

UNIT-1

OPERATING SYSTEM

SYLLABUS: Introduction to the Operating System, Types of Operating System: Batch System, Time Sharing System, Real Time System, Multi Programming, Distributed System, Functions and Services of OS.

The operating system is the most important program that runs on a computer. Every general purpose computer must have an operating system to run other programs and applications. Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

Definition:- An operating system is a system software which act as an interface between a user and computer hardware. It provides an environment in which a user may execute programs. A computer system can be divided in to four component.

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.

{

“Hardware Provides basic computing resources (CPU, memory, I/O devices).

Operating System- Controls and coordinates the use of hardware among application programs.

Application Programs- Solve computing problems of users (compilers, database systems, video

games, business programs such as banking software).

Users- People, machines, other computers”

}

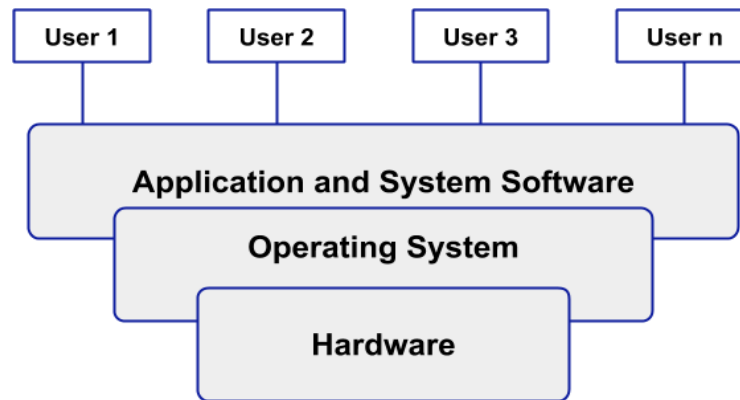


Fig1: Conceptual view of a computer system

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.

Goals of the Operating System:-

There are two types of goals of an Operating System i.e. Primary Goals and Secondary Goal.

- **Primary Goal:** The primary goal of an Operating System is to provide a user-friendly and convenient environment. We know that it is not compulsory to use the Operating System, but things become harder when the user has to perform all the process scheduling and converting the user code into machine code is also very difficult. So, we make the use of an Operating System to act as an intermediate between us and the hardware. All you need to do is give commands to the Operating System and the Operating System will do the rest for you. So, the Operating System should be convenient to use.
- **Secondary Goal:** The secondary goal of an Operating System is efficiency. The Operating System should perform all the management of resources in such a way that the resources are fully utilized and no resource should be held idle if some request to that resource is there at that instant of time

So, in order to achieve the above primary and secondary goals, the Operating System performs

a number of functions.

Evolution of Operating System:-

Evolution -Evolution mean the gradual development of something.

Evolution of operating system is divided into 5 phases

PHASE 0: IN THE BEGINNING (1940-1955):-

Phase 0: No operating system

- Computers are exotic experimental equipment.
- Program in machine language.
- Use plug boards to direct computer.
- No overlap between computation, I/O, think time, and response time.
- Programs manually loaded via card decks.

Phase 1 (1955-1970):-

- Make more efficient use of the computer
- computer: move the person away from the machine.
- User at console: one user at a time
- Batch monitor: load program, run, print
- OS becomes a batch monitor: a program that loads a user's
- If program failed, the OS record the contents of memory and saves it somewhere.
- Os/360 was introduced in 1963; worked in 1968. This Systems were enormously complicated. They were written in assembly code. No structured programming.

MODIFICATIONS:

- More efficient use of hardware.
- Efficiency increases because it processes the jobs as a batch collectively rather than individually.

Limitations

- No protection
- Difficult to debug!

Phase 2 (1970-1980):-

Interactive timesharing: CTSS:

- Developed at MIT. One of the first timesharing systems. to let multiple users interact with the system at the same time

- Sacrifice CPU time to get better response time
- Users do debugging, editing, and email online.

MODIFICATIONS:

- Better utilization of resources.
- More than one user executes their tasks simultaneously.

LIMITATIONS

- Thrashing- Thrashing caused by many Factors including
- Swapping
- Inefficient queuing
- Performance very non-linear response with load

Phase 3: 1980-1990:-

- OS becomes a subroutine library
- One application at a time (MSDOS,
- Required high level protection and privacy for user data.
- Mobile and computer operating systems have CP/M, ...)
- Gates approached Seattle Computer Products, bought 86-DOS, and created MS- DOS.
- GUI operating systems was developed first time in phase 3.
- So many types of GUI operating systems are develop in phase 5 major types are:

First “mice”, “windows”

Apple Lisa/ Macintosh: 1984

Xerox Star

pp /

“Look and Feel” suit 1988

Microsoft Windows: Win 1.0 (1985) .

Modifications:

- OS becomes a subroutine library and command executive.
- finish quickly and run existing programs.

Limitations:-

- Eventually PCs become powerful: OS regains all the complexity of a “big” OS
- memory protection because of multiprogramming.

Phase 4: (1990-2000):-

Networked Systems:

- Networking (Local Area Networking)
- Different machines share resources Printers, File Servers, Web Servers Client – Server Model Services: Computing File Storage

Modifications:

- Internet service providers (service between OS and apps)
- Information becomes a commodity.
- Advertising becomes a computer marketplace.

Limitations

complicated as compare to uni programing been developed in different ways and for different uses. Computer OS products are older and more familiar to larger groups of users.

Phase 5: 2000??-???? Mobile:-

OS system of mobiles.

window 95,

window 98,

window XP,

window crystal vista window 8,

window 10,

Android all Versions.

The designers and developers try to develop operating system and make it user friendly all GUI operating System is user friendly operating system. it is more easy for the user to use GUI OS as compared to Unix, Linux, Ms. Dos etc. because while using these OS user must familiar with its commands . Through the last 22 years, the simple idea of a computer operating system has been continually built on and improved. Through this time, Microsoft Windows, Android and Apple's Mac OS have emerged as the two dominant operating system designs. The goal in OS development is to make the machine convenient to use.

Functions of an Operating System:-

To achieve the goals of an Operating system, the Operating System performs a number of

functionalities. They are:

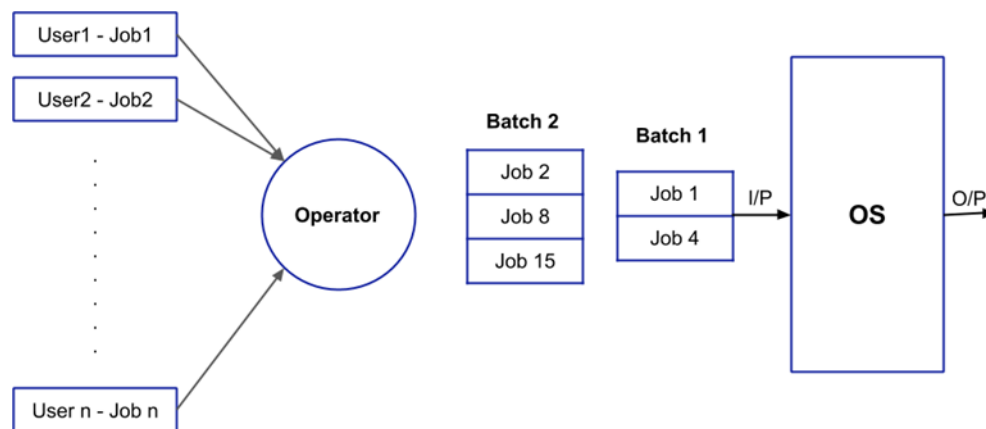
- **Process Management:** At a particular instant of time, the CPU may have a number of processes that are in the ready state. But at a time, only one process can be processed by a processor. So, the CPU should apply some kind of algorithm that can be used to provide uniform and efficient access to resources by the processes. The CPU should not give priority to only one process and it should make sure that every process which is in the ready state will be executed. Some of the CPU scheduling algorithms are First Come First Serve, Round Robin, Shortest Job First, Priority Scheduling, etc.
- **Memory Management:** For the execution of a process, the whole process is put into the main memory and the process is executed and after the execution of the process, the memory is freed and that memory can be used for other processes. So, it is the duty of the Operating System to manage the memory by allocating and deallocating the memory for the process.
- **I/O Device Management:** There are various I/O devices that are present in a system. Various processes require access to these resources and the process should not directly access these devices. So, it is the duty of the Operating System to allow the use of I/O devices by the various process that are requiring these resources.
- **File Management:** There are various files, folders and directory system in a particular computer. All these are maintained and managed by the Operating System of the computer. All these files related information are maintained by using a File Allocation Table or FAT. So, every detail related to the file i.e. filename, file size, file type, etc is stored in the File Allocation Table. Also, it is the duty of the Operating System to make sure that the files should not be opened by some unauthorized access.
- **Virtual Memory:** When the size of the program is larger than the main memory then it is the duty of the Operating System to load only frequently used pages in the main memory. This is called Virtual Memory.

Types of an Operating System:-

1. Batch Operating System

In a Batch Operating System, the similar jobs are grouped together into batches with the help of some operator and these batches are executed one by one. For example, let us assume that

we have 10 programs that need to be executed. Some programs are written in C++, some in C and rest in Java. Now, every time when we run these programmes individually then we will have to load the compiler of that particular language and then execute the code. But what if we make a batch of these 10 programmes. The benefit with this approach is that, for the C++ batch, you need to load the compiler only once. Similarly, for Java and C, the compiler needs to be loaded only once and the whole batch gets executed. The following image describes the working of a Batch Operating System



Advantages:

1. The overall time taken by the system to execute all the programmes will be reduced.
2. The Batch Operating System can be shared between multiple users.

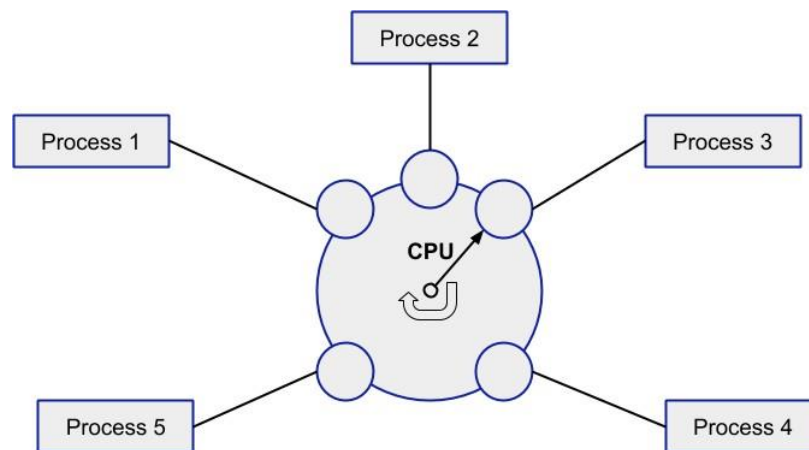
Disadvantages:

1. Manual interventions are required between two batches.
2. The CPU utilization is low because the time taken in loading and unloading of batches is very high as compared to execution time.

Examples of the batch operating system: transactions, payroll system, bank statements, reporting, integration, etc.

2. Time-Sharing or Multi-tasking Operating System

In a Multi-tasking Operating System, more than one processes are being executed at a particular time with the help of the time-sharing concept. So, in the time-sharing environment, we decide a time that is called time quantum and when the process starts its execution then the execution continues for only that amount of time and after that, other processes will be given chance for that amount of time only. In the next cycle, the first process will again come for its execution and it will be executed for that time quantum only and again next process will come. This process will continue. The following image describes the working of a Time-Sharing Operating System.



Advantages:

1. Since equal time quantum is given to each process, so each process gets equal opportunity to execute. The CPU will be busy in most of the cases and this is good to have case.

Disadvantages:

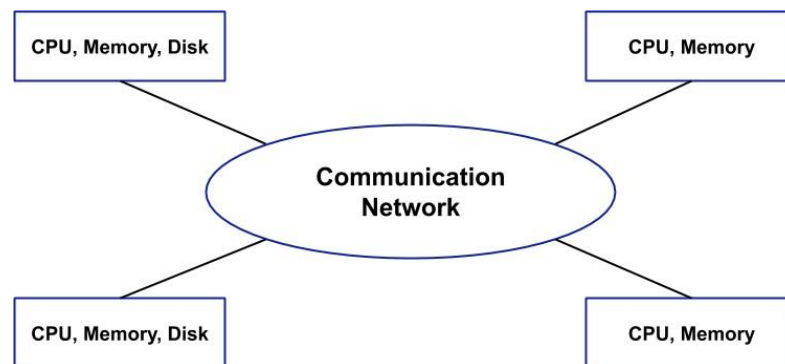
1. Process having higher priority will not get the chance to be executed first because the equal opportunity is given to each process.

Examples of multitasking: eating and watching TV simultaneously, chatting during classes, eating chocolates while walking, talking on a phone while walking, etc.

3. Distributed Operating System

In a Distributed Operating System, we have various systems and all these systems have their own CPU, main memory, secondary memory, and resources. These systems are connected to each other using a shared communication network. Here, each system can perform its task individually. The best part about these Distributed Operating System is remote access i.e. one user can access the data of the other system and can work accordingly. So, remote access is possible in these distributed Operating Systems. The

following image shows the working of a Distributed Operating System.



Advantages:

1. Since the systems are connected with each other so, the failure of one system can't stop the execution of processes because other systems can do the execution.
2. Resources are shared between each other.
3. The load on the host computer gets distributed and this, in turn, increases the efficiency.

Disadvantages:

1. Since the data is shared among all the computers, so to make the data secure and accessible to few computers, you need to put some extra efforts.
2. If there is a problem in the communication network then the whole communication will be broken.

Examples of distributed OS: intranets, the internet, sensors networks, etc.

4. Embedded Operating System

An Embedded Operating System is designed to perform a specific task for a particular device which is not a computer. For example, the software used in elevators is dedicated to the working of elevators only and nothing else. So, this can be an example of Embedded Operating System. The Embedded Operating System allows the access of device hardware to the software that is running on the top of the Operating System.

The block diagram of an embedded system consists of input devices, output devices, and memory.

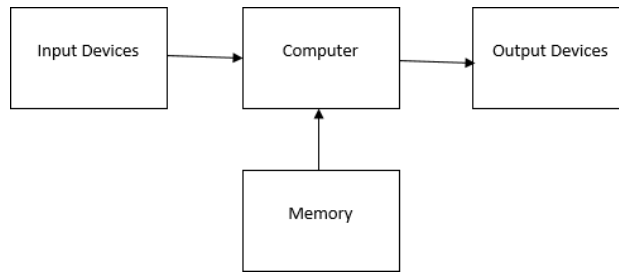


Fig The block diagram of an embedded

Input Devices: Input devices are used to send the data from the user to the system, here the user is the input. Some of the input devices are Keyboard, mouse, microphone, hard disk, sensors, switches, etc.

Output Devices: Out devices show the result to the humans in the form of text, image or sounds. Some of the output devices are printers, monitors, LCD, LED, motors, relays, buzzers, etc.

Memory: The memory is used to store the data. Some of the memory devices are SD card, EEPROM (Electrically Erasable Programmable Read-Only Memory), Flash memory. The memory devices used in the embedded system are Non-volatile RAM, volatile RAM, Dynamic Random Access Memory), etc.

Advantages:

1. Since it is dedicated to a particular job, so it is fast.
2. Low cost.
3. These consume less memory and other resources.

Disadvantages:

1. Only one job can be performed.
2. It is difficult to upgrade or is nearly scalable.

Some applications of the embedded operating system are shown in the below

- **Mobiles, Washing machines, Televisions, Microwave Ovens, Televisions, Computers, Laptops, Dishwashers, ATM's, Satellites, Vehicles**

5. Real-time Operating System

The Real-time Operating Systems are used in the situation where we are dealing with some real-time data. So, as soon as the data comes, the execution of the process should be done and there should be no delay i.e. no buffer delays should be there. Real-time OS is a time-sharing system that is based on the concept of clock interrupt. So, whenever you want to process a

large number of request in a very short period of time, then you should use Real-time Operating System. For example, the details of the temperature of the petroleum industry are very crucial and this should be done in real-time and in a very short period of time. A small delay can result in a life-death situation. So, this is done with the help of Real-time Operating System. There are two types of Real-time Operating System:

1. **Hard Real-time:** In this type, a small delay can lead to drastic change. So, when the time constraint is very important then we use the Hard Real-time.
2. **Soft Real-time:** Here, the time constraint is not that important but here also we are dealing with some real-time data.

Advantages:

1. There is maximum utilization of devices and resources. These systems are almost error-free.

Disadvantages:

1. The algorithms used in Real-time Operating System is very complex.
2. Specific device drivers are used for responding to the interrupts as soon as possible.

Example of real time OS:-

- **Radar systems, network switching control systems, satellite monitoring systems, satellite launch-control and maneuvering mechanisms, global positioning systems all have their roots in RTOS.**
- **Now a days RTOS are increasingly finding use in strategic and military operations. These are used in guided missile launching units, track-and-trace spy satellites, etc.**

6.Network operating System

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

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

The advantages of network operating systems are as follows –

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.

- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows –

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

Examples of network OS: Windows 2000, Linux, Microsoft windows, etc.

Multiprogramming

Definition: Multiprogramming OS is an ability of an operating system that executes more than one program using a single processor machine. More than one task or program or jobs are present inside the main memory at one point of time.

Buffering and spooling can overlap I/O and CPU tasks to improve the system performance but it has some limitations that a single user cannot always keep CPU or I/O busy all the time. To increase resource utilization, multiprogramming approaches. The OS could pick and start the execution of one of the jobs in memory, whenever the jobs does not need CPU that means the job is working with I/O at that time the CPU is idle at that time the OS switches to another job in memory and CPU executes a portion of it till the job issues a request for I/O and so on.

Let's P1 and P2 are two programs present in the main memory. The OS picks one program and starts executing it. During execution if the P1 program requires I/O operation, then the OS will simply switch over to P2 program. If the p2 program requires I/O then again it switches to P3 and so on. If there is no other program remaining after P3 then the CPU will pass its control back to the previous program.

Advantages

The advantages of multiprogramming operating system are as follows –

- CPU utilization is high because the CPU is never goes to idle state.
- Memory utilization is efficient.
- CPU throughput is high and also supports multiple interactive user terminals.

Disadvantages

The disadvantages of multiprogramming operating system are as follows –

- CPU scheduling is compulsory because lots of jobs are ready to run on CPU simultaneously.
- User is not able to interact with jobs when it is executing.
- Programmers also cannot modify a program that is being executed.

Note: If several jobs are ready in main memory and if there is not enough space for all of them, then the system has to choose them by making a decision, this process is called job scheduling. When the OS selects a job from the group of jobs and loads that job into memory for execution, therefore it needs memory management, if several such jobs are ready then it needs CPU scheduling.

Operating System Services:-

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

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

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

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

Program execution:-

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

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

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

I/O Operation:-

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide

the peculiarities of specific hardware devices from the users. An Operating System manages the communication between user and device drivers.

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

File system manipulation:-

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

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

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

Communication:-

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

The OS handles routing and connection strategies, and the problems of contention and security.

Following are the major activities of an operating system with respect to communication –

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

Error handling:-

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory

hardware. Following are the major activities of an operating system with respect to error handling –

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

Resource Management:-

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

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

Protection:-

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection –

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

Components of Operating System:-

The components of an operating system play a key role to make a variety of computer system parts work together. The operating components are discussed below.



Kernel

The kernel in the OS provides the basic level of control on all the computer peripherals. In the operating system, the kernel is an essential component that loads firstly and remains within the main memory. So that memory accessibility can be managed for the programs within the RAM, it creates the programs to get access from the hardware resources. It resets the operating states of the CPU for the best operation at all times.

Process Execution

The OS gives an interface between the hardware as well as an application program so that the program can connect through the hardware device by simply following procedures & principles configured into the OS. The program execution mainly includes a process created through an OS kernel that uses memory space as well as different types of other resources.

Interrupt

In the operating system, interrupts are essential because they give a reliable technique for the OS to communicate & react to their surroundings. An interrupt is nothing but one kind of signal between a device as well as a computer system otherwise from a program in the computer that requires the OS to leave and decide accurately what to do subsequently. Whenever an interrupt signal is received, then the hardware of the computer puts on hold automatically whatever computer program is running presently, keeps its status & runs a computer program which is connected previously with the interrupt.

Memory Management

The functionality of an OS is nothing but memory management which manages main memory & moves processes backward and forward between disk & main memory during implementation. This tracks each & every memory position; until it is assigned to some process otherwise it is open. It verifies how much memory can be allocated to processes and also makes a decision to know

which process will obtain memory at what time. Whenever memory is unallocated, then it tracks correspondingly to update the status. Memory management work can be divided into three important groups like memory management of hardware, OS and application memory management.

Multitasking

It describes the working of several independent computer programs on a similar computer system. Multitasking in an OS allows an operator to execute one or more computer tasks at a time. Since many computers can perform one or two tasks at a time, usually this can be done with the help of time-sharing, where each program uses the time of a computer to execute.

Networking

Networking can be defined as when the processor interacts with each other through communication lines. The design of communication-network must consider routing, connection methods, safety, the problems of opinion & security.

Presently most of the operating systems maintain different networking techniques, hardware, & applications. This involves that computers that run on different operating systems could be included in a general network to share resources like data, computing, scanners, printers, which uses the connections of either wired otherwise wireless.

Security

If a computer has numerous individuals to allow the immediate process of various processes, then the many processes have to be protected from other activities. This system security mainly depends upon a variety of technologies that work effectively. Current operating systems give an entry to a number of resources, which are obtainable to work the software on the system, and to external devices like networks by means of the kernel. The operating system should be capable of distinguishing between demands which have to be allowed for progressing & others that don't need to be processed. Additionally, to permit or prohibit a security version, a computer system with a high level of protection also provides auditing options. So this will allow monitoring the requests from accessibility to resources

User Interface

A GUI or user interface (UI) is the part of an OS that permits an operator to get the information. A user interface based on text displays the text as well as its commands which are typed over a

command line with the help of a keyboard.

The OS-based applications mainly provide a specific user interface for efficient communication. The main function of a user interface of an application is to get the inputs from the operator & to provide o/p's to the operator. But, the sorts of inputs received from the user interface as well as the o/p types offered by the user interface may change from application to application. The UI of any application can be classified into two types namely GUI (graphical UI) & CLI (command line user interface).

Thus, this is all about an overview of an operating system. The main components of an OS mainly include kernel, API or application program interface, user interface & file system, hardware devices and device drivers.

.

UNIT-2

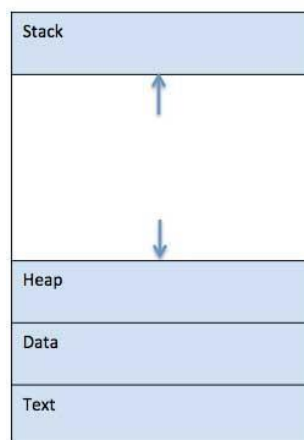
PROCESS MANAGEMENT

SYLLABUS: Process Concept, Process State, Process Control Block, Process Scheduling, CPU Scheduling- Scheduling Criteria, Scheduling Algorithms, Preemptive & Non Preemptive Scheduling.

Process

A program in execution is called process. The processor (**A processor, is a small chip that resides in computers and other electronic devices. Its basic job is to receive input and provide the appropriate output.**) has to manage several activities at one time. Each activity is correspond to one process. Process execution must progress in sequential fashion. The OS must maintain a data structure for each process, which describes the state and resource ownership of that process, and which enables the OS to exert control over each process.

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



- | | |
|---|---|
| 1 | Stack
The process Stack contains the temporary data such as method/function parameters, return address and local variables. |
|---|---|

2	Heap This is dynamically allocated memory to a process during its run time.
3	Text This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	Data This section contains the global and static variables.

Difference between Process and a Program:-

Main()

```
{
  int i, prod=1;
  for(i=0;i<100;i++)
    prod=prod*i;
}
```

- This is a program that contain one multiplication statement(prod=prod*i) but the process will execute 100 multiplication.
- Process is active entity.
- Program is passive entity.

SR.NO.	PROGRAM	PROCESS
1.	Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
2.	Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
3.	Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
4.	Program is a static entity.	Process is a dynamic entity.
5.	Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
6.	Program does not have any control block.	Process has its own control block called Process Control Block.

Process States or Process Life Cycle:-

When a process comes in execution it change its states. The state of a process is defined in part by the current activity of that process. Each process may be one of the following state during the time of execution.

New: In this state the process is being created.

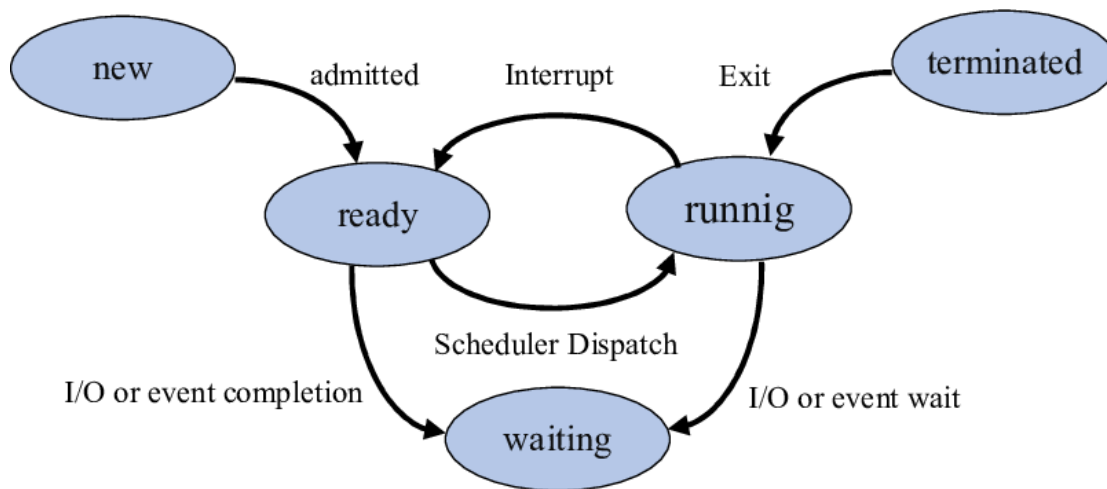
Ready: In this state the process is to be assign a duty by the processor.

Waiting: In this state the process is waiting or blocked for some event to occur such as input/output completion.

Running: In this state Instructions are executed.

Terminated: In this state the process has finished the execution

State Transition Diagram:-



Process Control Block (PCB):-

In OS each process is represented by a process control block or task control block. It is a data structure that physically represent a process in the memory of a computer system. It contains pieces of information associate with a specific process that include following:-

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

Pointer:- A pointer to parent process.

Process State:- It indicates the information about the state of process such as blocked ready running etc.

Process ID:- Each process is assigned a unique identification number, when it is entered into the system.

Program Counter:- It indicates the address of the next instruction to be executed.

CPU Registers:- It indicates the information about the contents of the CPU registers. The information of CPU registers must be saved when an interrupt occurs, so that the process can be continued correctly afterward. Registers hold the processed the result of calculations or addresses pointing to the memory locations of desired data.

CPU Scheduling Information:- It indicates the information needed for CPU scheduling such as process priority, pointers to scheduling queues and other scheduling parameters.

Memory Management Information:- It indicates the information needed for memory management such as value of the base and limit registers, page tables or segment tables, amount

of memory units allocated to the process etc.

Accounting Information:- It indicates the information about process number, CPU used by the process time limits etc.

I/O Status Information:- It indicates the information about I/O devices allocated to the process a list of open files access rights of files opened and so on,

Link to Parent Process:- A new process can be created from existing process; the existing process is called the parent process of the newly created process. The address of the PCB of parent process is stored.

Link to Child Process:- The addresses of the PCBs of the child processes in the main memory are stored.

CPU Scheduling:-

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Schedulers:-

A scheduler is a type of system software that allows you to handle process scheduling.

There are mainly three types of Process Schedulers:

1. Long Term
2. Short Term
3. Medium Term

Long Term Scheduler or Job Scheduler :-

Long term scheduler is also known as a **job scheduler**. This scheduler regulates the program and select process from the queue and loads them into memory for execution. It also regulates the degree of multi-programming.

However, the main goal of this type of scheduler is to offer a balanced mix of jobs, like Processor, I/O jobs., that allows managing multiprogramming.

Medium Term Scheduler:-

Medium-term scheduling is an important part of **swapping**. It enables you to handle the swapped out-processes. In this scheduler, a running process can become suspended, which makes an I/O request.

A running process can become suspended if it makes an I/O request. A suspended processes can't make any progress towards completion. In order to remove the process from memory and make space for other processes, the suspended process should be moved to secondary storage.

Short Term Scheduler:-

Short term scheduling is also known as **CPU scheduler**. The main goal of this scheduler is to boost the system performance according to set criteria. This helps you to select from a group of processes that are ready to execute and allocates CPU to one of them. The dispatcher gives control of the CPU to the process selected by the short term scheduler.

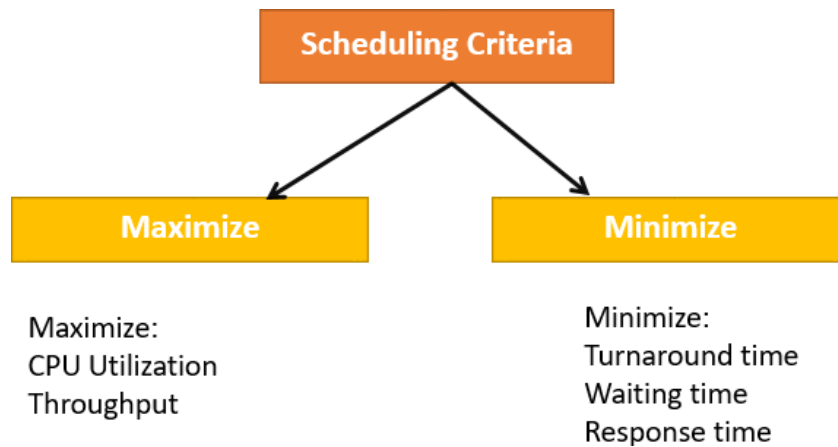
Difference between Schedulers:-

Long-Term Vs. Short Term Vs. Medium-Term

Long-Term	Short-Term	Medium-Term
Long term is also known as a job scheduler	Short term is also known as CPU scheduler	Medium-term is also called swapping scheduler.
It is either absent or minimal in a time-sharing system.	It is insignificant in the time-sharing order.	This scheduler is an element of Time-sharing systems.
Speed is less compared to the short term scheduler.	Speed is the fastest compared to the short-term and medium-term scheduler.	It offers medium speed.
Allow you to select processes from the loads and pool back into the memory	It only selects processes that is in a ready state of the execution.	It helps you to send process back to memory.
Offers full control	Offers less control	Reduce the level of multiprogramming.

CPU Scheduling Criteria

A CPU scheduling algorithm tries to maximize and minimize the following:



Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

Throughput: The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that specific process needs to wait in the ready queue.

Response time: It is an amount to time in which the request was submitted until the first response is produced.

Turnaround Time: Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completiontime is the turnaround time.

Summary: Scheduling Criteria

- There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:
 - **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from

40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput** - Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.
- **Turnaround time** - Time required for a particular process to complete, from submission time to completion.
- **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
- **Response time** - The time taken in an interactive program from the issuance of a command to the *commence* of a response to that command.

Preemptive Scheduling:

- Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state.
- The resources (mainly CPU cycles) are allocated to the process for the limited amount of time and then is taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining.
- That process stays in ready queue till it gets next chance to execute.

Non-Preemptive Scheduling:

- Non-preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state.
- In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.
- In case of non-preemptive scheduling does not interrupt a process running CPU in middle of the execution.
- Instead, it waits till the process complete its CPU burst time and then it can allocate the CPU to another process.

Preemptive Scheduling	Non Preemptive Scheduling
The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Process can be interrupted in between.	Process cannot be interrupted till it terminates or switches to waiting state.
If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

In brief:

Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

Turn Around Time: Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

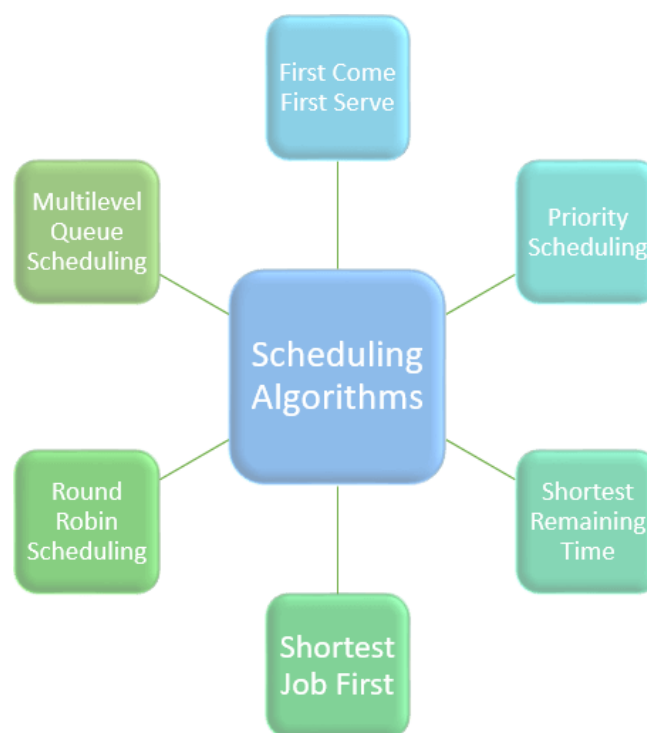
Waiting Time(W.T): Time Difference between turnaround time and burst time.

Waiting Time = Turn Around Time – Burst Time

Types of CPU scheduling Algorithm

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling



(a) First Come First Serve (FCFS)

In FCFS Scheduling

- The process which arrives first in the ready queue is firstly assigned the CPU.
- In case of a tie, process with smaller process id is executed first.
- It is always non-preemptive in nature.
- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.

- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Advantages-

- It is simple and easy to understand.
- It can be easily implemented using queue data structure.
- It does not lead to starvation.

Disadvantages-

- It does not consider the priority or burst time of the processes.
- ▯ It suffers from convoy effect i.e. processes with higher burst time arrived before the processes with smaller burst time.

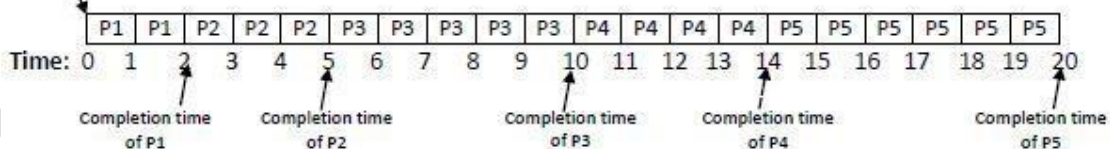
Example 1:

Q. Consider the following processes with burst time (CPU Execution time). Calculate the average waiting time and average turnaround time?

Process id	Arrival time	Burst time/CPU execution time
P1	0	2
P2	1	3
P3	2	5
P4	3	4
P5	4	6

Sol.

Gantt chart



Turnaround time = Completion time – Arrival time

Waiting time = Turnaround time – Burst time

Process id	Arrival time	Burst time	Completion time	Turnaround time	Waiting time
P1	0	2	2	2-0=2	2-2=0
P2	1	3	5	5-1=4	4-3=1
P3	2	5	10	10-2=8	8-5=3
P4	3	4	14	14-3=11	11-4=7
P5	4	6	20	20-4=16	16-6=10

Average turnaround time = $\sum_{i=0}^n \text{Turnaround time}(i)/n$

where, n= no. of process

Average waiting time = $\sum_{i=0}^n \text{Waiting time}(i)/n$

where, n= no. of process

Average turnaround time = $2+4+8+11+16/5 = 41/5 = 8.2$

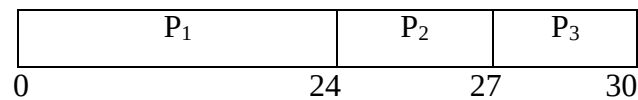
Average waiting time = $0+1+3+7+10/5 = 21/5 = 4.2$

Example 2:

Consider the processes P1, P2, P3 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time,

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	24
P2	0	3
P3	0	3

Gantt chart



PROCESS	WAIT TIME	TURN AROUND TIME
P1	0	24
P2	24	27
P3	27	30

Total Wait Time = $0 + 24 + 27 = 51$ ms

Average Waiting Time = (Total Wait Time) / (Total number of processes) = $51/3 = 17$ ms

Total Turn Around Time: $24 + 27 + 30 = 81$ ms

Average Turn Around time = (Total Turn Around Time) / (Total number of processes)
 $= 81 / 3 = 27$ ms

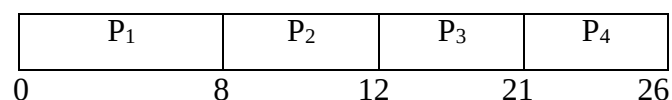
Throughput = $3 \text{ jobs}/30 \text{ sec} = 0.1 \text{ jobs/sec}$

Example 3:

Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with given Arrival Time and Burst Time.

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt chart



PROCESS	WAIT TIME	TURN AROUND TIME
P1	0	$8 - 0 = 8$
P2	$8 - 1 = 7$	$12 - 1 = 11$
P3	$12 - 2 = 10$	$21 - 2 = 19$
P4	$21 - 3 = 18$	$26 - 3 = 23$

Total Wait Time:= $0 + 7 + 10 + 18 = 35$ ms

Average Waiting Time = (Total Wait Time) / (Total number of processes)= $35/4 = 8.75$ ms

Total Turn Around Time: $8 + 11 + 19 + 23 = 61$ ms

Average Turn Around time = (Total Turn Around Time) / (Total number of processes)
 $61/4 = 15.25$ ms

Throughput: $4 \text{ jobs}/26 \text{ sec} = 0.15385 \text{ jobs/sec}$

(b) Shortest Job First (SJF)

- Process which have the shortest burst time are scheduled first.
- If two processes have the same burst time, then FCFS is used to break the tie.
- This is a non-pre-emptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time process will take.
- Pre-emptive mode of Shortest Job First is called as Shortest Remaining Time First (SRTF).

Advantages-

- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages-

- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time.
- Priorities can not be set for the processes.
- Processes with larger burst time have poor response time.

Example-01:

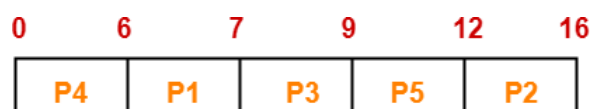
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Solution-

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turnaround time.

Gantt Chart-



Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	7	$7 - 3 = 4$	$4 - 1 = 3$
P2	16	$16 - 1 = 15$	$15 - 4 = 11$
P3	9	$9 - 4 = 5$	$5 - 2 = 3$
P4	6	$6 - 0 = 6$	$6 - 6 = 0$
P5	12	$12 - 2 = 10$	$10 - 3 = 7$

Now,

- Average Turn Around time = $(4 + 15 + 5 + 6 + 10) / 5 = 40 / 5 = 8$ unit
- Average waiting time = $(3 + 11 + 3 + 0 + 7) / 5 = 24 / 5 = 4.8$ unit

Example-02:

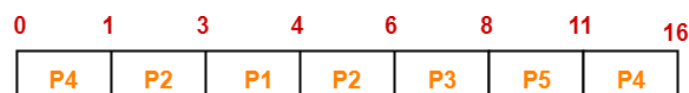
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF pre-emptive, calculate the average waiting time and average turnaround time.

Solution-

Gantt Chart-



Gantt Chart

Process Id	Exit time	Turn Around time	Waiting time
P1	4	$4 - 3 = 1$	$1 - 1 = 0$
P2	6	$6 - 1 = 5$	$5 - 4 = 1$
P3	8	$8 - 4 = 4$	$4 - 2 = 2$
P4	16	$16 - 0 = 16$	$16 - 6 = 10$
P5	11	$11 - 2 = 9$	$9 - 3 = 6$

Now,

- Average Turn Around time = $(1 + 5 + 4 + 16 + 9) / 5 = 35 / 5 = 7$ unit
- Average waiting time = $(0 + 1 + 2 + 10 + 6) / 5 = 19 / 5 = 3.8$ unit

Example-03:

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	7
P2	1	5
P3	2	3
P4	3	1
P5	4	2
P6	5	1

If the CPU scheduling policy is shortest remaining time first, calculate the average waiting time and average turnaround time.

Solution-**Gantt Chart-**

Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	19	$19 - 0 = 19$	$19 - 7 = 12$
P2	13	$13 - 1 = 12$	$12 - 5 = 7$
P3	6	$6 - 2 = 4$	$4 - 3 = 1$
P4	4	$4 - 3 = 1$	$1 - 1 = 0$
P5	9	$9 - 4 = 5$	$5 - 2 = 3$
P6	7	$7 - 5 = 2$	$2 - 1 = 1$

Now,

- Average Turn Around time = $(19 + 12 + 4 + 1 + 5 + 2) / 6 = 43 / 6 = 7.17$ unit
- Average waiting time = $(12 + 7 + 1 + 0 + 3 + 1) / 6 = 24 / 6 = 4$ unit

Example -04:

Consider the set of 3 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	9
P2	1	4
P3	2	9

If the CPU scheduling policy is SRTF, calculate the average waiting time and average turn around time.

Solution-**Gantt Chart-**

Gantt Chart

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	$13 - 0 = 13$	$13 - 9 = 4$
P2	5	$5 - 1 = 4$	$4 - 4 = 0$
P3	22	$22 - 2 = 20$	$20 - 9 = 11$

Now,

- Average Turn Around time = $(13 + 4 + 20) / 3 = 37 / 3 = 12.33$ unit
- Average waiting time = $(4 + 0 + 11) / 3 = 15 / 3 = 5$ unit

Example-05:

Consider the set of 4 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	20
P2	15	25
P3	30	10
P4	45	15

If the CPU scheduling policy is SRTF, calculate the waiting time of process P2.

Solution-

Gantt Chart-



Gantt Chart

Now, we know-

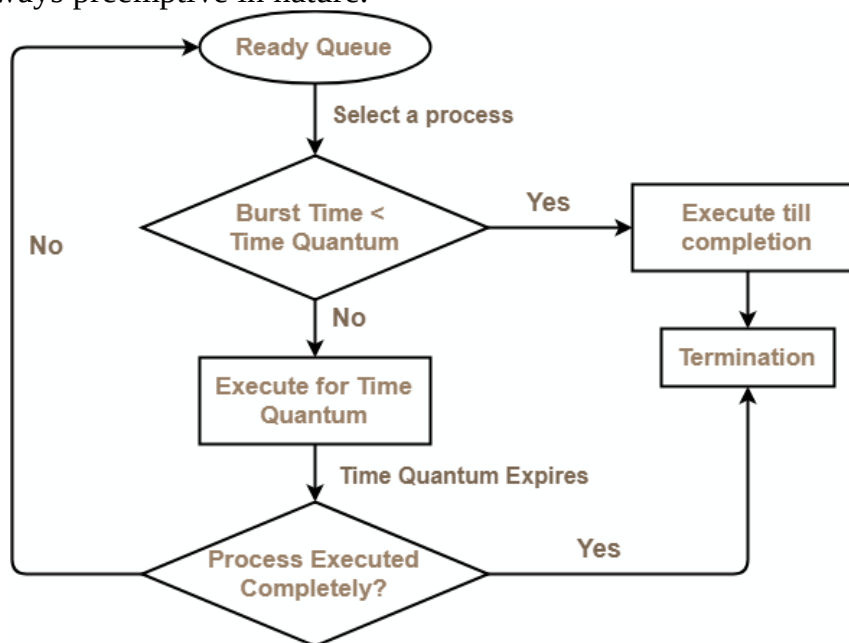
- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Thus,

- Turn Around Time of process P2 = 55 – 15 = 40 unit
- Waiting time of process P2 = 40 – 25 = 15 unit

(b) Round Robin Scheduling

- CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
- This fixed amount of time is called as time quantum or time slice.
- After the time quantum expires, the running process is preempted and sent to the ready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.



Round Robin Scheduling

Advantages-

- It gives the best performance in terms of average response time.
- It is best suited for time sharing system, client server architecture and interactive system.

Disadvantages-

- It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
- Its performance heavily depends on time quantum.
- Priorities can not be set for the processes.

With decreasing value of time quantum,

- Number of context switch increases
- Response time decreases
- Chances of starvation decreases

Thus, smaller value of time quantum is better in terms of response time.

With increasing value of time quantum,

- Number of context switch decreases
- Response time increases
- Chances of starvation increases

Thus, higher value of time quantum is better in terms of number of context switch.

- With increasing value of time quantum, Round Robin Scheduling tends to become FCFS Scheduling.
- When time quantum tends to infinity, Round Robin Scheduling becomes FCFS Scheduling.
- The performance of Round Robin scheduling heavily depends on the value of time quantum.
- The value of time quantum should be such that it is neither too big nor too small.

Example-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turnaround time.

Solution-

Ready Queue- P5, P1, P2, P5, P4, P1, P3, P2, P1

Gantt Chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	13	$13 - 0 = 13$	$13 - 5 = 8$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	5	$5 - 2 = 3$	$3 - 1 = 2$
P4	9	$9 - 3 = 6$	$6 - 2 = 4$
P5	14	$14 - 4 = 10$	$10 - 3 = 7$

Now,

- Average Turn Around time = $(13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6$ unit
- Average waiting time = $(8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8$ unit

Problem-02:

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 2, calculate the *average waiting time* and *average turnaround time*.

Solution-

Ready Queue- P5, P6, P2, P5, P6, P2, P5, P4, P1, P3, P2, P1

Gantt chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	8	$8 - 0 = 8$	$8 - 4 = 4$
P2	18	$18 - 1 = 17$	$17 - 5 = 12$
P3	6	$6 - 2 = 4$	$4 - 2 = 2$
P4	9	$9 - 3 = 6$	$6 - 1 = 5$
P5	21	$21 - 4 = 17$	$17 - 6 = 11$
P6	19	$19 - 6 = 13$	$13 - 3 = 10$

Now,

- Average Turn Around time = $(8 + 17 + 4 + 6 + 17 + 13) / 6 = 65 / 6 = 10.84$ unit
- Average waiting time = $(4 + 12 + 2 + 5 + 11 + 10) / 6 = 44 / 6 = 7.33$ unit

Problem-03: Consider the set of 6 processes whose arrival time and burst time are given below-

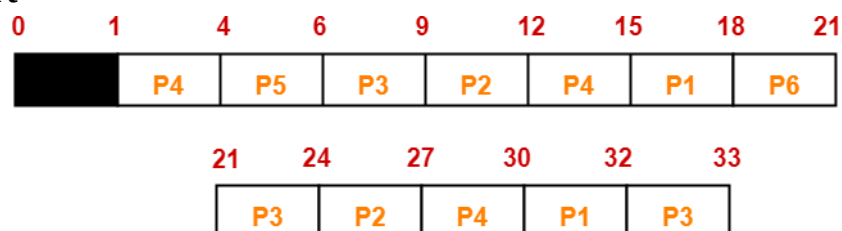
Process Id	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 3, calculate the average waiting time and average turnaround time.

Solution-

Ready Queue- P3, P1, P4, P2, P3, P6, P1, P4, P2, P3, P5, P4

Gantt chart-



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

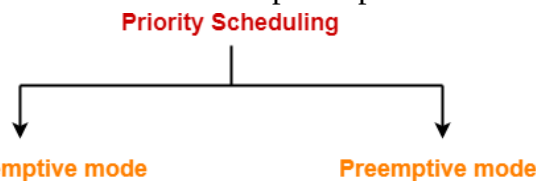
Process Id	Exit time	Turn Around time	Waiting time
P1	32	$32 - 5 = 27$	$27 - 5 = 22$
P2	27	$27 - 4 = 23$	$23 - 6 = 17$
P3	33	$33 - 3 = 30$	$30 - 7 = 23$
P4	30	$30 - 1 = 29$	$29 - 9 = 20$
P5	6	$6 - 2 = 4$	$4 - 2 = 2$
P6	21	$21 - 6 = 15$	$15 - 3 = 12$

Now,

- Average Turn Around time = $(27 + 23 + 30 + 29 + 4 + 15) / 6 = 128 / 6 = 21.33$ unit
- Average waiting time = $(22 + 17 + 23 + 20 + 2 + 12) / 6 = 96 / 6 = 16$ unit

(c) Priority Scheduling

- Out of all the available processes, CPU is assigned to the process having the highest priority.
- In case of a tie, it is broken by **FCFS Scheduling**.
- Priority Scheduling can be used in both preemptive and non-preemptive mode.



- The waiting time for the process having the highest priority will always be zero in preemptive mode.
- The waiting time for the process having the highest priority may not be zero in non-preemptive mode.

Priority scheduling in preemptive and non-preemptive mode behaves exactly same under following conditions-

- The arrival time of all the processes is same
- All the processes become available

Advantages-

- It considers the priority of the processes and allows the important processes to run first.
- Priority scheduling in pre-emptive mode is best suited for real time operating system.

Disadvantages-

- Processes with lesser priority may starve for CPU.
- There is no idea of response time and waiting time.

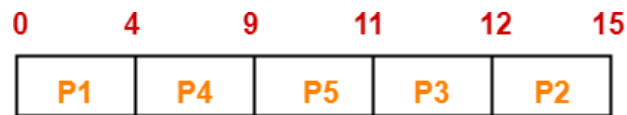
Problem-01:

Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority non-preemptive, calculate the average waiting time and average turnaround time. (*Higher number represents higher priority*)

**Solution-
Gantt Chart-**



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	4	$4 - 0 = 4$	$4 - 4 = 0$
P2	15	$15 - 1 = 14$	$14 - 3 = 11$
P3	12	$12 - 2 = 10$	$10 - 1 = 9$
P4	9	$9 - 3 = 6$	$6 - 5 = 1$
P5	11	$11 - 4 = 7$	$7 - 2 = 5$

Now,

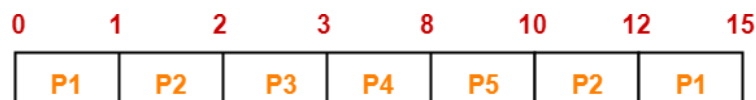
- Average Turn Around time = $(4 + 14 + 10 + 6 + 7) / 5 = 41 / 5 = 8.2$ unit
- Average waiting time = $(0 + 11 + 9 + 1 + 5) / 5 = 26 / 5 = 5.2$ unit

Problem-02: Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority).

**Solution-
Gantt Chart-**



Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	15	$15 - 0 = 15$	$15 - 4 = 11$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	3	$3 - 2 = 1$	$1 - 1 = 0$
P4	8	$8 - 3 = 5$	$5 - 5 = 0$
P5	10	$10 - 4 = 6$	$6 - 2 = 4$

Now,

- Average Turn Around time = $(15 + 11 + 1 + 5 + 6) / 5 = 38 / 5 = 7.6$ unit
- Average waiting time = $(11 + 8 + 0 + 0 + 4) / 5 = 23 / 5 = 4.6$ unit

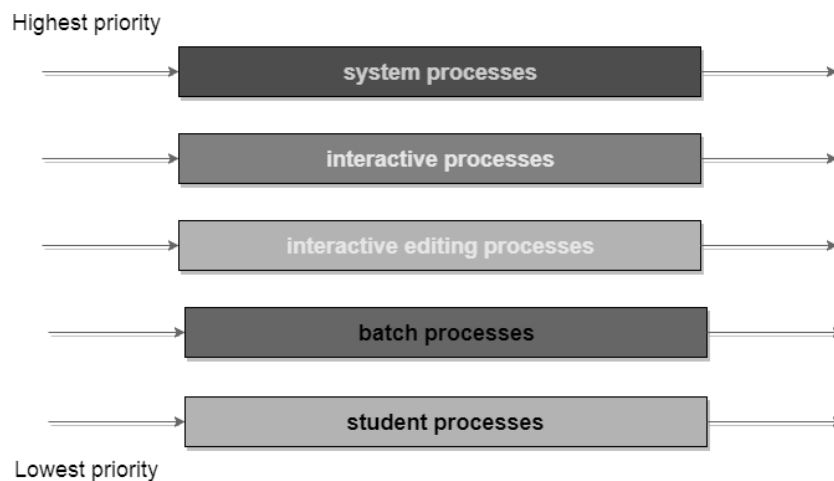
(d) Multilevel Queue Scheduling

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be pre-empted.



UNIT-3

DEADLOCK

SYLLABUS: System model, Characterization, Deadlock Prevention, Deadlock Avoidance, and Detection, Recovery from deadlock.

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

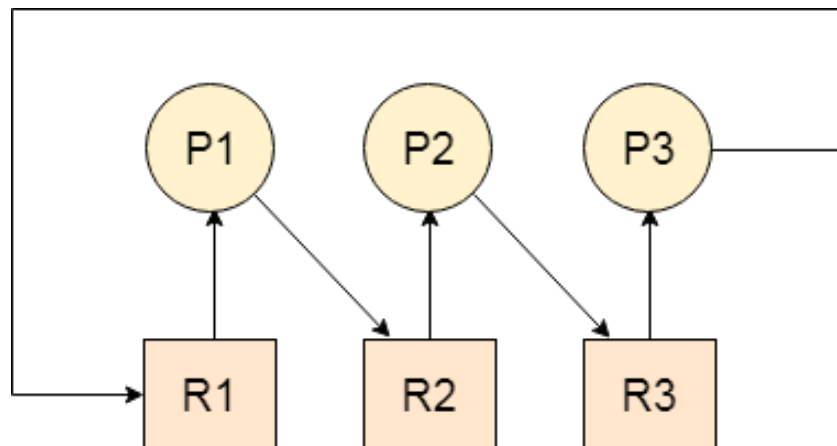
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

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

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R2 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



System Model:-

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs.
- Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc. By definition, all the resources within a category are equivalent, and a request of this category can be equally satisfied by any one of the resources in that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category needs to be further divided into separate categories. For example, "printers" may need to be separated into "laser printers" and "color inkjet printers".
- Some categories may have a single resource.
- In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:
 1. **Request** - If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls `open( )`, `malloc( )`, `new( )`, and `request( )`.
 2. **Use** - The process uses the resource, e.g. prints to the printer or reads from the file.
 3. **Release** - The process relinquishes the resource. so that it becomes available for other processes. For example, `close( )`, `free( )`, `delete( )`, and `release( )`.
- For all kernel-managed resources, the kernel keeps track of what resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available. Application-managed resources can be controlled using mutexes or `wait( )` and `signal( )` calls, (i.e. binary or counting semaphores.)
- A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set (and which can only be released when that other waiting process makes progress.)

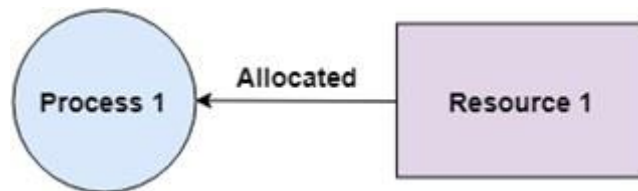
Deadlock Characterization:-

Necessary conditions for Deadlocks

1. Mutual Exclusion

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

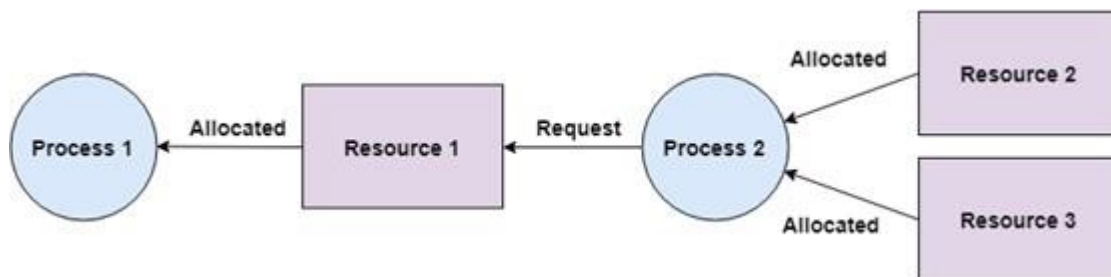
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



2. Hold and Wait

A process waits for some resources while holding another resource at the same time.

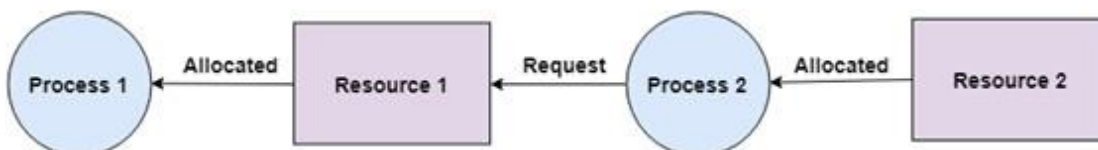
A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



3. No preemption

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

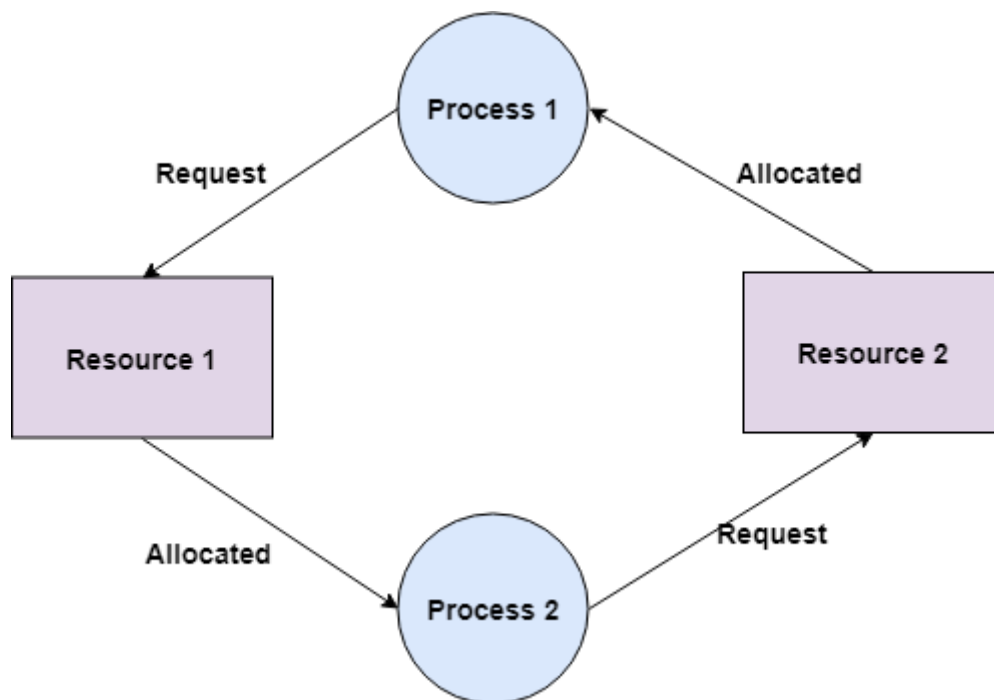
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



4. Circular Wait

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Methods for Handling Deadlocks

- Generally speaking there are three ways of handling deadlocks:
1. **Deadlock prevention or avoidance** - Do not allow the system to get into a deadlocked state. In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)

2. **Deadlock detection and recovery** - Abort a process or preempt some resources when deadlocks are detected. Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.
3. **Ignore the problem all together** - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take. If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

1. Deadlock Prevention

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

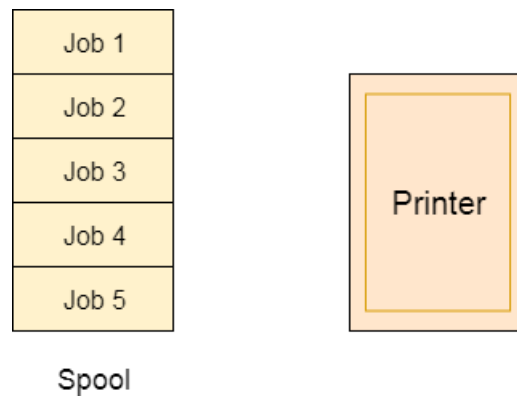
However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

Mutual Exclusion

Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that pool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

3. No Preemption

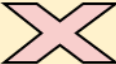
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

Deadlock Avoidance:-

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

Resources Assigned

Process	Type 1	Type 2	Type 3	Type 4
A	3	0	2	2
B	0	0	1	1
C	1	1	1	0
D	2	1	4	0

Resources still needed

Process	Type 1	Type 2	Type 3	Type 4
A	1	1	0	0
B	0	1	1	2
C	1	2	1	0
D	2	1	1	2

E = (7 6 8 4)

P = (6 2 8 3)

A = (1 4 0 1)

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

Banker's Algorithm:-

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S . Everytime a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than S . It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

Let us assume that there are n processes and m resource types. Some data structures that are used to implement the banker's algorithm are:

1. Available

It is an **array** of length m . It represents the number of available resources of each type. If $Available[j] = k$, then there are k instances available, of resource type $R(j)$.

2. Max

It is an $n \times m$ matrix which represents the maximum number of instances of each resource that a process can request. If $Max[i][j] = k$, then the process $P(i)$ can request at most k instances of resource type $R(j)$.

3. Allocation

It is an $n \times m$ matrix which represents the number of resources of each type currently allocated to each process. If $Allocation[i][j] = k$, then process $P(i)$ is currently allocated k instances of resource type $R(j)$.

4. Need

It is an $n \times m$ matrix which indicates the remaining resource needs of each process. If $Need[i][j] = k$, then process $P(i)$ may need k more instances of resource type $R(j)$ to complete its task.

$$Need[i][j] = Max[i][j] - Allocation[i][j]$$

Safety Algorithm:-

This also used for finding out whether a system is in a safe state or not.

1) Let *Work* and *Finish* be vectors of length ' m ' and ' n ' respectively.

Initialize: $Work = Available$

$Finish[i] = false$; for $i=1, 2, 3, 4, \dots, n$

2) Find an i such that both

a) $Finish[i] = false$

b) $Need_i \leq Work$

if no such i exists goto step (4)

3) $Work = Work + Allocation[i]$

$Finish[i] = true$

goto step (2)

4) if $Finish[i] = true$ for all i

then the system is in a safe state

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i . $Request_i[j] = k$ means process P_i wants k instances of resource type R_j . When a request for resources is made by process P_i , the following actions are taken:

1) If $Request_i \leq Need_i$

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i \leq Available$

Goto step (3); otherwise, P_i must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

$Available = Available - Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

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

Question1. What will be the content of the Need matrix?

$Need[i, j] = Max[i, j] - Allocation[i, j]$ For

$P_0 = (7, 5, 3) - (0, 1, 0)$

Need for $P_0 = (7, 4, 3)$ For

$P_1 = (3, 2, 2) - (2, 0, 0)$

Need for $P_1 = (1, 2, 2)$

For $P_2 = (9, 0, 2) - (3, 0, 2)$

Need for $P_2 = (6, 0, 0)$ For

$P_3 = (2, 2, 2) - (2, 1, 1)$

Need for $P_3 = (0, 1, 1)$ For

$P_4 = (4, 3, 3) - (0, 0, 2)$

Need for $P_4 = (4, 3, 1)$

So, the content of Need Matrix is:

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

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

Step 1 of Safety Algo

$m=3, n=5$

Work = Available

3	3	2		
0	1	2	3	4

Finish =

false	false	false	false	false
-------	-------	-------	-------	-------

Step 2:

For $i=0$

Need₀ = 7, 4, 3

Finish [0] is false and Need₀ > Work

So P₀ must wait

But Need ≤ Work

Step 2:

For $i=1$

Need₁ = 1, 2, 2

Finish [1] is false and Need₁ < Work

So P₁ must be kept in safe sequence

Step 3

Work = Work + Allocation₁

5	3	2		
0	1	2	3	4

Finish =

false	true	false	false	false
-------	------	-------	-------	-------

Step 2:

For $i=2$

Need₂ = 6, 0, 0

Finish [2] is false and Need₂ > Work

So P₂ must wait

Step 2:

For $i=3$

Need₃ = 0, 1, 1

Finish [3] is false and Need₃ < Work

So P₃ must be kept in safe sequence

Step 3

Work = Work + Allocation₃

7	4	3		
0	1	2	3	4

Finish =

false	true	false	true	false
-------	------	-------	------	-------

Step 2:

For $i=4$

Need₄ = 4, 3, 1

Finish [4] is false and Need₄ < Work

So P₄ must be kept in safe sequence

Step 3

Work = Work + Allocation₄

7	4	5		
0	1	2	3	4

Finish =

false	true	false	true	true
-------	------	-------	------	------

Step 2:

For $i=0$

Need₀ = 7, 4, 3

Finish [0] is false and Need < Work

So P₀ must be kept in safe sequence

Step 3

Work = Work + Allocation₀

7	5	5		
0	1	2	3	4

Finish =

true	true	false	true	true
------	------	-------	------	------

Step 2:

For $i=2$

Need₂ = 6, 0, 0

Finish [2] is false and Need₂ < Work

So P₂ must be kept in safe sequence

Step 3

Work = Work + Allocation₂

10	5	7		
0	1	2	3	4

Finish =

true	true	true	true	true
------	------	------	------	------

Step 4

Finish [i] = true for $0 \leq i \leq n$

Hence the system is in Safe state

The safe sequence is P₁, P₃, P₄, P₀, P₂

Question3. What will happen if process **P1** requests one additional instance of resource type **A** and two instances of resource type **C**?

A B C
Request₁ = 1, 0, 2

To decide whether the request is granted we use Resource Request algorithm

Step 1
Request₁ < Need₁ ✓

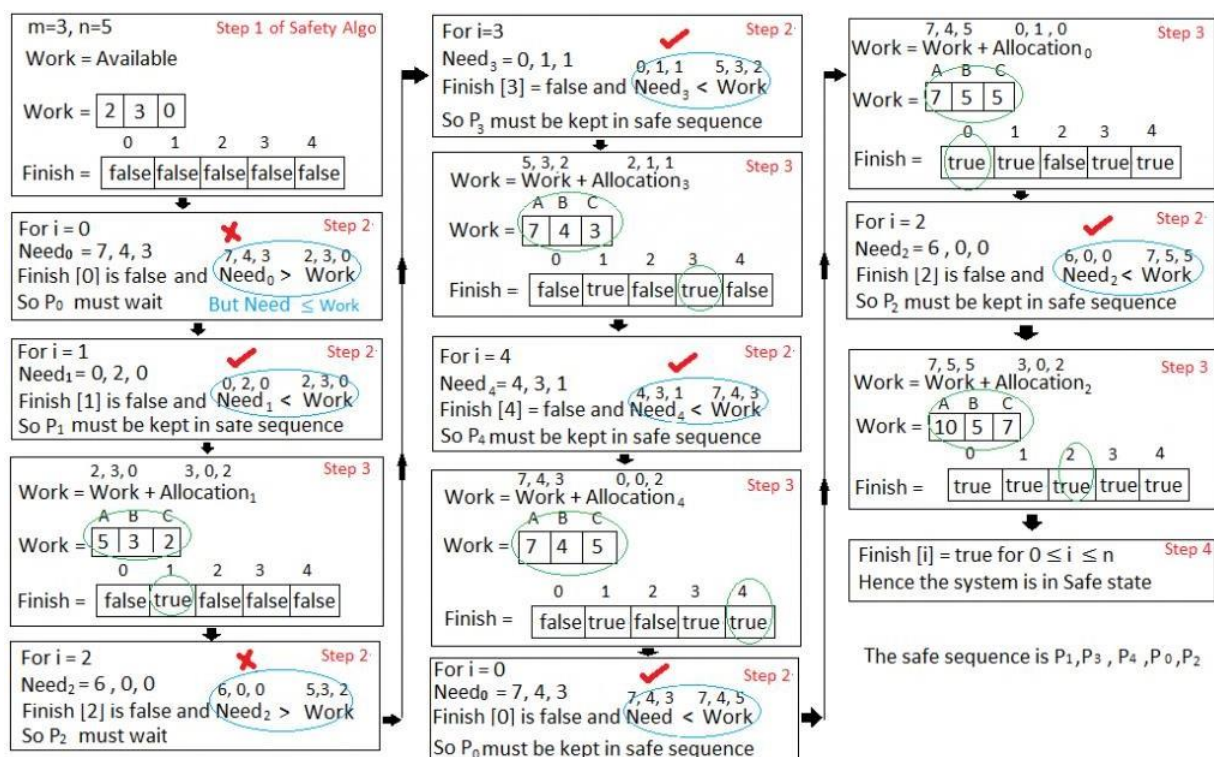
Step 2
Request₁ < Available ✓

Step 3

Available = Available – Request₁
Allocation₁ = Allocation₁ + Request₁
Need₁ = Need₁ - Request₁

Process	Allocation	Need	Available
	A B C	A B C	A B C
P ₀	0 1 0	7 4 3	2 3 0
P ₁	3 0 2	0 2 0	
P ₂	3 0 2	6 0 0	
P ₃	2 1 1	0 1 1	
P ₄	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



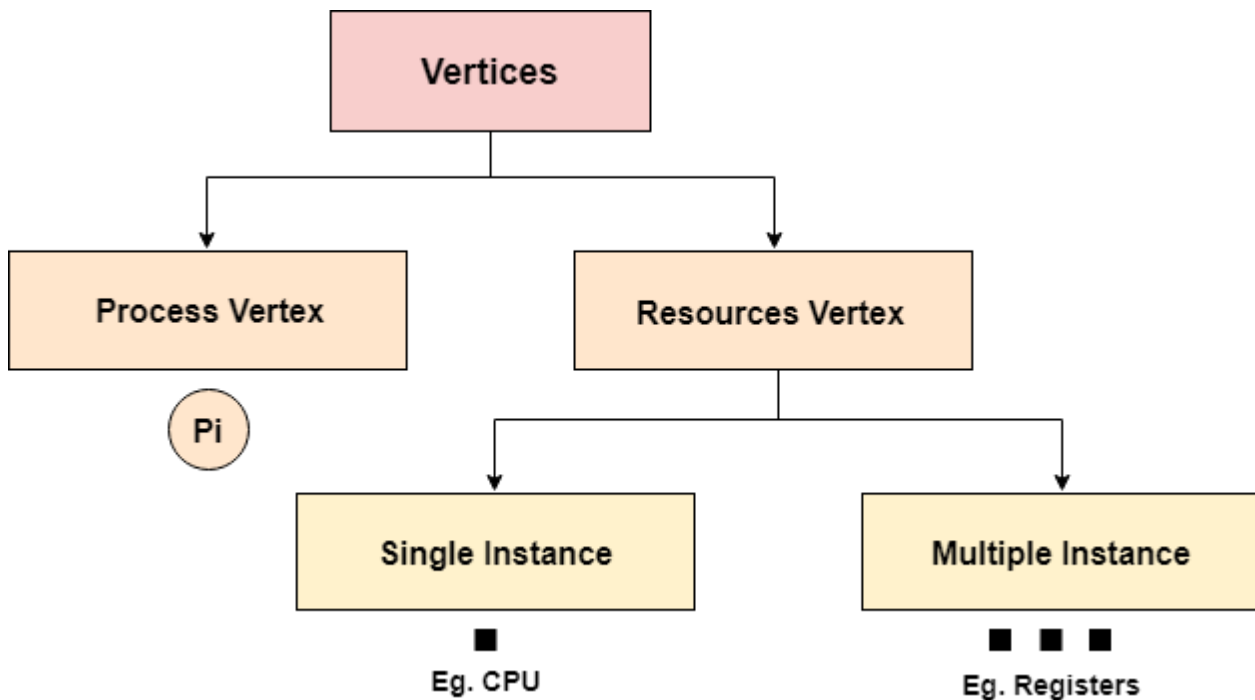
Hence the new system state is safe, so we can immediately grant the request for process **P1**.

Resource Allocation Graph:-

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

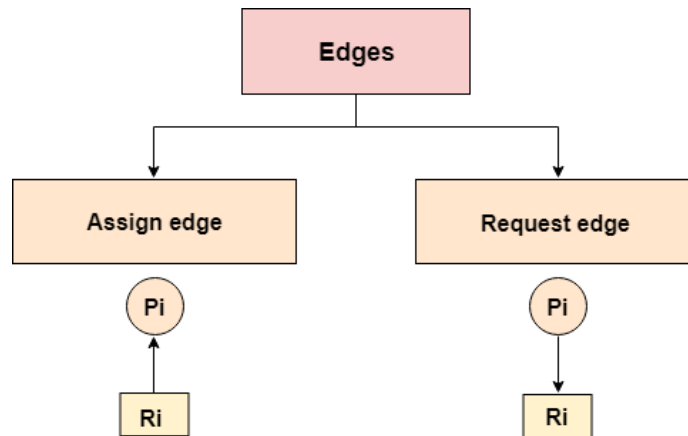
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

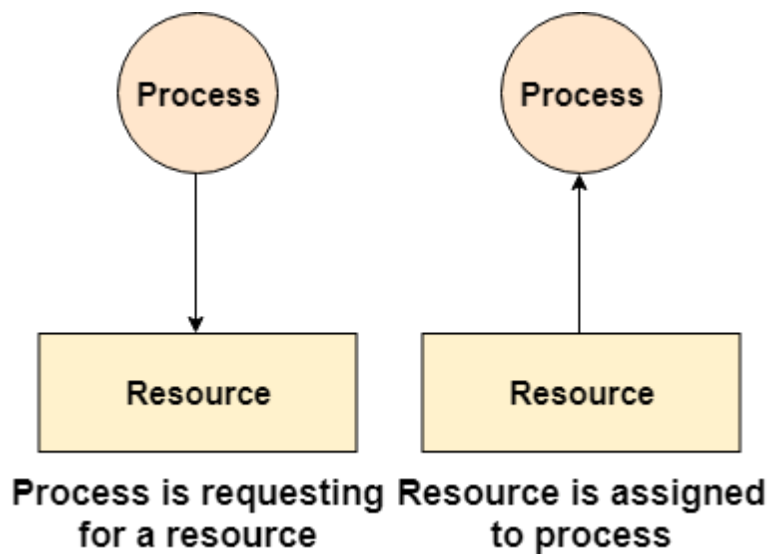
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instance of the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

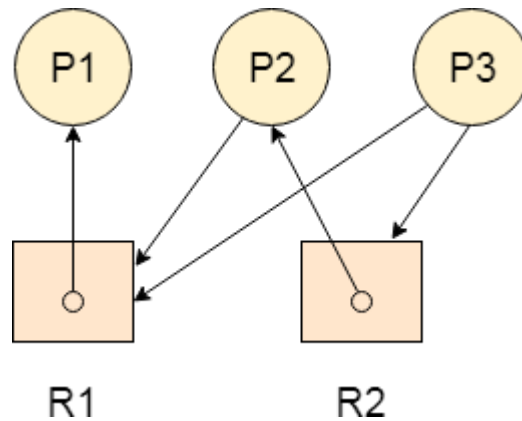


Example

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.

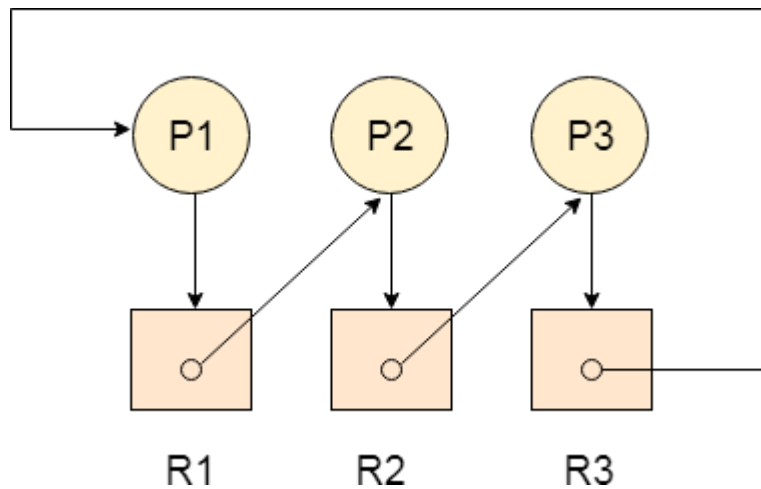


Deadlock Detection using RAG

If a cycle is being formed in a Resource allocation graph where all the resources have the single instance then the system is deadlocked.

In Case of Resource allocation graph with multi-instanced resource types, Cycle is a necessary condition of deadlock but not the sufficient condition.

The following example contains three processes P1, P2, P3 and three resources R1, R2, R3. All the resources are having single instances each.



If we analyze the graph then we can find out that there is a cycle formed in the graph since the system is not satisfying all the four conditions of deadlock.

Allocation Matrix

Allocation matrix can be formed by using the Resource allocation graph of a system. In Allocation matrix, an entry will be made for each of the resource assigned. For Example, in the following matrix, an entry is being made in front of P1 and below R3 since R3 is assigned to P1.

Process	R1	R2	R3
P1	0	0	1
P2	1	0	0
P3	0	1	0

Request Matrix

In request matrix, an entry will be made for each of the resource requested. As in the following example, P1 needs R1 therefore an entry is being made in front of P1 and below R1.

Process	R1	R2	R3
P1	1	0	0
P2	0	1	0
P3	0	0	1

Avial = (0,0,0)

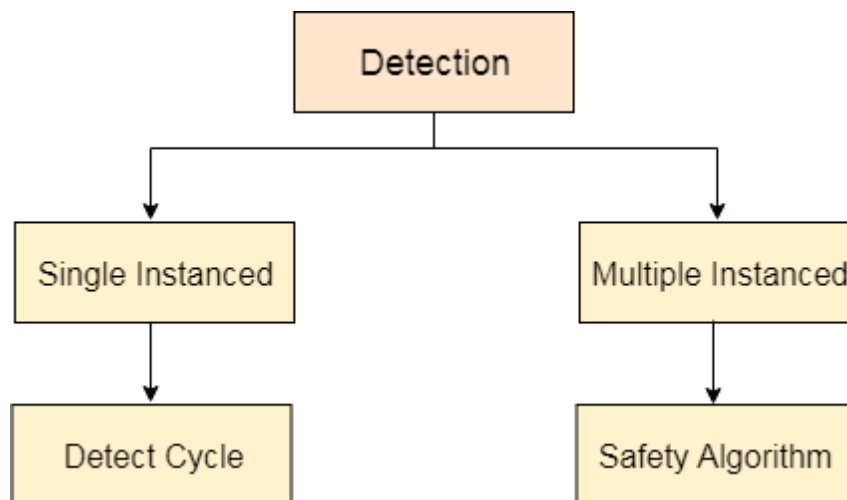
Neither we are having any resource available in the system nor a process going to release. Each of the process needs at least single resource to complete therefore they will continuously be holding each one of them.

We cannot fulfill the demand of at least one process using the available resources therefore the system is deadlocked as determined earlier when we detected a cycle in the graph.

Deadlock Detection and Recovery

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes

For Resource

Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

For Process

Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.

