets that it has been distributed.
- The errand appointed to the youngster is not, at this point required.
- The parent is leaving, and the working framework doesn't permit a youngster to proceed if its parent ends.

## 2.6 Inter-Process Communication

Cycles executing simultaneously in the working framework might be either free cycles or coordinating cycles. A cycle is free in the event that it can't influence or be influenced by different cycles executing in the framework. Any cycle that doesn't impart information to some other cycle is autonomous. A cycle is coordinating on the off chance that it can influence or be influenced by different cycles executing in the framework. There are a few explanations behind giving a climate that permits cycle participation:

- Information sharing. Since a few clients might be keen on a similar snippet of data (for example, a shared record), we should give a climate to permit simultaneous admittance to such data.
- Computation speedup. On the off chance that we need a specific undertaking to run quicker, we should break it into subtasks, every one of which will execute in corresponding with the others. Notice that such a speedup can be accomplished just if the PC has numerous preparing components, (for example, CPUs or I/O channels).

- Modularity. We might need to build the framework in a particular style, isolating the framework capacities into discrete cycles or strings.
- Convenience. Indeed, even an individual client may deal with numerous assignments simultaneously.

For example, a client might be altering, printing, and incorporating in equal. Coordinating cycles require an inter process correspondence (IPC) instrument that will permit them to trade information and data. There are two key models of interprocess correspondence:

(1) shared memory and (2) message passing.

In the shared-memory model, an area of memory that is shared by collaborating measures is set up. Cycles would then be able to trade data by perusing and composing information to the shared district. In the message passing model, correspondence happens by methods for messages traded between the participating cycles.

Message passing is valuable for trading more modest measures of information, on the grounds that no contentions need be kept away from. Message passing is additionally simpler to actualize than is shared memory for intercomputer correspondence. Shared memory permits greatest speed and comfort of correspondence, as it very well may be done at memory speeds when inside a PC. Shared memory is quicker than message passing, as message-passing frameworks are ordinarily actualized utilizing framework calls and along these lines require the additional tedious errand of portion mediation.

### 2.6.1 IPC is best provided by a message-passing system

The capacity of a message framework is to permit cycles to speak with each other without the need to turn to shared information.

An IPC office gives in any event the two tasks:

1. send (message)
2. receive (message).

Messages sent by a cycle can be of either fixed or variable size.

On the off chance that lone fixed-sized messages can be sent, the framework level usage is direct. Nonetheless, makes the assignment of programming more troublesome.

In the event that it is variable-sized messages, at that point it require a more mind boggling framework level usage, yet the programming task gets less difficult.

In the event that measures P and Q need to impart, they should send messages to and get messages from one another, a correspondence link((such as shared memory, equipment transport, or organization) should exist between them.

Here are a few techniques for legitimately actualizing a connection and the send/get activities:

1. Immediate or aberrant correspondence
2. Symmetric or hilter kilter correspondence
3. Programmed or express buffering
4. Send by duplicate or send by reference
5. Fixed-sized or variable-sized messages

### 2.6.2 Communication between Process by referring each other

#### 2.6.2.1 Direct Communication

With direct correspondence, each cycle that needs to impart should expressly name the beneficiary or sender of the correspondence.

This plan displays evenness in tending to. In this plan, the send and get natives are characterized as:

Send(P, message): Send a message to handle P.

get (Q, message)- Receive a message from measure Q.

A correspondence interface in this plan has the accompanying properties:

- A connection is set up consequently between each pair of cycles that need to convey.
- The cycles need to realize just each other's personality to impart.

- A connection is related with precisely two cycles. Precisely one connection exists between each pair of cycles.

In unevenness just the sender names the beneficiary; the beneficiary isn't needed to name the sender. In this plan, the send and get natives are characterized as follows:

Send(P, message): Send a message to handle P.

receive(id, message):Receive a message from any cycle; the variable id is set to the name of the cycle with which correspondence has occurred.

**Disadvantage of symmetric and asymmetric plans**

The burden in both symmetric and awry plans is the restricted particularity of the subsequent cycle definitions. Changing the name of a cycle may require inspecting any remaining cycle definitions.

#### 2.6.2.2 Indirect Communication

With indirect communication, the messages are shipped off and gotten from letter boxes, or ports.

A letter drop can be seen conceptually as an item into which messages can be set by measures and from which messages can be taken out. Every letter drop has an interesting ID. In this plan, a cycle can speak with some other cycle through various distinctive letter drops.

Two cycles can impart just on the off chance that they share a letter drop.

The send and get natives are characterized as follows:

send(A, message): Send a message to letter box A.

receive(A, message): Receive a message from post box A.

In this plan, a correspondence interface has the accompanying properties:

- A connection is set up between a couple of cycles in particular if the two individuals from the pair have a shared post box.
- A connection might be related with multiple cycles.
- Various connections may exist between each pair of discussing measures, with each connection comparing to one post box.

Presently guess that measures P1, P2, and P3 all offer letter drop A. Cycle P1 makes an impression on, some time P2 and P3 each execute a get from A. Which cycle will get the message sent by P1 ? The appropriate response relies upon the plan that we pick:

- Allow a connect to be related with all things considered two cycles.
- Allow all things considered each cycle in turn to execute a get activity.
- Allow the framework to choose discretionarily which cycle will get the message (that is, either P2 or P3, however not both, will get the message).

The framework may recognize the beneficiary to the sender.

A letter box might be claimed either by a cycle or by the working framework.

## 2.7 Synchronization

Message passing may be either blocking or non-blocking also known as synchronous and asynchronous.

**Blocking send:**

The sending process is blocked until the message is received by the receiving process or by the mailbox.

**Non-blocking send:**

The sending process sends the message and resumes operation.

**Blocking receive**:

The receiver blocks until a message is available.

**Non-blocking receive:**

The receiver retrieves either a valid message or a null. When both the send and receive are blocking, we have a rendezvous between the sender and the receiver.

### 2.7.1 Buffering

Regardless of whether the correspondence is immediate or backhanded, messages traded by imparting measures dwell in an impermanent line. Such a line can be executed thereby :

Zero limits: The line has most extreme length 0; hence, the connection can't have any messages holding up in it. For this situation, the sender should obstruct until the beneficiary gets the message.

The zero-limit case is now and again alluded to as a message framework with no buffering;

Limited limit: The line has limited length n subsequently; at most n messages can live in it. On the off chance that the line isn't full when another message is sent, the last is set in the line (either the message is duplicated or a pointer to the message is kept), and the sender can proceed with execution without pausing. The connection has a limited limit, in any case. On the off chance that the connection is full, the sender should obstruct until space is accessible in the line. It is otherwise called programmed buffering.

Unbounded limit: The line has possibly boundless length; subsequently, quite a few messages can stand by in it. The sender won't ever impede. It is otherwise called programmed buffering.

## 2.8 CPU Scheduling

"CPU scheduling is a process of determining which process will own CPU to run while the execution of another process is on hold, which means in waiting state due to unavailability of any resource like I/O etc., thereby maximizing utilization of CPU."

### 2.8.1 Objectives of process scheduling algorithm are

- Keep CPU as occupied as could be expected under the circumstances
- Reasonable portion of CPU
- Max throughput: Number of cycles that total their execution per time unit
- Min turnaround time: Time taken by a cycle to complete execution
- Min holding up time: Time a cycle holds up in prepared line
- Min reaction time: Time when a cycle creates first reaction

### 2.8.2 Central processor Scheduler –

- Selects from among the cycles in memory that are prepared to execute, and distributes the CPU to one of them
- CPU planning choices may happen when a cycle:

1. Changes from hurrying to holding up state
2. Changes from rushing to prepared state
3. Changes from holding on to prepared
4. Ends

- Scheduling under 1 and 4 is non preemptive
- All other booking is pre-emptive

### 2.8.3 Dispatcher

Dispatcher module gives control of the CPU to the cycle chose by the momentary scheduler; this includes:

- exchanging setting
- changing to client mode
- leaping to the legitimate area in the client program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one cycle and start another running

### 2.8.4 Scheduling Criteria

- CPU use – keep the CPU as occupied as could be expected under the circumstances
- Throughput – No. of cycles that total their execution per time unit
- Turnaround time – measure of time to execute a specific cycle
- Waiting time – measure of time a cycle has been holding up in the prepared line
- Response time – measure of time it takes from when a solicitation was submitted until the principal reaction is created, not yield (for time-sharing climate)

### 2.8.5 Optimization Criteria

- Max CPU use
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

## 2.9 Scheduling Algorithms

There are the following algorithms which can be used to schedule the jobs.

### 2.9.1. First Come First Serve

It is the least difficult calculation to execute. The cycle with the negligible appearance time will get the CPU first. The lesser the appearance time, the sooner will the cycle gets the CPU. It is the non-preemptive kind of scheduling.

**Advantages of FCFS**

- Straight forward
- Simple
- First come first serve based algorithm

**Disadvantages of FCFS**

- The planning strategy is non preemptive, the cycle will race to the finishing.
- Because of the non-preemptive nature of the calculation, the issue of starvation may happen.
- In spite of the fact that it is not difficult to actualize, yet it is poor in execution since the normal holding up time is higher as contrast with other booking calculations.

**Example**

We should take an illustration of The FCFS planning calculation. In the Following timetable, there are 5 cycles with measure ID P0, P1, P2, P3 and P4. P0 shows up at time 0, P1 at time 1, P2 at time 2, P3 shows up at time 3 and Process P4 shows up at time 4 in the prepared line. The cycles and their particular Arrival and Burst time are given in the accompanying table.

The Turnaround time and the holding up time are determined by utilizing the accompanying formula.

Turn Around Time = Completion Time - Arrival Time

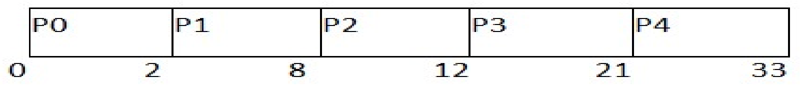
Waiting Time = Turnaround time - Burst Time

The normal holding up Time is dictated by adding the separate holding up season of the multitude of cycles and isolated the whole by the complete number of cycles.

| Process ID | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 0 | 0 | 2 | 2 | 2 | 0 |
| 1 | 1 | 6 | 8 | 7 | 1 |
| 2 | 2 | 4 | 12 | 8 | 4 |
| 3 | 3 | 9 | 21 | 18 | 9 |
| 4 | 4 | 12 | 33 | 29 | 17 |

Avg Waiting Time=31/5

**(Gantt chart)**



## 2.9.2 Shortest Job First (SJF)

Till now, we were planning the cycles as indicated by their appearance time (in FCFS booking). In any case, SJF booking calculation plans the cycles as indicated by their burst time.

In SJF planning, the cycle with the most reduced burst time, among the rundown of accessible cycles in the prepared line, will be booked straightaway.

In any case, it is hard to foresee the blasted time required for a cycle consequently this calculation is hard to actualize in the framework.

**Advantage of SJF**

- Most extreme throughput
- Least normal pausing and turnaround time

**Disadvantage of SJF**

- May endure with the issue of starvation
- It isn't implementable on the grounds that the specific Burst time for a cycle can't be known ahead of time.

There are various procedures accessible by which, the CPU burst season of the cycle can be resolved. We will talk about them later in detail.

**Example:**

In the accompanying model, there are five positions named as P1, P2, P3, P4 and P5. Their appearance time and burst time are given in the table below. Since, No Process shows up at time 0 thus; there will be a vacant space in the Gantt diagram from time 0 to 1 (the time at which the principal cycle shows up).

As per the calculation, the OS plans the cycle which is having the most reduced burst time among the accessible cycles in the prepared line.

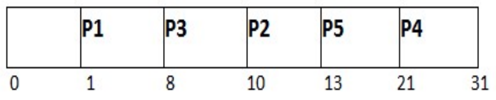
Till now, we have just one cycle in the prepared line thus the scheduler will plan this to the processor regardless of what is its blasted time.

This will be executed till 8 units of time. Till then we have three additional cycles showed up in the prepared line consequently the scheduler will pick the cycle with the most minimal burst time.

Among the cycles given in the table, P3 will be executed next since it is having the most reduced burst time among all the accessible cycles.

So that is the way the strategy will go on in most limited occupation first (SJF) booking calculation.

| PID | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 1 | 7 | 8 | 7 | 0 |
| 2 | 3 | 3 | 13 | 10 | 7 |
| 3 | 6 | 2 | 10 | 4 | 2 |
| 4 | 7 | 10 | 31 | 24 | 14 |
| 5 | 9 | 8 | 21 | 12 | 4 |



Avg Waiting Time = 27/5

**Shortest Remaining Time First (SRTF) Scheduling Algorithm**

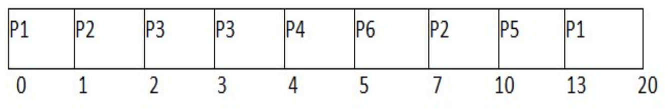
This Algorithm is the preemptive variant of SJF planning. In SRTF, the execution of the cycle can be halted after certain measure of time. At the appearance of each cycle, the momentary scheduler plans the cycle with the most un-residual burst time among the rundown of accessible cycles and the running cycle.

When all the cycles are free in the prepared line, No acquisition will be done and the calculation will fill in as SJF planning. The setting of the cycle is saved in the Process Control Block when the cycle is eliminated from the execution and the following cycle is booked. This PCB is gotten to on the following execution of this cycle.

**Example**

In this Example, there are five positions P1, P2, P3, P4, P5 and P6. Their appearance time and burst time are given beneath in the table.

| Process ID | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time | Response Time |
|---|---|---|---|---|---|---|
| 1 | 0 | 8 | 20 | 20 | 12 | 0 |
| 2 | 1 | 4 | 10 | 9 | 5 | 1 |
| 3 | 2 | 2 | 4 | 2 | 0 | 2 |
| 4 | 3 | 1 | 5 | 2 | 1 | 4 |
| 5 | 4 | 3 | 13 | 9 | 6 | 10 |
| 6 | 5 | 2 | 7 | 2 | 0 | 5 |



Avg Waiting Time = 24/6

The Gantt outline is set up as indicated by the appearance and burst time given in the table.

1. Since, at time 0, the solitary accessible cycle is P1 with CPU blasted time 8. This is the lone accessible cycle in the rundown consequently it is planned.

2. The next process shows up at time unit 1. Since the calculation we are utilizing is SRTF which is a preemptive one, the current execution is halted and the scheduler checks for the cycle with the least blasted time.

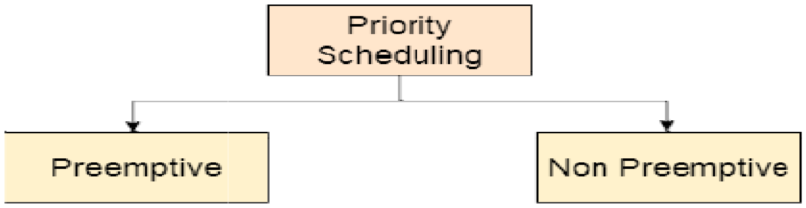
   Till now, there are two cycles accessible in the prepared line. The OS has executed P1 for one unit of time till now; the leftover burst season of P1 is 7 units. The burst season of Process P2 is 4 units. Henceforth Process P2 is planned on the CPU as indicated by the calculation.
3. The next process P3 shows up at time unit 2. Right now, the execution of cycle P3 is halted and the cycle with the most un-outstanding burst time is looked. Since the cycle P3 has 2 unit of burst time henceforth it will be given need over others.
4. The Next Process P4 shows up at time unit 3. At this appearance, the scheduler will stop the execution of P4 and check which cycle is having least burst time among the accessible cycles (P1, P2, P3 and P4). P1 and P2 are having the leftover burst time 7 units and 3 units individually.

   P3 and P4 are having the leftover burst time 1 unit each. Since, both are equivalent henceforth the planning will be finished by their appearance time. P3 shows up sooner than P4 and in this way it will be planned once more.
5. The Next Process P5 shows up at time unit 4. Till this time, the Process P3 has finished its execution and it is not any more in the rundown. The scheduler will look at the leftover burst season of the multitude of accessible cycles. Since the burst season of cycle P4 is 1 which is least among all thus this will be planned.
6. The Next Process P6 shows up at time unit 5, till this time, the Process P4 has finished its execution. We have 4 accessible cycles till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst season of P6 is the least among all consequently P6 is planned. Since, presently, all the cycles are accessible thus the calculation will currently work same as SJF. P6 will be executed till its consummation and afterward the cycle with the most un-residual time will be planned.

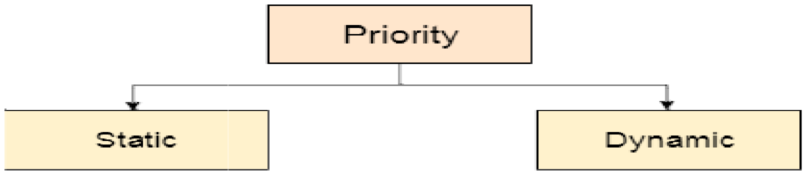
When all the cycles show up, No appropriation is done and the calculation will fill in as SJF.

### 2.9.3 Priority Scheduling

In Priority booking, there is a need number doled out to each cycle. In certain frameworks, the lower the number, the higher the need. While, in the others, the higher the number, the higher will be the need. The Process with the higher need among the accessible cycles is given the CPU. There are two sorts of need booking calculation exists. One is Preemptive need planning while the other is Non Preemptive Priority booking.



The need number relegated to every one of the cycle might differ. On the off chance that the need number doesn't change itself all through the cycle, it is called static need, while on the off chance that it continues to change itself at the customary spans, it is called dynamic need.



**Non Preemptive Priority Scheduling**

In the Non Preemptive Priority booking, The Processes are planned by the need number relegated to them. When the cycle gets booked, it will run till the fulfilment. By and large, the lower the need number, the higher is the need of the cycle.

**Example:**

In the Example, there are 7 cycles P1, P2, P3, P4, P5, P6 and P7. Their needs, Arrival Time and burst time are given in the table.

| Process ID | Priority | Arrival Time | Burst Time |
|---|---|---|---|
| 1 | 2 | 0 | 3 |
| 2 | 6 | 2 | 5 |
| 3 | 3 | 1 | 4 |
| 4 | 5 | 4 | 2 |
| 5 | 7 | 6 | 9 |
| 6 | 4 | 5 | 4 |
| 7 | 10 | 7 | 10 |

We can prepare the Gantt chart according to the Non Preemptive priority scheduling.

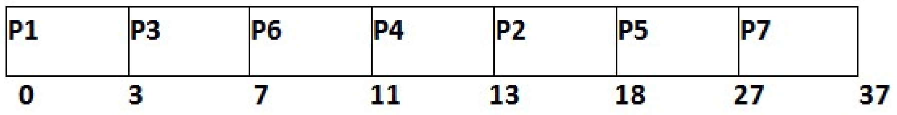
The Process P1 arrives at time 0 with the burst time of 3 units and the priority number 2. Since No other process has arrived till now hence the OS will schedule it immediately.

Meanwhile the execution of P1, two more Processes P2 and P3 are arrived. Since the priority of P3 is 3 hence the CPU will execute P3 over P2.

Meanwhile the execution of P3, All the processes get available in the ready queue. The Process with the lowest priority number will be given the priority. Since P6 has priority number assigned as 4 hence it will be executed just after P3.

After P6, P4 has the least priority number among the available processes; it will get executed for the whole burst time.

Since all the jobs are available in the ready queue hence All the Jobs will get executed according to their priorities. If two jobs have similar priority number assigned to them, the one with the least arrival time will be executed.



From the GANTT Chart prepared, we can determine the completion time of every process. The turnaround time, waiting time and response time will be determined.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

| Process Id | Priority | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time | Response Time |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 3 | 3 | 3 | 0 | 0 |
| 2 | 6 | 2 | 5 | 18 | 16 | 11 | 13 |
| 3 | 3 | 1 | 4 | 7 | 6 | 2 | 3 |
| 4 | 5 | 4 | 2 | 13 | 9 | 7 | 11 |
| 5 | 7 | 6 | 9 | 27 | 21 | 12 | 18 |
| 6 | 4 | 5 | 4 | 11 | 6 | 2 | 7 |
| 7 | 10 | 7 | 10 | 37 | 30 | 18 | 27 |

Avg Waiting Time = (0+11+2+7+12+2+18)/7 = 52/7 units

**Preemptive Priority Scheduling**

In Pre-emptive Priority Scheduling, at the hour of appearance of a cycle in the prepared line, its Priority is contrasted and the need of different cycles present in the prepared line just as with the one which is being executed by the CPU by then of time. The One with the most elevated need among all the accessible cycles will be given the CPU next.

The contrast between preemptive need planning and non preemptive need booking is that, in the preemptive need planning, the work which is being executed can be halted at the appearance of a higher need work.

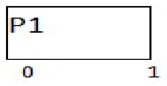
When all the positions prepare accessible in the line, the calculation will act as non-preemptive need booking, which implies the work planned will run till the finish and no appropriation will be done. Example

There are 7 cycles P1, P2, P3, P4, P5, P6 and P7 given. Their separate needs, Arrival Times and Burst times are given in the table beneath.

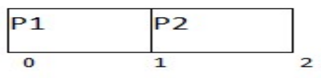
| Process Id | Priority | Arrival Time | Burst Time |
|---|---|---|---|
| 1 | 2(L) | 0 | 1 |
| 2 | 6 | 1 | 7 |
| 3 | 3 | 2 | 3 |
| 4 | 5 | 3 | 6 |
| 5 | 4 | 4 | 5 |
| 6 | 10(H) | 5 | 15 |
| 7 | 9 | 15 | 8 |

**GANTT chart Preparation**

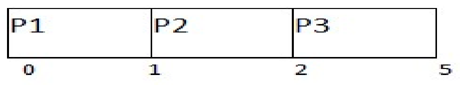
At time 0, P1 arrives with the burst time of 1 units and priority 2. Since no other process is available hence this will be scheduled till next job arrives or its completion (whichever is lesser).



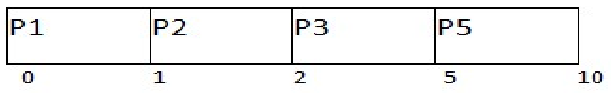
At time 1, P2 arrives. P1 has completed its execution and no other process is available at this time hence the Operating system has to schedule it regardless of the priority assigned to it.



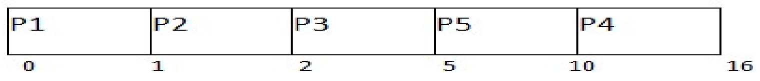
The Next process P3 arrives at time unit 2, the priority of P3 is higher to P2. Hence the execution of P2 will be stopped and P3 will be scheduled on the CPU.



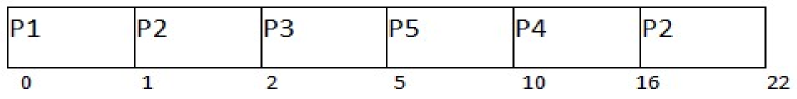
During the execution of P3, three more processes P4, P5 and P6 becomes available. Since, all these three have the priority lower to the process in execution so PS can't preempt the process. P3 will complete its execution and then P5 will be scheduled with the priority highest among the available processes.



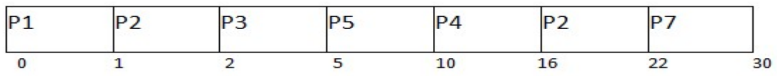
Meanwhile the execution of P5, all the processes got available in the ready queue. At this point, the algorithm will start behaving as Non Preemptive Priority Scheduling. Hence now, once all the processes get available in the ready queue, the OS just took the process with the highest priority and execute that process till completion. In this case, P4 will be scheduled and will be executed till the completion.



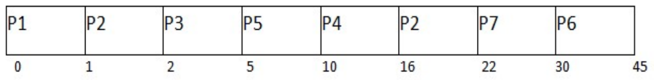
Since P4 is completed, the other process with the highest priority available in the ready queue is P2. Hence P2 will be scheduled next.



P2 is given the CPU till the completion. Since its remaining burst time is 6 units hence P7 will be scheduled after this.



The only remaining process is P6 with the least priority, the Operating System has no choice unless of executing it. This will be executed at the last.



The Completion Time of each process is determined with the help of GANTT chart. The turnaround time and the waiting time can be calculated by the following formula.

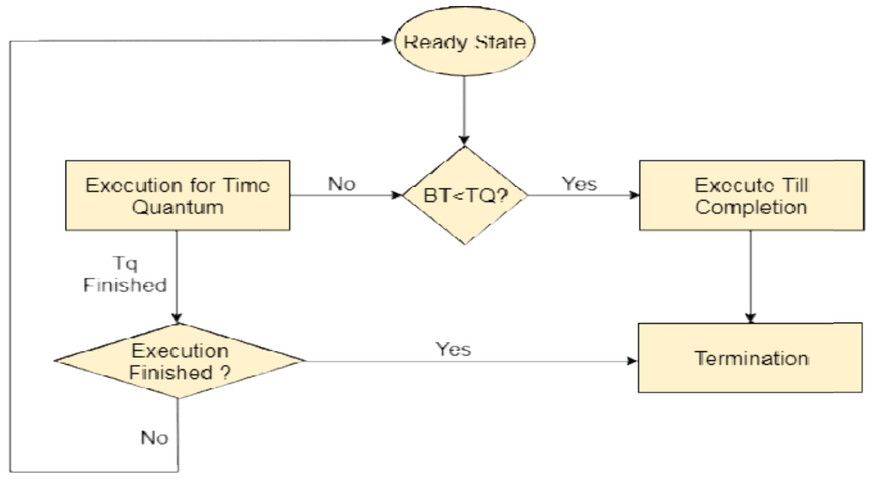
1. Turnaround Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

| Process Id | Priority | Arrival Time | Burst Time | Completion Time | Turn around Time | Waiting Time |
|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 1 | 1 | 1 | 0 |
| 2 | 6 | 1 | 7 | 22 | 21 | 14 |
| 3 | 3 | 2 | 3 | 5 | 3 | 0 |
| 4 | 5 | 3 | 6 | 16 | 13 | 7 |
| 5 | 4 | 4 | 5 | 10 | 6 | 1 |
| 6 | 10 | 5 | 15 | 45 | 40 | 25 |
| 7 | 9 | 6 | 8 | 30 | 24 | 16 |

Avg Waiting Time = (0+14+0+7+1+25+16)/7 = 63/7 = 9 units

### 2.9.4 Round Robin Scheduling Algorithm

Round Robin Scheduling Algorithm is quite possibly the most famous booking calculation which can really be actualized in the greater part of the working frameworks. This is the preemptive variant of the early bird gets the worm booking. The Algorithm centers around Time Sharing. In this calculation, each cycle gets executed in a cyclic way. A specific time cut is characterized in the framework which is called time quantum. Each cycle present in the prepared line is alloted the CPU for that time quantum, in the event that the execution of the cycle is finished during that time, at that point the cycle will end else the cycle will return to the prepared line and trusts that the following turn will finish the execution.



**Advantages**

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fare allocation of CPU.

**Disadvantages**

1. The higher the time quantum, the higher the response time in the system.
2. The lower the time quantum, the higher the context switching overhead in the system.
3. Deciding a perfect time quantum is really a very difficult task in the system.

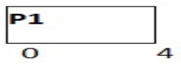
**RR Scheduling Example**

In the following example, there are six processes named as P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table. The time quantum of the system is 4 units.

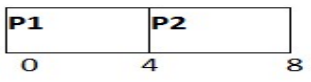
| Process ID | Arrival Time | Burst Time |
|---|---|---|
| 1 | 0 | 5 |
| 2 | 1 | 6 |
| 3 | 2 | 3 |
| 4 | 3 | 1 |
| 5 | 4 | 5 |
| 6 | 6 | 4 |

This is solved through step by step:

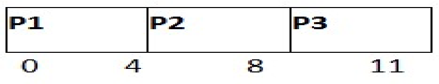
Step1



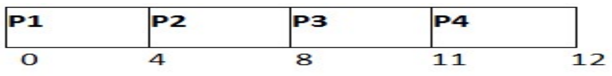
Step2



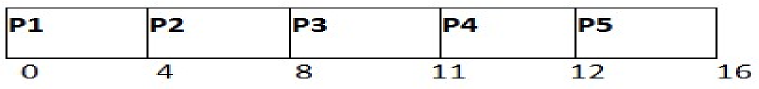
Step3



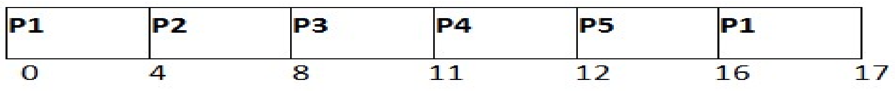
Step4



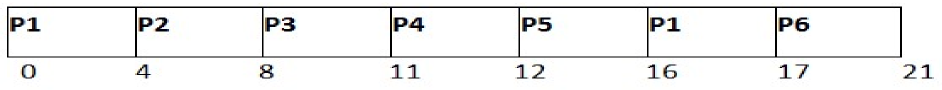
Step5



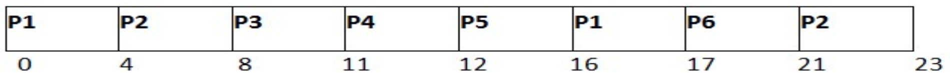
Step6



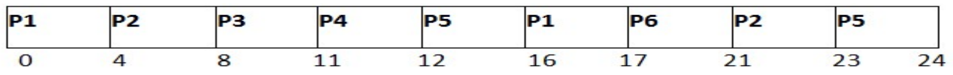
Step7



Step8



Step9



### 2.9.5 Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

### 2.9.6 Multilevel Feedback Queue Scheduling

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process

    - method used to determine which queue a process will enter when that process needs service

## 2.10 Process Synchronization

- Concurrent access to shared data may result in data inconsistency (change in behavior)
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Suppose that we wanted to provide a solution to the "producer-consumer" problem that fills all the buffers.
- We can do so by having an integer variable "count" that keeps track of the number of full buffers.
- Initially, count is set to 0.
- It is incremented by the producer after it produces a new buffer.
- It is decremented by the consumer after it consumes a buffer.

Producer

```
while (true)
{
/*  produce an item and put in next Produced  */
while (count == BUFFER_SIZE)
            ; // do nothing
  buffer [in] = next Produced;
   in = (in + 1) % BUFFER_SIZE;
    count++;
}
```

Consumer

```
while (true)
  {
while (count == 0)
```

```
    ; // do nothing
  next Consumed =  buffer[out];
      out = (out + 1) % BUFFER_SIZE;
              count--;
   /*  consume the item in next Consumed
  }
```

### 2.10.1 Critical section problem

A section of code which reads or writes shared data.

**Race Condition**

- The situation where two or more processes try to access and manipulate the same data and output of the process depends on the orderly execution of those processes is called as Race Condition.
- count++ could be implemented as
  - register1 = count
  - register1 = register1 + 1
  - count = register1
- count-- could be implemented as
  - register2 = count
  - register2 = register2 - 1
  - count = register2
- Consider this execution interleaving with "count = 5" initially:
  - S0: producer execute register1 = count {register1 = 5}
  - S1: producer execute register1 = register1 + 1 {register1 = 6}
  - S2: consumer execute register2 = count {register2 = 5}
  - S3: consumer execute register2 = register2 - 1 {register2 = 4}
  - S4: producer execute count = register1 {count = 6 }
  - S5: consumer execute count = register2 {count = 4}

### 2.10.2 Requirements for the Solution to Critical-Section Problem

1. **Mutual Exclusion**: - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections
2. **Progress:** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting: -** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

- To general approaches are used to handle critical sections in operating systems: (1) Preemptive Kernel (2) Non Preemptive Kernel
  - Preemptive Kernel allows a process to be preempted while it is running in kernel mode.
  - Non Preemptive Kernel does not allow a process running in kernel mode to be preempted. (these are free from race conditions)

### 2.10.3 Peterson's Solution

- It is restricted to two processes that alternates the execution between their critical and remainder sections.
- Assume that the LOAD and STORE instructions are atomic; that is, cannot be interrupted.
- The two processes share two variables: – int turn; – Boolean flag[2]
- The variable turn indicates whose turn it is to enter the critical section.
- The flag array is used to indicate if a process is ready to enter the critical section. flag[i] = true implies that process Pi is ready!

Note:- Peterson's Solution is a software based solution.

**Algorithm for Process Pi**

```
while (true) {
flag[i] = TRUE;
      turn = j;
```

```
  while ( flag[j] && turn == j);
             CRITICAL SECTION
  flag[i] = FALSE;
           REMAINDER SECTION
    }
```

**Solution to Critical Section Problem using Locks.**

```
  Do
{
acquire lock
    Critical Section
                 release lock
Remainder Section
          }while (True);
```

Note:- Race Conditions are prevented by protecting the critical region by the locks.

**2.10.4 Synchronization Hardware**

- In general we can provide any solution to critical section problem by using a simple tool called as LOCK where we can prevent the race condition.
- Many systems provide hardware support (hardware instructions available on several systems) for critical section code.
- In Uni Processor hardware environment by disabling interrupts we can solve the critical section problem. So that Currently running code would execute without any preemption .
- But by disabling interrupts on multiprocessor systems is time taking so that it is inefficient compared to Uni Processor system.
- Now a days Modern machines provide special atomic hardware instructions that allow us to either test memory word and set value Or swap contents of two memory words automatically i.e. done through an uninterruptible unit.

**Special Atomic hardware Instructions**

- Test And Set ()
- Swap ()
- The Test And Set () Instruction is one kind of special atomic hardware instruction that allow us to test memory and set the value. We can provide Mutual Exclusion by using Test And Set () instruction.
- Definition:

```
Boolean Test And Set (Boolean *target)
 {
     Boolean rv = *target;
        *target = TRUE;
   return rv:
    }
```

- Mutual Exclusion Implementation with TestAndSet()
- To implement Mutual Exclusion using TestAndSet() we need to declare Shared Boolean variable called as 'lock' ( initialized to false ) .
- Solution:

```
while (true) {
     while ( TestAndSet (&lock ))
                 ;   /* do nothing
         //   critical section
      lock = FALSE;
       //     remainder section
 }
```

- The Swap() Instruction is another kind of special atomic hardware instruction that allow us to swap the contents of two memory words.
- By using Swap() Instruction we can provide Mutual Exclusion.
- Definition:-
- void Swap (Boolean *a, Boolean *b)
- {
- Boolean temp = *a;

```
            *a = *b;
        *b = temp:
}
```

- Shared Boolean variable called as ‘lock’ is to be declared to implement Mutual Exclusion in Swap() also, which is initialized to FALSE.
- Solution:

```
    while (true)  {
        key = TRUE;
       while ( key == TRUE)
                    Swap (&lock, &key );
                            //    critical section
        lock = FALSE;
         //      remainder section
     }
```

Note:- Each process has a local Boolean variable called as ‘key’.

# CHAPTER 3
# MEMORY MANAGEMENT

## 3.1 Basic Introduction to Memory Management

Memory the board is worried about dealing with the essential memory. Memory comprises of exhibit of bytes or words each with their own location. The directions are gotten from the memory by the CPU dependent on the worth program counter.

### 3.1.1 Functions of memory management

- Keeping track of status of each memory location.
- Determining the allocation policy.
- Memory allocation technique.
- De-allocation technique.

### 3.1.2 Need for Multi programming in Memory Management

However, The CPU can directly access the main memory, Registers and cache of the system. The program always executes in main memory. The size of main memory affects degree of Multi programming to most of the extant. If the size of the main memory is larger than CPU can load more processes in the main memory at the same time and therefore will increase degree of Multi programming as well as CPU utilization.

Let's consider,

1. Process Size = 4 MB
2. Main memory size = 4 MB
3. The process can only reside in the main memory at any time.
4. If the time for which the process does IO is P,
5. 
6. Then,
7. 
8. CPU utilization = (1-P)

9. let's say,
10. P = 70%
11. CPU utilization = 30 %
12. Now, increase the memory size, Let's say it is 8 MB.
13. Process Size = 4 MB
14. Two processes can reside in the main memory at the same time.
15. Let's say the time for which, one process does its IO is P,
16.
17. Then
18.
19. CPU utilization = (1-P^2)
20. let's say P = 70 %
21. CPU utilization = (1-0.49) =0.51 = 51 %

Therefore, we can state that the CPU utilization will be increased if the memory size gets increased.

## 3.2 Partitioning of Memory Management

### 3.2.1 Fixed Partitioning

The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

**1. Internal Fragmentation**

If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

**2. External Fragmentation**

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

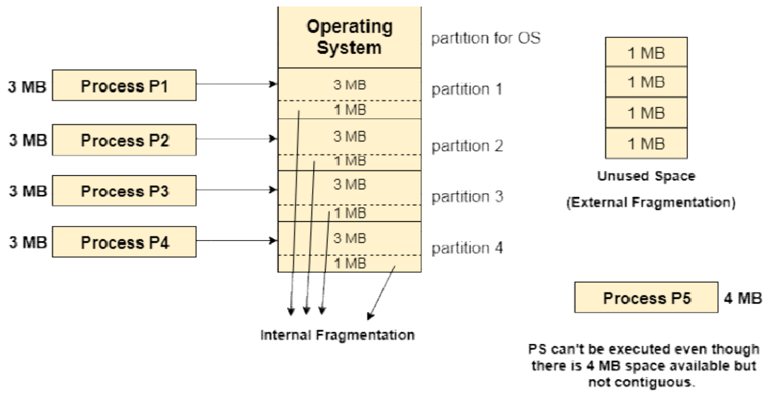
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

**3. Limitation on the size of the process**

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

**4. Degree of multiprogramming is less**

By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.



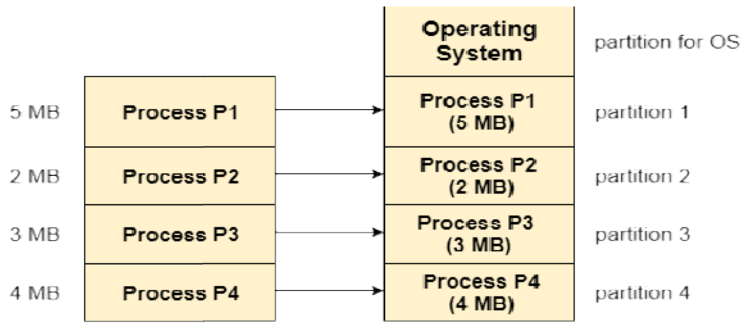
Fixed Partitioning

(Contiguous memory allocation)

### 3.2.2 Dynamic Partitioning

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be avoided.



**Dynamic Partitioning**
(Process Size = Partition Size)

**Advantages of Dynamic Partitioning over fixed partitioning**

**1. No Internal Fragmentation**

Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

**2. No Limitation on the size of the process**

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

**3. Degree of multiprogramming is dynamic**

Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

Disadvantages of dynamic partitioning

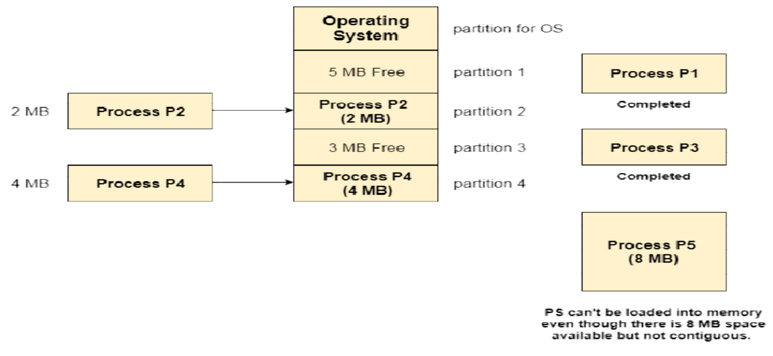
External Fragmentation

Absence of internal fragmentation doesn't mean that there will not be external fragmentation.

Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory.

After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.

The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.



External Fragmentation in Dynamic Partitioning

## 3.3 Complex Memory Allocation

In Fixed partitioning, the list of partitions is made once and will never change but in dynamic partitioning, the allocation and deallocation is very complex since the partition size will be varied every time when it is assigned to a new process. OS has to keep track of all the partitions.

Due to the fact that the allocation and deallocation are done very frequently in dynamic memory allocation and the partition size will be changed at each time, it is going to be very difficult for OS to manage everything.

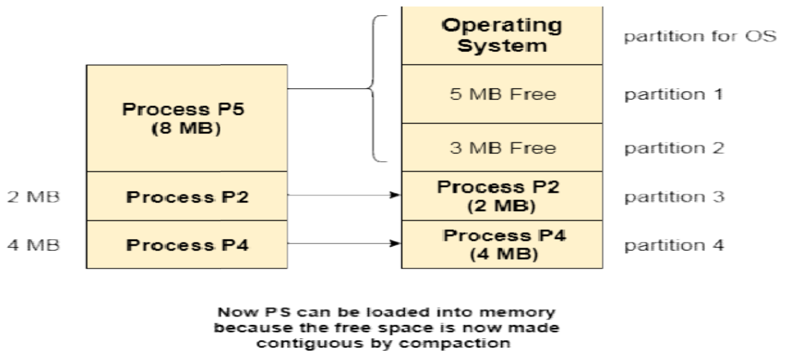
## 3.4 Compaction

We got to know that the dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems.

To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.

We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.



Compaction

As shown in the image above, the process P5, which could not be loaded into the memory due to the lack of contiguous space, can be loaded now in the memory since the free partitions are made contiguous.

**Problem with Compaction**

The efficiency of the system is decreased in the case of compaction due to the fact that all the free spaces will be transferred from several places to a single place.

Huge amount of time is invested for this procedure and the CPU will remain idle for all this time. Despite of the fact that the compaction avoids external fragmentation, it makes system inefficient.

Let us consider that OS needs 6 NS to copy 1 byte from one place to another.

1. 1 B transfer needs 6 NS
2. 256 MB transfer needs 256 X 2^20 X 6 X 10 ^ -9 secs

   hence, it is proved to some extent that the larger size memory transfer needs some huge amount of time that is in seconds.

## 3.5 Partitioning Algorithms

There are various algorithms which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.

The explanation about each of the algorithm is given below.

### 3.5.1 First Fit Algorithm

First Fit algorithm scans the linked list and whenever it finds the first big enough hole to store a process, it stops scanning and load the process into that hole. This procedure produces two partitions. Out of them, one partition will be a hole while the other partition will store the process.

First Fit algorithm maintains the linked list according to the increasing order of starting index. This is the simplest to implement among all the algorithms and produces bigger holes as compare to the other algorithms.

### 3.5.2 Next Fit Algorithm

Next Fit algorithm is similar to First Fit algorithm except the fact that, Next fit scans the linked list from the node where it previously allocated a hole.

Next fit doesn't scan the whole list, it starts scanning the list from the next node. The idea behind the next fit is the fact that the list has been scanned once therefore the probability of finding the hole is larger in the remaining part of the list.

Experiments over the algorithm have shown that the next fit is not better then the first fit. So it is not being used these days in most of the cases.

### 3.5.3 Best Fit Algorithm

The Best Fit algorithm tries to find out the smallest hole possible in the list that can accommodate the size requirement of the process.

Using Best Fit has some disadvantages.

1. It is slower because it scans the entire list every time and tries to find out the smallest hole which can satisfy the requirement the process.
2. Due to the fact that the difference between the whole size and the process size is very small, the holes produced will be as small as it cannot be used to load any process and therefore it remains useless.

Despite of the fact that the name of the algorithm is best fit, It is not the best algorithm among all.

### 3.5.4 Worst Fit Algorithm

The worst fit algorithm scans the entire list every time and tries to find out the biggest hole in the list which can fulfill the requirement of the process.

Despite of the fact that this algorithm produces the larger holes to load the other processes, this is not the better approach due to the fact that it is slower because it searches the entire list every time again and again.

### 3.5.5 Quick Fit Algorithm

The quick fit algorithm suggests maintaining the different lists of frequently used sizes. Although, it is not practically suggestible because the procedure takes so much time to create the different lists and then expending the holes to load a process.

The first fit algorithm is **the best algorithm** among all because

1. It takes lesser time compare to the other algorithms.
2. It produces bigger holes that can be used to load other processes later on.
3. It is easiest to implement.

## 3.6 Need for Paging

Disadvantage of Dynamic Partitioning

The main disadvantage of Dynamic Partitioning is External fragmentation. Although, this can be removed by Compaction but as we have discussed earlier, the compaction makes the system inefficient.

We need to find out a mechanism which can load the processes in the partitions in a more optimal way. Let us discuss a dynamic and flexible mechanism called paging.

### 3.6.1 Need for Paging

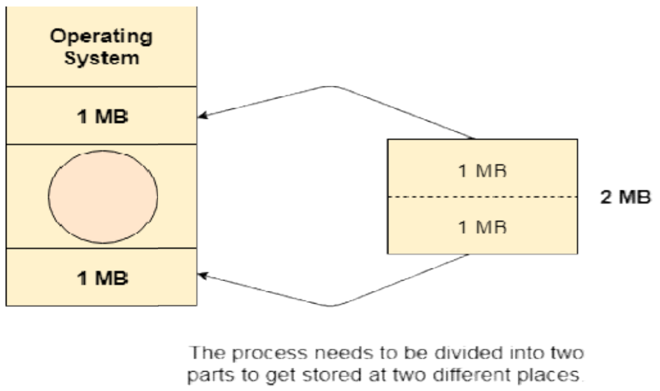
Let’s consider a process P1 of size 2 MB and the main memory which is divided into three partitions. Out of the three partitions, two partitions are holes of size 1 MB each.

P1 needs 2 MB space in the main memory to be loaded. We have two holes of 1 MB each but they are not contiguous.

Although, there is 2 MB space available in the main memory in the form of those holes but that remains useless until it become contiguous. This is a serious problem to address.

We need to have some kind of mechanism which can store one process at different locations of the memory.

The Idea behind paging is to divide the process in pages so that, we can store them in the memory at different holes. We will discuss paging with the examples in the next sections.


Operating System
1 MB
1 MB
1 MB
1 MB
2 MB
The process needs to be divided into two parts to get stored at two different places

**Paging with Example**

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

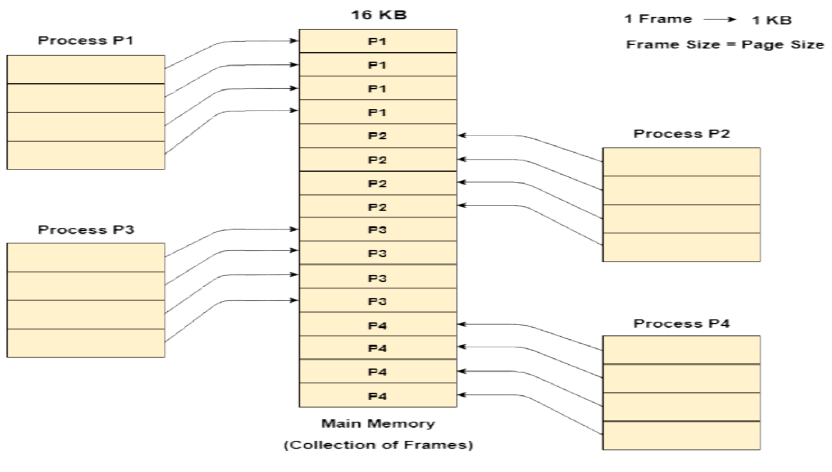
**Example**

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

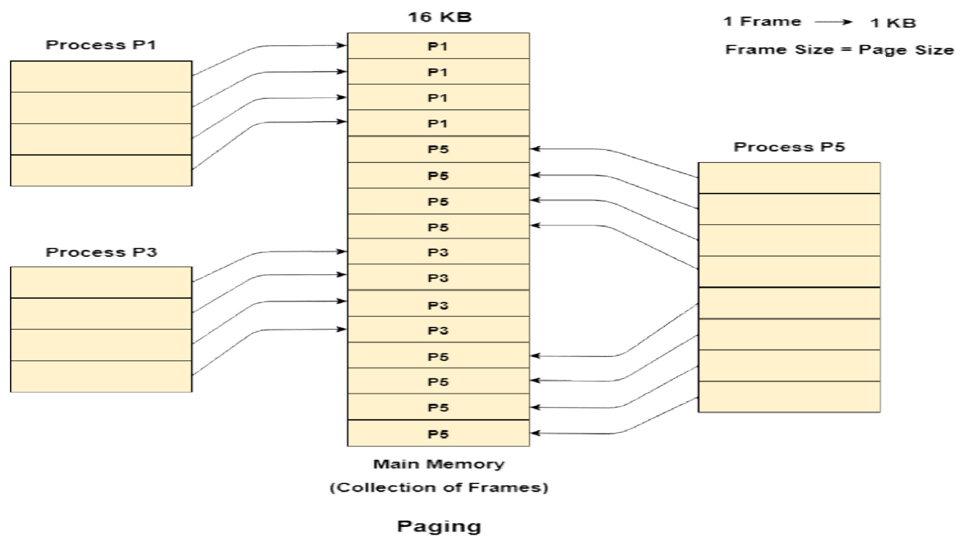
Frames, pages and the mapping between the two is shown in the image below.



Paging

Let's consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.



### 3.6.2 Memory Management Unit

The purpose of Memory Management Unit (MMU) is to convert the logical address into the physical address. The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.

When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.

The logical address has two parts.

1. Page Number
2. Offset

Memory management unit of OS needs to convert the page number to the frame number.

**Example**

Considering the above image, let's say that the CPU demands 10th word of 4th page of process P3. Since the page number 4 of process P1 gets stored at frame number 9 therefore the 10th word of 9th frame will be returned as the physical address.

Physical and Logical Address Space

Physical Address Space

Physical address space in a system can be defined as the size of the main memory. It is really important to compare the process size with the physical address space. The process size must be less than the physical address space.

Physical Address Space = Size of the Main Memory

If, physical address space = 64 KB = 2 ^ 6 KB = 2 ^ 6 X 2 ^ 10 Bytes = 2 ^ 16 bytes

Let us consider,
word size = 8 Bytes = 2 ^ 3 Bytes

Hence,
Physical address space (in words) = (2 ^ 16) / (2 ^ 3) = 2 ^ 13 Words

Therefore,
Physical Address = 13 bits

In General,
If, Physical Address Space = N Words

then, Physical Address = $\log_2 N$

Logical Address Space

Logical address space can be defined as the size of the process. The size of the process should be less enough so that it can reside in the main memory.

**Let's say,**

Logical Address Space = 128 MB = (2 ^ 7 X 2 ^ 20) Bytes = 2 ^ 27 Bytes
Word size = 4 Bytes = 2 ^ 2 Bytes

Logical Address Space (in words) = (2 ^ 27) / (2 ^ 2) = 2 ^ 25 Words
Logical Address = 25 Bits

In general,

If, logical address space = L words
Then, Logical Address = $\text{Log}_2 L$ bits

What is a Word?

The Word is the smallest unit of the memory. It is the collection of bytes. Every operating system defines different word sizes after analyzing the n-bit address that is inputted to the decoder and the 2 ^ n memory locations that are produced from the decoder.

## 3.7 Page Table

Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.

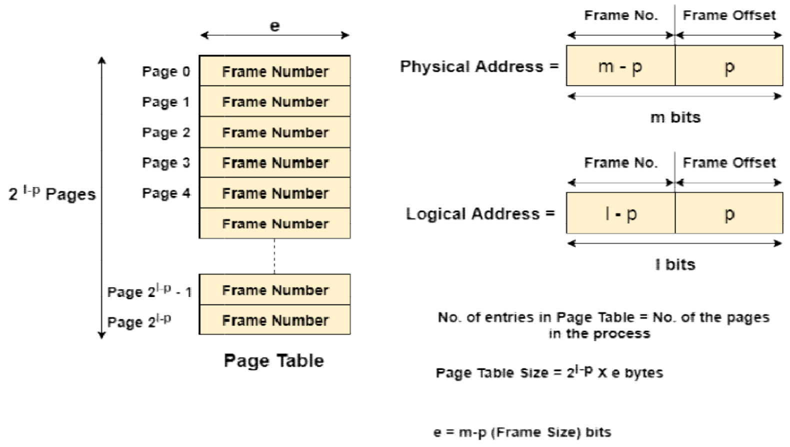
Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes.

Physical addresses are the actual frame address of the memory. They are generally used by the hardware or more specifically by RAM subsystems.

The image given below considers,

Physical Address Space = M words
Logical Address Space = L words
Page Size = P words

Physical Address = $\log_2 M = m$ bits
Logical Address = $\log_2 L = l$ bits
page offset = $\log_2 P = p$ bits



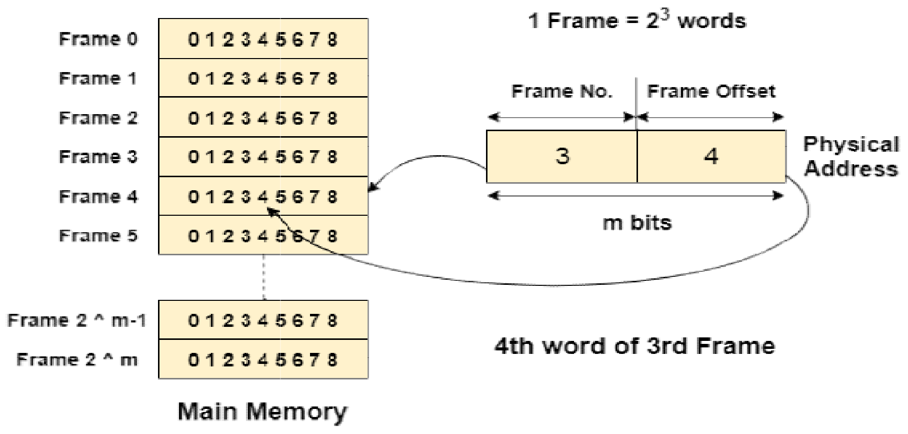
The CPU always accesses the processes through their logical addresses. However, the main memory recognizes physical address only.

In this situation, a unit named as Memory Management Unit comes into the picture. It converts the page number of the logical address to the frame number of the physical address. The offset remains same in both the addresses.

To perform this task, Memory Management unit needs a special kind of mapping which is done by page table. The page table stores all the Frame numbers corresponding to the page numbers of the page table.

In other words, the page table maps the page number to its actual location (frame number) in the memory.

In the image given below shows, how the required word of the frame is accessed with the help of offset.



### 3.7.1 Mapping from page table to main memory

In operating systems, there is always a requirement of mapping from logical address to the physical address. However, this process involves various steps which are defined as follows.

**1. Generation of logical address**

CPU generates logical address for each page of the process. This contains two parts: page number and offset.

2. **Scaling**

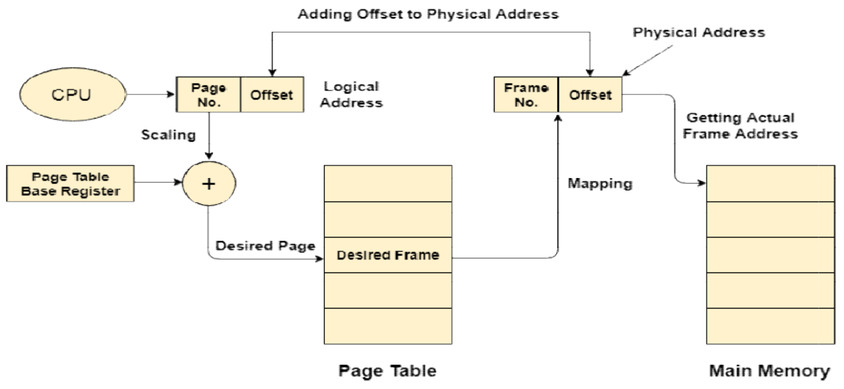
To determine the actual page number of the process, CPU stores the page table base in a special register. Each time the address is generated, the value of the page table base is added to the page number to get the actual location of the page entry in the table. This process is called scaling.

3. **Generation of physical Address**

The frame number of the desired page is determined by its entry in the page table. A physical address is generated which also contains two parts : frame number and offset. The Offset will be similar to the offset of the logical address therefore it will be copied from the logical address.

4. **Getting Actual Frame Number**

The frame number and the offset from the physical address is mapped to the main memory in order to get the actual word address.



## 3.7.2 Page Table Entry

Along with page frame number, the page table also contains some of the bits representing the extra information regarding the page.

Let's see what the each bit represents about the page.

1. **Caching Disabled**

Sometimes, there are differences between the information closest to the CPU and the information closest to the user. Operating system always wants CPU to access user's data as soon as possible. CPU accesses cache which can be inaccurate in some of the cases, therefore, OS can disable the cache for the required pages. This bit is set to 1 if the cache is disabled.

**2. Referenced**

There are various page replacement algorithms which will be covered later in this tutorial. This bit is set to 1 if the page is referred in the last clock cycle otherwise it remains 0.

**3. Modified**

This bit will be set if the page has been modified otherwise it remains 0.

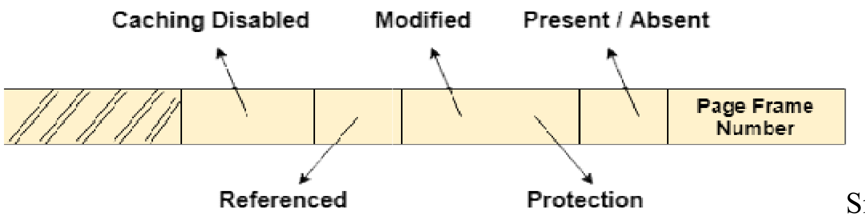
**4. Protection**

The protection field represents the protection level which is applied on the page. It can be read only or read & write or execute. We need to remember that it is not a bit rather it is a field which contains many bits.

**5. Present/Absent**

In the concept of demand paging, all the pages doesn't need to be present in the main memory Therefore, for all the pages that are present in the main memory, this bit will be set to 1 and the bit will be 0 for all the pages which are absent.

If some page is not present in the main memory then it is called page fault of the page table



Size

However, the part of the process which is being executed by the CPU must be present in the main memory during that time period. The page table must also be present in the main memory all the time because it has the entry for all the pages.

The size of the page table depends upon the number of entries in the table and the bytes stored in one entry.

**Let's consider,**

1. Logical Address = 24 bits
2. Logical Address space = 2 ^ 24 bytes
3. Let's say, Page size = 4 KB = 2 ^ 12 Bytes
4. Page offset = 12
5. Number of bits in a page = Logical Address - Page Offset = 24 - 12 = 12 bits
6. Number of pages = 2 ^ 12 = 2 X 2 X 10 ^ 10 = 4 KB
7. Let's say, Page table entry = 1 Byte
8. Therefore, the size of the page table = 4 KB X 1 Byte = 4 KB

Here we are lucky enough to get the page table size equal to the frame size. Now, the page table will be simply stored in one of the frames of the main memory. The CPU maintains a register which contains the base address of that frame, every page number from the logical address will first be added to that base address so that we can access the actual location of the word being asked.

However, in some cases, the page table size and the frame size might not be same. In those cases, the page table is considered as the collection of frames and will be stored in the different frames.

Finding Optimal Page Size

We have seen that the bigger page table size cause an extra overhead because we have to divide that table into the pages and then store that into the main memory.

Our concern must be about executing processes not on the execution of page table. Page table provides a support for the execution of the process. The larger the page Table, the higher the overhead.

**We know that,**

1. Page Table Size = number of page entries in page table X size of one page entry
2. Let's consider an example,
3. Virtual Address Space = 2 GB = 2 X 2 ^ 30 Bytes
4. Page Size = 2 KB = 2 X 2 ^ 10 Bytes

5. Number of Pages in Page Table = (2 X 2 ^ 30)/(2 X 2 ^ 10) = 1 M pages

There will be 1 million pages which is quite big number. However, try to make page size larger, say 2 MB.

Then, Number of pages in page table = (2 X 2 ^ 30)/(2 X 2 ^ 20) = 1 K pages.

If we compare the two scenarios, we can find out that the page table size is anti proportional to Page Size.

In Paging, there is always wastage on the last page. If the virtual address space is not a multiple of page size, then there will be some bytes remaining and we have to assign a full page to those many bytes. This is simply a overhead.

**Let's consider,**

1. Page Size = 2 KB
2. Virtual Address Space = 17 KB
3. Then number of pages = 17 KB / 2 KB

The number of pages will be 9 although the 9th page will only contain 1 byte and the remaining page will be wasted.

**In general,**

1. If page size = p bytes
2. Entry size = e bytes
3. Virtual Address Space = S bytes
4. Then, overhead O = (S/p) X e + (p/2)

On an average, the wasted number of pages in a virtual space is p/2(the half of total number of pages).

For, the minimal overhead,

1. $\partial O/\partial p = 0$
2. $-S/(p^2) + ½ = 0$
3. $p = \sqrt{(2.S.e)}$ bytes

Hence, if the page size $\sqrt{(2.S.e)}$ bytes then the overhead will be minimal.

## 3.8 Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

### 3.8.1 Working of Virtual Memory

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

## 3.9 Demand Paging

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory.

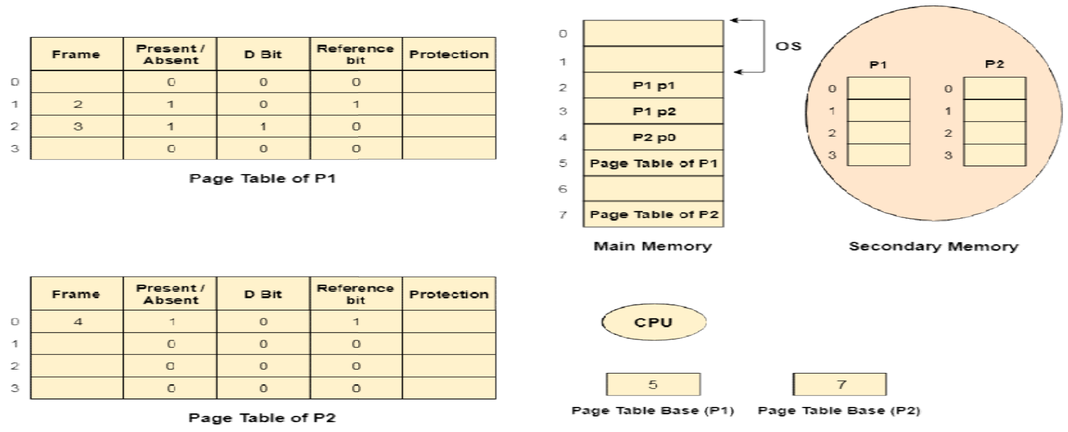
A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced. We will discuss each one of them later in detail.

Snapshot of a virtual memory management system

Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each. The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.

The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.

The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.



| | Frame | Present / Absent | D Bit | Reference bit | Protection |
|---|---|---|---|---|---|
| 0 | | 0 | 0 | 0 | |
| 1 | 2 | 1 | 0 | 1 | |
| 2 | 3 | 1 | 1 | 0 | |
| 3 | | 0 | 0 | 0 | |

Page Table of P1

| | Frame | Present / Absent | D Bit | Reference bit | Protection |
|---|---|---|---|---|---|
| 0 | 4 | 1 | 0 | 1 | |
| 1 | | 0 | 0 | 0 | |
| 2 | | 0 | 0 | 0 | |
| 3 | | 0 | 0 | 0 | |

Page Table of P2

Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

**Demand Paging**

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required.

Whenever any page is referred for the first time in the main memory, then that page will be found in the secondary memory.

After that, it may or may not be present in the main memory depending upon the page replacement algorithm which will be covered later in this tutorial.

## 3.9.1 What is a Page Fault?

If the referred page is not present in the main memory then there will be a miss and the concept is called Page miss or page fault. The CPU has to access the missed page from the secondary memory. If the number of page fault is very high then the effective access time of the system will become very high.

## 3.9.2 What is Thrashing?

If the number of page faults is equal to the number of referred pages or the number of page faults are so high so that the CPU remains busy in just reading the pages from the secondary memory then the effective access time will be the time taken by the CPU to read one word from the secondary memory and it will be so high. The concept is called thrashing.

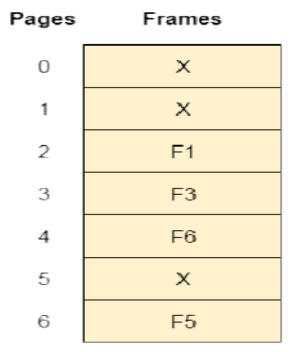
If the page fault rate is PF %, the time taken in getting a page from the secondary memory and again restarting is S (service time) and the memory access time is ma then the effective access time can be given as;

EAT = PF X S + (1 - PF) X (ma)

### 3.9.3 Inverted Page Table

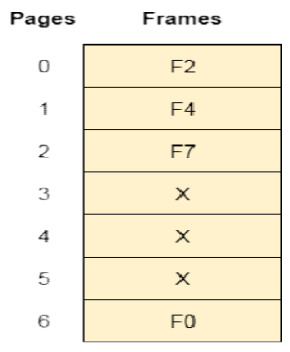
Inverted Page Table is the global page table which is maintained by the Operating System for all the processes. In inverted page table, the number of entries is equal to the number of frames in the main memory. It can be used to overcome the drawbacks of page table.

1. There is always a space reserved for the page regardless of the fact that whether it is present in the main memory or not. However, this is simply the wastage of the memory if the page is not present.



| Pages | Frames |
|---|---|
| 0 | X |
| 1 | X |
| 2 | F1 |
| 3 | F3 |
| 4 | F6 |
| 5 | X |
| 6 | F5 |

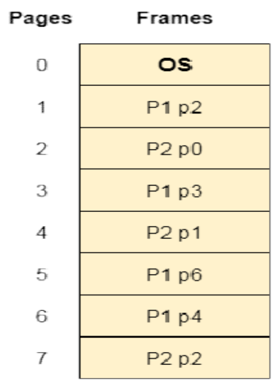
Page Table of P1



| Pages | Frames |
|---|---|
| 0 | F2 |
| 1 | F4 |
| 2 | F7 |
| 3 | X |
| 4 | X |
| 5 | X |
| 6 | F0 |

Page Table of P2

We can save this wastage by just inverting the page table. We can save the details only for the pages which are present in the main memory. Frames are the indices and the information saved inside the block will be Process ID and page number.



| Pages | Frames |
|---|---|
| 0 | OS |
| 1 | P1 p2 |
| 2 | P2 p0 |
| 3 | P1 p3 |
| 4 | P2 p1 |
| 5 | P1 p6 |
| 6 | P1 p4 |
| 7 | P2 p2 |

Inverted Page Table

**References:**

1. https://www.geeksforgeeks.org/introduction-of-operating-system
2. https://www.javatpoint.com/operating-system
3. https://www.gatevidyalay.com/operating-system

Contact Us:

University Campus Address:

**Jayoti Vidyapeeth Women's University**

Vadaant Gyan Valley, Village-Jharna, Mahala Jobner Link Road,
Jaipur Ajmer Express Way, NH-8, Jaipur- 303122, Rajasthan (INDIA)
(Only Speed Post is Received at University Campus Address, No. any Courier Facility is available at Campus Address)

|  |  |
|---|---|
| Pages | : 64 |
| Book Price | : ₹ 150/- |

Women University Press

Jayoti Publication Desk

*Year & Month of Publication- 3/3/2021*