OS Important ki Answers

1. Define OS and its functions.

Ans. A programs that acts as a intermediary between a user of a computer and the computer hardware.

- OS is a **resource allocator**
    - Manages all resources
    - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
    - Controls execution of programs to prevent errors and improper use of the computer

Operating System Functions:

1. Process Management

A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

2. Memory Management

All data in memory before and after processing

All instructions in memory in order to execute

Memory management activities

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes and data to move into and out of memory
- Allocating and deallocating memory space as needed

3.Storage Management

- OS provides uniform, logical view of information storage
    - Abstracts physical properties to logical storage unit - file
    - Each medium is controlled by device (i.e., disk drive, tape drive)

- File-System management
    - Files usually organized into directories
    - Access control

OS activities include

- Creating and deleting files and directories
- Primitives to manipulate files and dirs
- Mapping files onto secondary storage
- Backup files onto stable storage media

4. Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time.
- Proper management is of central importance.
- OS activities:
    - Free-space management
    - Storage allocation
    - Disk scheduling

5. I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user.
- I/O subsystem responsible for:
    - Buffering (storing data temporarily while it is being transferred)
    - Caching (storing parts of data in faster storage for performance).
    - Spooling (method of managing input/output (I/O) tasks by storing data temporarily in a buffer before sending it to a device, such as a printer).
    - General device-driver interface
    - Drivers for specific hardware devices

6. Protection and Security

- Protection – Mechanism for controlling access of processes or users to resources defined by the OS.
- Security – Defense of the system against attacks.
- Systems generally first distinguish among users, to determine who can do what

- User identities include name and associated number, one per user
- User ID then associated with all files, processes of that user to determine access control

Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file

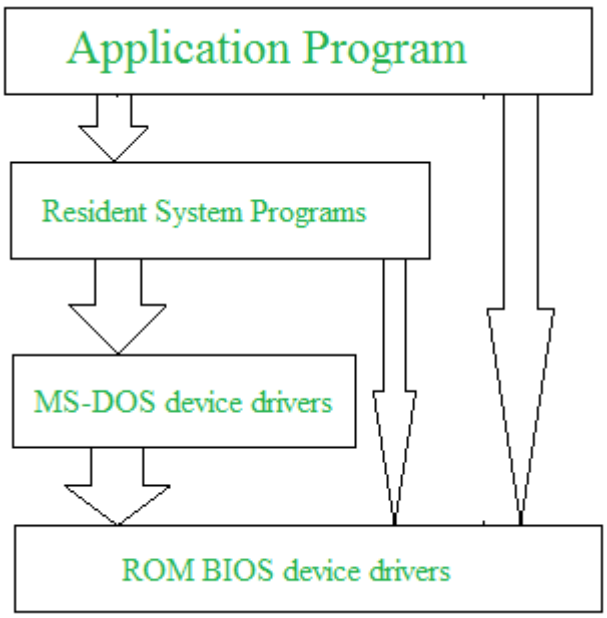
2.Explain about OS structures with neat diagram

Ans.We have the following structures in the operating system:

1. Simple/Monolithic Structure
2. Micro-Kernel Structure
3. Hybrid-Kernel Structure
4. Exo-Kernel Structure
5. Layered Structure
6. Modular Structure
7. Virtual Machines

1. Simple/Monolithic structure

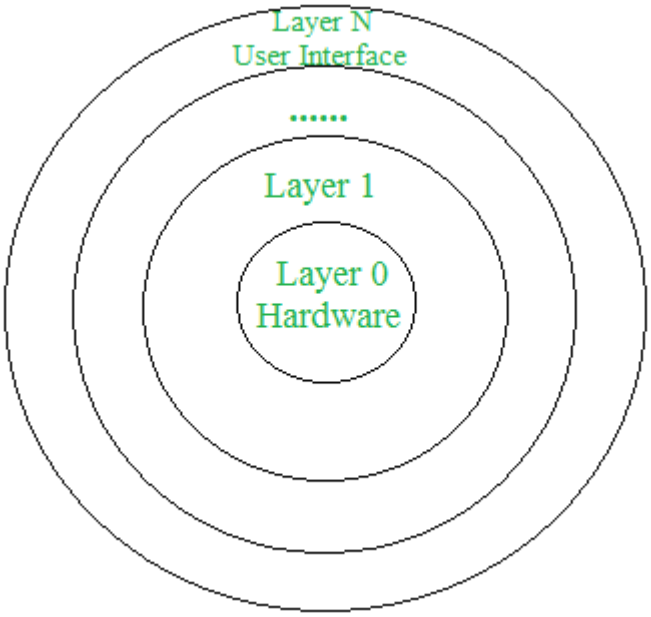
Such operating systems do not have well-defined structures and are small, simple, and limited. The interfaces and levels of functionality are not well separated. MS-DOS is an example of such an operating system. In MS-DOS, application programs are able to access the basic I/O routines. These types of operating systems cause the entire system to crash if one of the user programs fails.

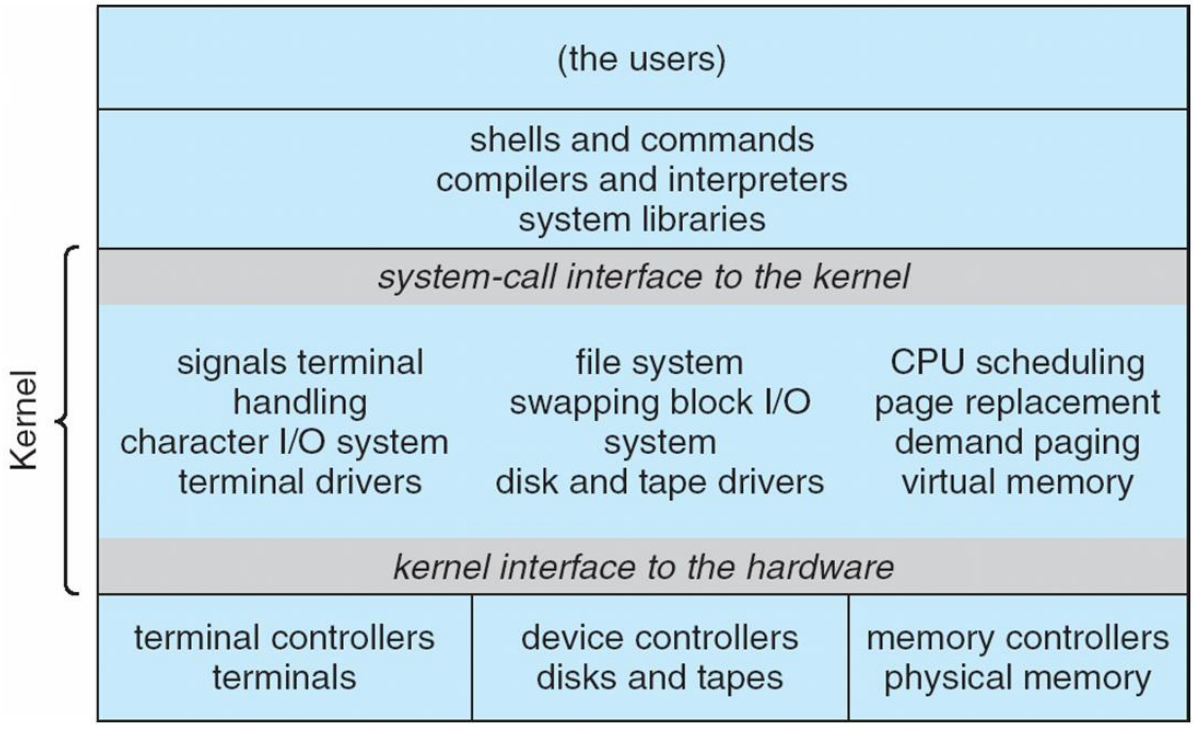


2. Layered structure

An OS can be broken into pieces and retain much more control over the system. In this structure, the OS is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers. This simplifies the debugging process, if lower-level layers are debugged and an error occurs during debugging, then the error must be on that layer only, as the lower-level layers have already been debugged.

The main disadvantage of this structure is that at each layer, the data needs to be modified and passed on which adds overhead to the system. Moreover, careful planning of the layers is necessary, as a layer can use only lower-level layers. UNIX is an example of this structure.

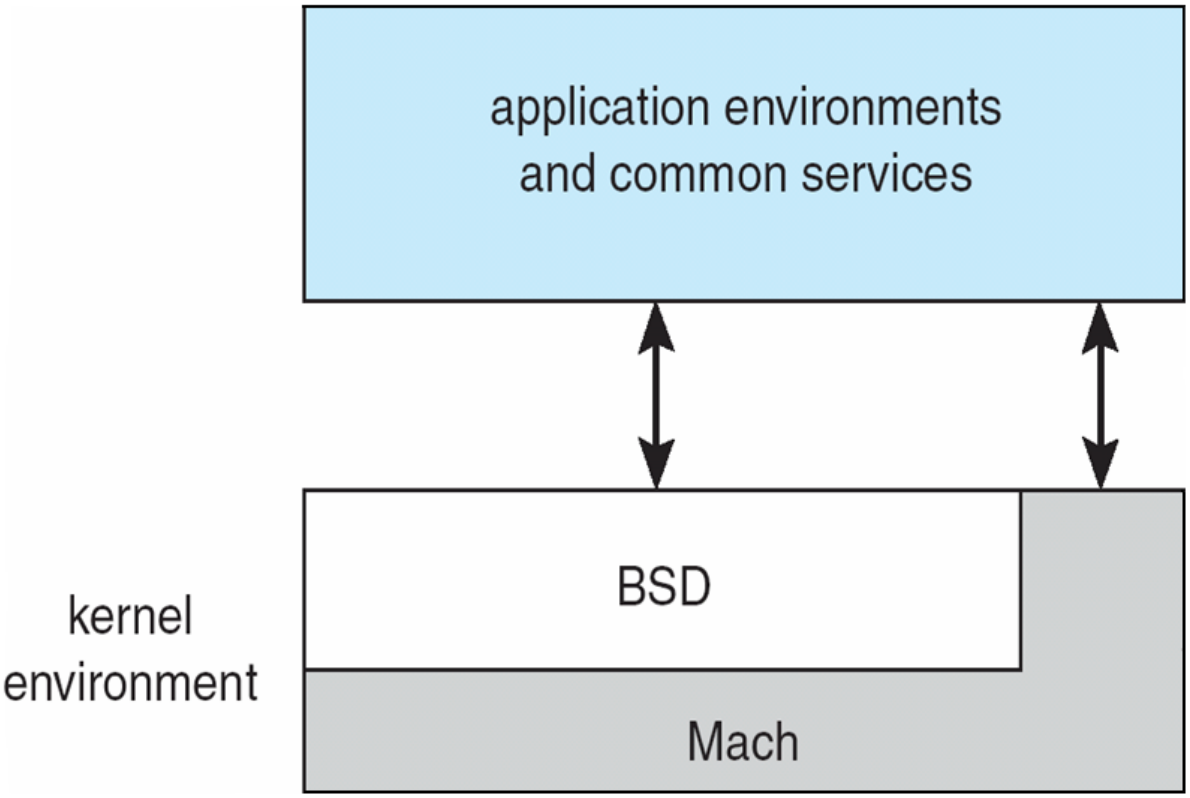




UNIX System Structure (nen lite theeskuntunna)

3. Microkernel System Structure

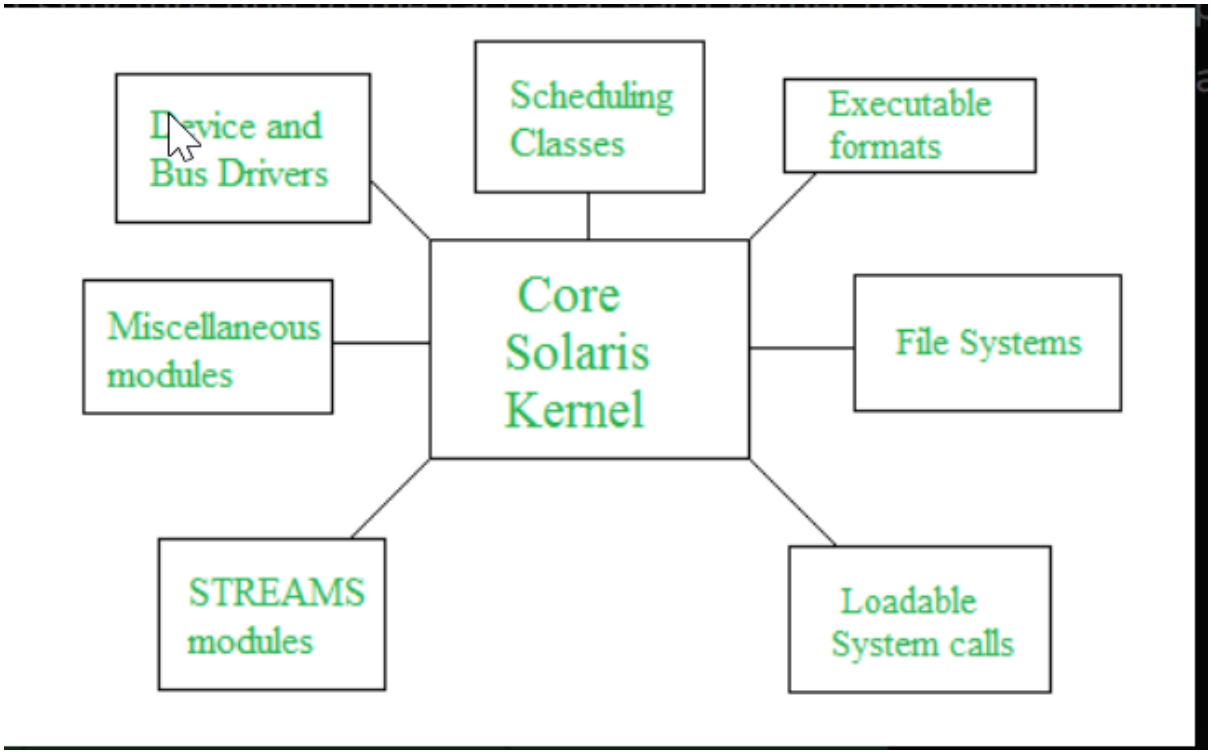
This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs. This results in a smaller kernel called the micro-kernel. Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified. Thus it is more secure and reliable as if a service fails, then rest of the operating system remains untouched. Mac OS is an example of this type of OS.



Mac OS X Structure

4. Modular Structure

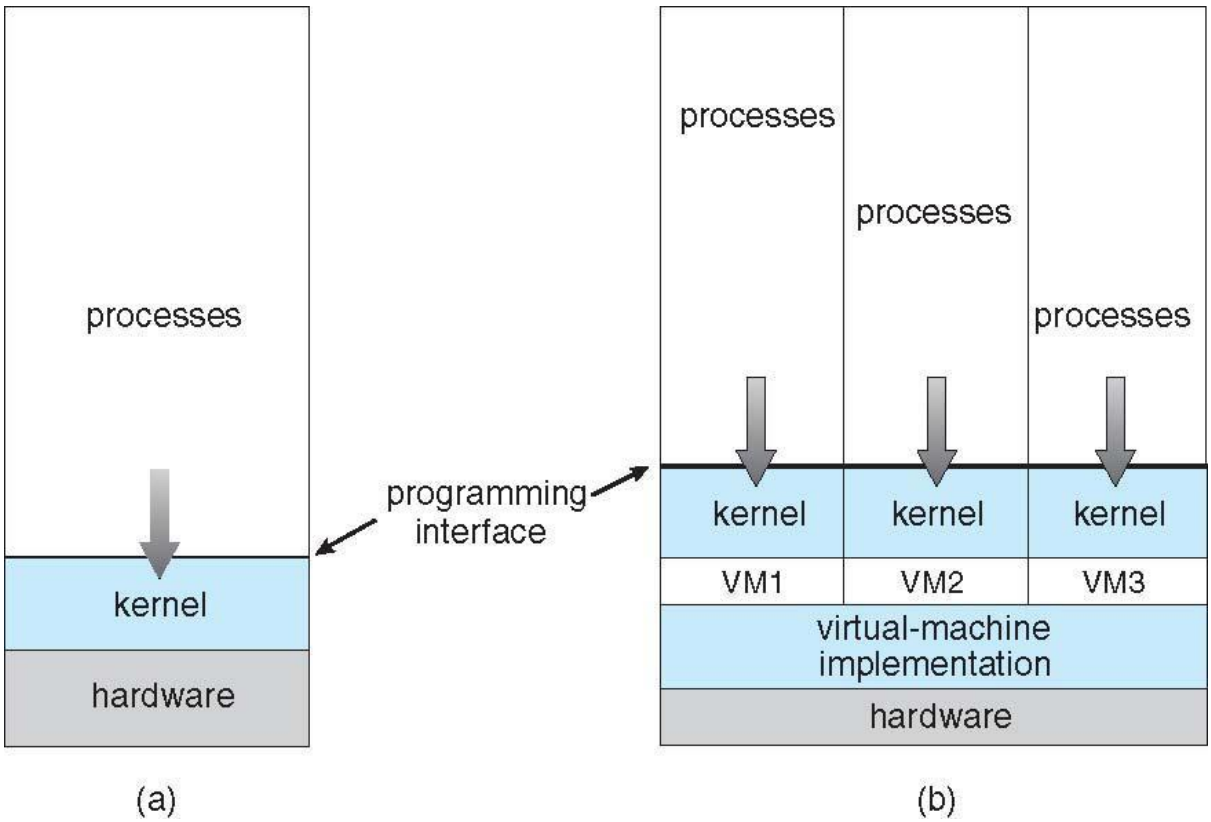
It is considered as the best approach for an OS. It involves designing of a modular kernel. The kernel has only a set of core components and other services are added as dynamically loadable modules to the kernel either during runtime or boot time. It resembles layered structure due to the fact that each kernel has defined and protected interfaces, but it is more flexible than a layered structure as a module can call any other module. For example Solaris OS is organized as shown in the figure.



5. VMs (virtual machines)

- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments.
- Create the illusion that each separate environment is running on its own private computer.
- Virtual machine implementations involve several components.
    - At the base is the host, the underlying hardware system that runs the virtual machines.

The virtual machine manager (VMM) (also known as hypervisor) creates and runs virtual machines.



(a) Nonvirtual machine (b) virtual machine

3. What is process and Explain about various fields of PCB

- A process is a program in execution.
- A program by itself is not a process. A program is a ***passive*** entity, such as a file containing a list of instructions stored on disk (often called an **executable file**).
- In contrast, a process is an ***active*** entity, with a program counter specifying the next instruction to execute and a set of associated resources.
- A program becomes a process when an executable file is loaded into memory.

Process Control Block

1. **Pointer:** It is a stack pointer that is required to be saved when the process is switched from one state to another to retain the current position of the process.
2. **Process state:** It stores the respective state of the process.
3. **Process number:** Every process is assigned a unique id known as process ID or PID which stores the process identifier.
4. **Program counter:** It stores the counter,**:** which contains the address of the next instruction that is to be executed for the process.
5. **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
6. **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
7. **Open files list :** This information includes the list of files opened for a process.

4. Define System Calls and various types of system calls.

Ans. The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

Types of System Calls

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

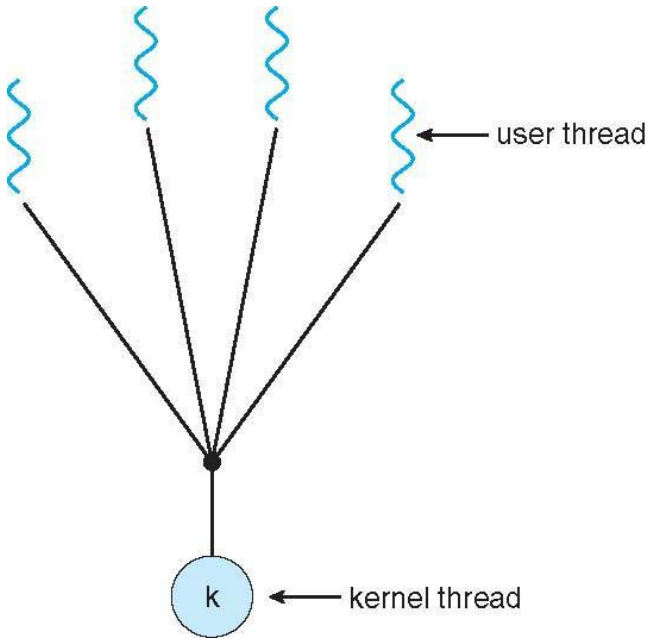
Protection

These system calls manage access to system resources, such as setting file security or changing the ownership of a file.

5. Explain about various Multithreading Models

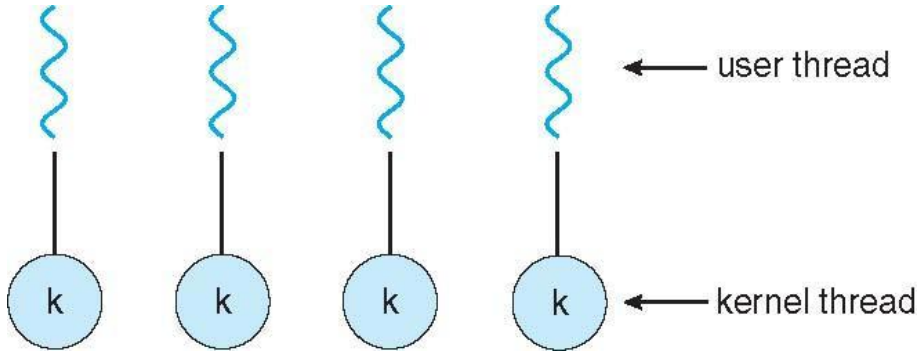
1. Many-to-One:

- Many User-threads are mapped to a single kernel-thread.
- If a single thread is blocked, then the entire operation will be blocked and even kernel thread is also going to get blocked .
- One Kernel thread is responsible to manage all the operations of many user threads.



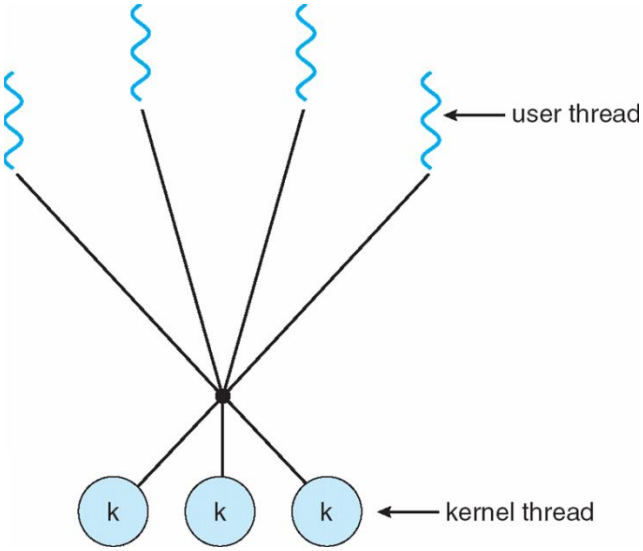
2. One-to-One:

- Each user-thread is mapped to each kernel-thread.
- If one thread is blocked, the another thread will run the application with the remaining kernel-threads.



Many-to-Many:

- One Kernel-thread will be associated with one or more user threads.
- In this model, if one user-thread is blocked, then the other user-thread may be handed over to the other kernel-thread and continue the execution of that particular task. Many-to-Many model provides more concurrency to execute multiple programs



6. Explain in detail about scheduling criteria and scheduling algorithms.

Ans.

CPU Scheduling Criteria

Scheduling criteria refer to the factors that are used to evaluate and prioritize processes or threads for execution.

The criteria include the following:

- **CPU utilization:** The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible.
- **Throughput:** A measure of the work done by the CPU is the number of processes being executed and completed per unit of time.
- **Turnaround time:** The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time.

Turn Around Time = Completion Time – Arrival Time.

- **Waiting time** – amount of time a process has been waiting in the ready queue.

Waiting Time = Turnaround Time – Burst Time

- **Response time** – Time from the submission of rquest until the first response is produced (or) amount of time it takes from when a request was submitted until the first response is produced, not output.

Response Time = CPU Allocation Time(when the CPU was allocated for the first) – Arrival Time

(ivi extra anamaata)

1. **Priority**: The relative importance or urgency of a process or thread.

2. **Burst Time**: The amount of time a process or thread requires to complete its execution.
3. **Arrival Time**: The time at which a process or thread arrives in the system.
4. **Deadline**: The time by which a process or thread must complete its execution.

Scheduling Algorithms:

Scheduling algorithms are used to select the next process or thread to execute based on the scheduling criteria. The most common scheduling algorithms are:

1. First Come First Serve:

FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.

2. Shortest Job First(SJF):

Shortest job first (SJF) is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed. The full form of SJF is Shortest Job First.

3. Priority Scheduling:

Priority CPU Scheduling Algorithm is a method of CPU scheduling algorithm that works **based on the priority** of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there is more than one process with equal value, then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

4. Round robin:

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of First come First Serve CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

5. Multilevel Queue Scheduling:

Processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a **foreground (interactive)** process and a **background (batch)** process. These two classes have different scheduling needs. For this kind of situation **Multilevel Queue Scheduling** is used.

6. Multilevel Feedback Queue Scheduling::

Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling is like **Multilevel Queue Scheduling** but in this process can move between the queues. And thus, much more efficient than multilevel queue scheduling.

8. What is a semaphore? Explain various types of semaphores.

- Ans. Semaphore is an integer variable that is shared by multiple processes.
- It is a software solution for critical section problem which is used to achieve process synchronization in the multiprocessing environment.
- It can be accessed by two standard atomic operations :

wait (S) & Signal (S)

wait (S):

It is a simple atomic operation that decrements the value of S by 1.

Signal (S):

It is a simple atomic operation that increases the value of S by 1.

Binary Semaphore

- It can range only between 0 and 1
- It provides mutual exclusion i.e., the reason it is also called as Mutex locks.
- Can implement a counting semaphore S as a binary semaphore

Counting Semaphore

- It can range over an unrestricted domain.
- It is used to control access to a resource that has multiple instances.

9.Explain the bounder buffer (or producer consumer problem) and how semaphores can be used as a solution to this problem.

Ans.

- It is the Producer Consumer problem.
- Producer put the items into the buffer.
- Consumer will consume/take the items from the buffer.

Here the problems are

- Producer cannot produce any item if the buffer is full
- If the buffer is empty the consumer cannot consume anything.
- The producer and consumer should not insert and remove data simultaneously
- This bounded buffer problem is solved by using the semaphore values i.e., wait and signal.
- In this problem, the producer and consumer processes share the following data structures:

semaphore mutex = 1;

semaphore empty = n;

semaphore full = 0

- **The structure of the producer process:**

```
do
{
    wait (empty);//wait until
                    empty>0
    wait (mutex);//acquire lock

  //  add the item to the  buffer

      signal (mutex); //release lock
      signal (full); //++ full

   } while (TRUE);
```

- **The structure of the consumer process:**

```
  do
{
          wait (full); //wait until
                    full>0
      wait (mutex); //acquire lock

 //  remove an item from buffer

          signal (mutex); //release lock
          signal (empty); //++ empty

    } while (TRUE);
```

10. Define deadlock. What are the 4 necessary conditions for deadlock?

Ans. **Deadlock** is **a situation in which two or more processes are blocked indefinitely, each waiting for the other to release a resource**. This occurs when each process is holding a resource and waiting

for another resource that is held by another process, and there is no way for either process to proceed.

The four necessary conditions for deadlock to occur are:

- **Mutual Exclusion**: Two or more resources are non-shareable, meaning that only one process can use them at a time.
- **Hold and Wait**: A process is holding at least one resource and waiting for another resource that is held by another process.
- **No Preemption**: A resource cannot be taken from a process unless the process releases the resource.
- **Circular Wait**: A set of processes waiting for each other in a circular form, where each process is waiting for a resource held by another process in the chain.

These four conditions must be met simultaneously for a deadlock to occur.

11. Discuss in detail about deadlock prevention methodology.

Ans. We can prevent a Deadlock by eliminating any of the above four conditions.

Eliminate Mutual Exclusion: It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Eliminate Hold and wait: Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization.

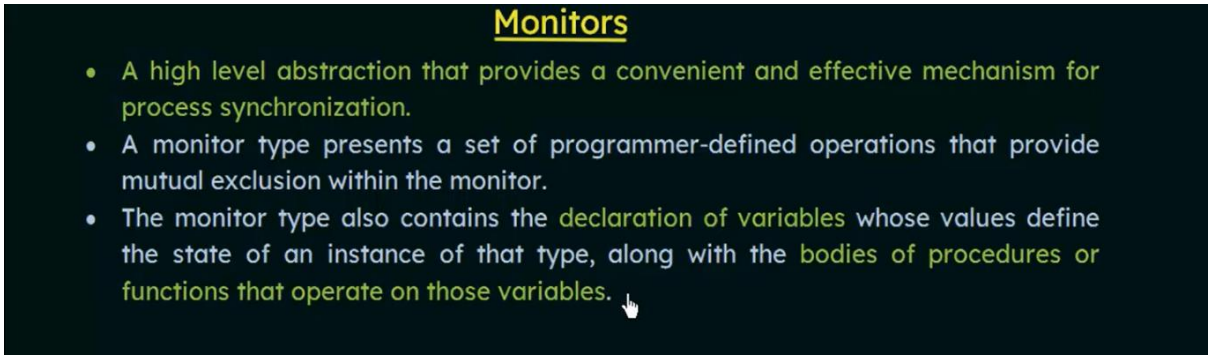
For example, if a process requires a printer at a later time and we have allocated a printer before the start of its execution printer will remain blocked till it has completed its execution. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.

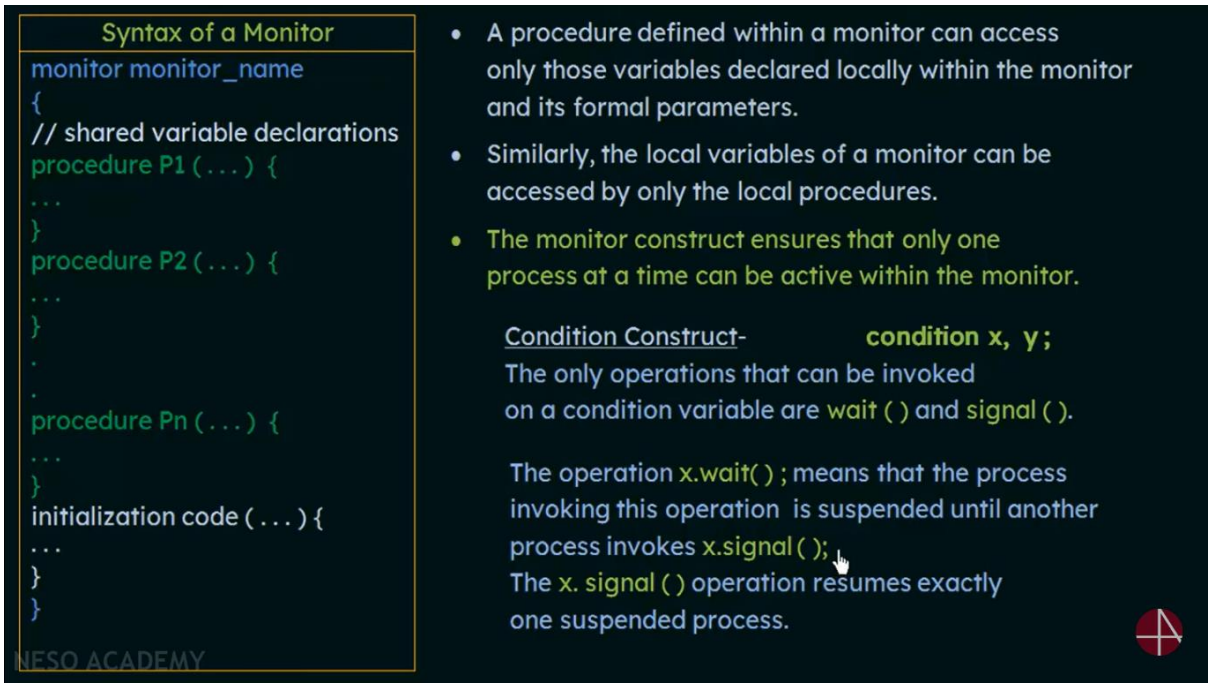
Eliminate No Preemption : Preempt resources from the process when resources are required by other high-priority processes.

Eliminate Circular Wait : Each resource will be assigned a numerical number. A process can request the resources to increase/decrease. order of numbering.

For Example, if the P1 process is allocated R5 resources, now next time if P1 asks for R4, R3 lesser than R5 such a request will not be granted, only a request for resources more than R5 will be granted.

13. Explain in detail about Monitors.





15. What are the main functions of the OS in connection with Memory management.

Ans. The main functions of the Operating System (OS) in connection with Memory management are:

- **Memory Allocation**: The OS manages the allocation of memory to different processes or programs, ensuring that each process has the necessary memory to run efficiently.
- **Memory Deallocation**: The OS deallocates memory when a process or program is terminated or completes its execution, freeing up memory for other processes or programs.
- **Memory Protection**: The OS provides memory protection by preventing one process from accessing or modifying the memory of another process, ensuring that each process runs in its own isolated environment.

- **Memory Fragmentation Management**: The OS manages memory fragmentation by allocating and deallocating memory in a way that minimizes fragmentation, ensuring that memory is used efficiently.
- **Memory Swapping**: The OS swaps memory pages between main memory and secondary storage (hard disk) to manage memory usage and optimize performance.
- **Memory Protection and Error Handling**: The OS detects and handles memory-related errors, such as page faults, segmentation faults, and bus errors, to ensure system stability and reliability.
- **Memory Optimization**: The OS optimizes memory usage by allocating and deallocating memory based on the needs of running processes, minimizing memory waste and maximizing system performance.
- **Memory Monitoring**: The OS monitors memory usage and performance, providing tools and metrics to help system administrators and developers optimize memory usage and troubleshoot memory-related issues.

16. Explain in detail about Reader – writer problem. How semaphores can be used as a solution to this problem.

Ans.

- A data set is shared among several concurrent processes
- Readers – can only read the data set; they do **not** perform any operation
- Writers – writer process can write the data.

- **If reader is in the critical section**, we can allow other readers to enter the critical section.

But we need to make sure that the writer is not inside the critical section.

- **If writer is inside the critical section**, they don't allow the other reader to enter into the critical section.

And even they don't allow the writer also to enter into the critical section.

- At the time of **reading, no writer is allowed**.
- At the time of **writing, no reader or another writer is allowed.**

Solution for Readers-Writers problem using Semaphores

We will make use of two semaphores and one integer variable

1. Semaphore mutex initialized to 1 which is used to ensure mutual exclusion when readcount is updated i.e when any reader enters or exit from the critical section
2. Semaphore wrt initialized to 1 common to both reader and writer

3. Integer readcount initialized to 0 that keeps track of how many processes are currently reading the object

12. Find the safe state by following the snapshot which contains P0 – P4, resource type A,B,C (Banker's Algorithm)

(or) Explain in detail about safety algorithm with example.

Ans.

Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

   Work* = *Available

   ***Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both:

   (a) ***Finish* [*i*] = *false***

   (b) $Need_i \leq Work$

   If no such ***i*** exists, go to step 4

3. $Work = Work + Allocation_i$

   Finish*[*i*] = *true

   go to step 2

4. If ***Finish* [*i*] == *true*** for all ***i***, then the system is in a safe state

Example of Banker's Algorithm

- 5 processes P_0 through P_4;
  3 resource types:
  A (10 instances), B (5instances), and C (7 instances)
- Snapshot at time T_0:

	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	

Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	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

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0 >$ satisfies safety criteria

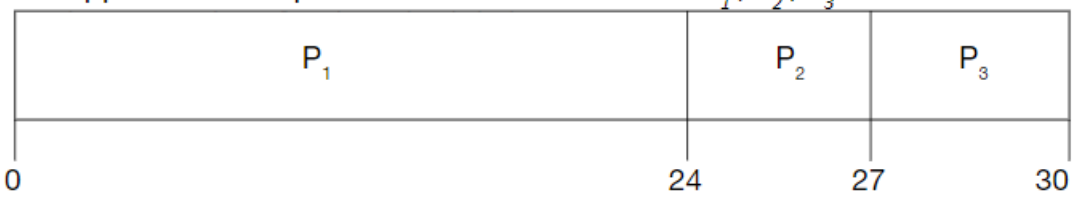
5. Explain in detail about sheduling algorithms with examples

First-Come, First-Served (FCFS) Scheduling

The process that requests the cpu first is allocated the cpu first.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3



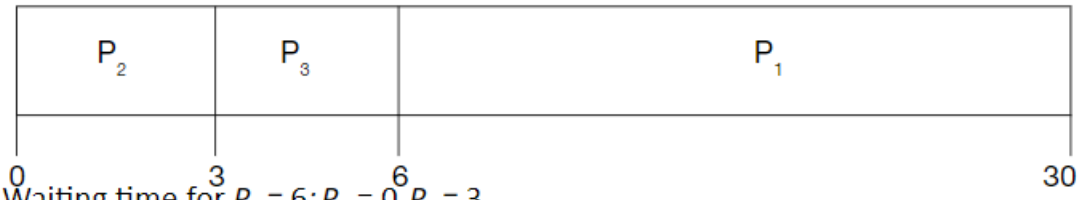
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for P_1 = 6; P_2 = 0 P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- There is a **convoy effect** as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization.
- FCFS scheduling algorithm is nonpreemptive.
- FCFS is not well suited for time sharing environment.

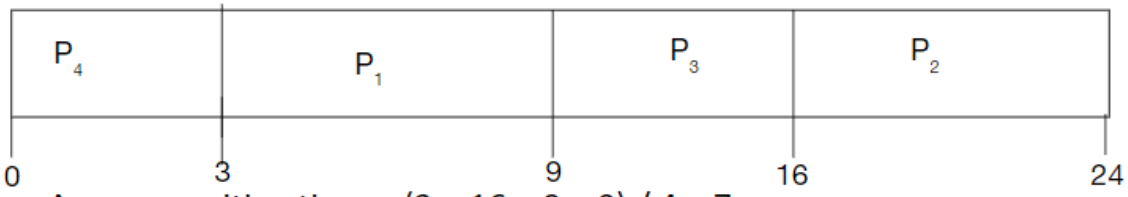
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request.

Example of (SJF) Scheduling

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart

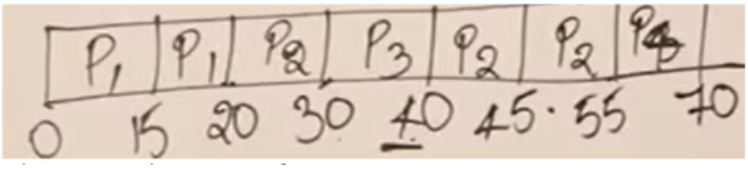


- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7
- By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds.

Shortest-Job-First (SJF) Scheduling

- The SJF algorithm can be either preemptive or nonpreemptive.
- The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process.
- A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.

PNO	AT	BT
1	0	20
2	15	25
3	30	10
4	45	15



SRTF

Shortest remaining time first:

PNO	AT	BT
1	0	7
2	1	5
3	2	3
4	3	1
5	4	2
6	5	1

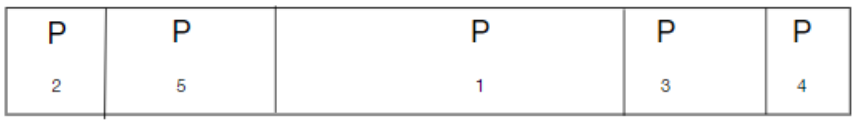
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



0 1 6 16 18 19

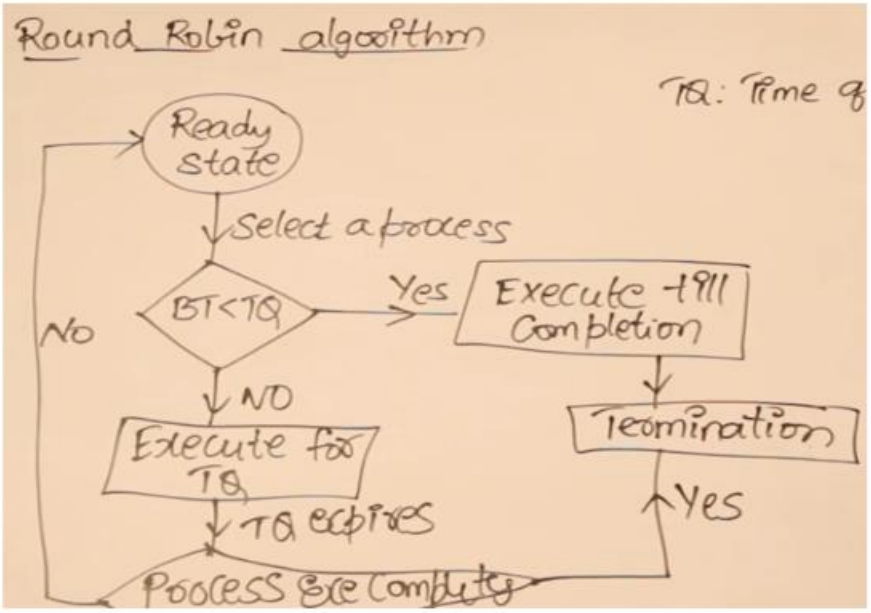
- Average waiting time = (6+0+16+18+1)/5= 8.2 msec

P NO	Priority	AT	BT
1	2	0	4
2	4	1	2
3	6	2	3
4	10	3	5
5	8	4	1
6	12	5	4
7	9	6	6

Round Robin Scheduling

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes.
- A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from 10 to 100 milliseconds in length. The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process
- Performance
  - q large ⇒ FIFO
  - q small ⇒ q must be large with respect to context switch, otherwise overhead is too high

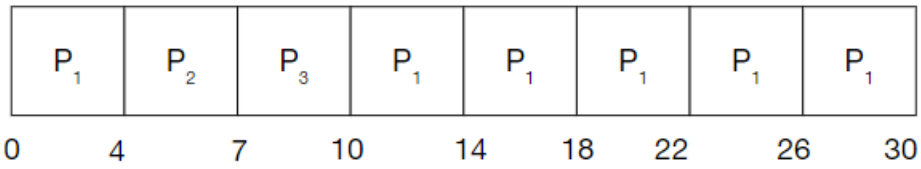
Round Robin Scheduling



Example of RR with time quantum=4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:

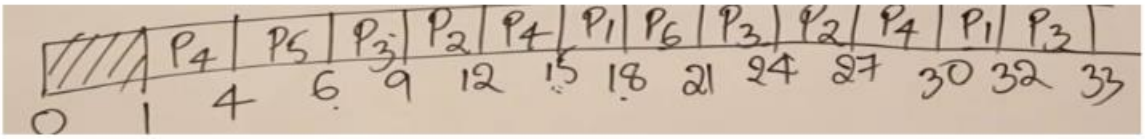


Let's calculate the average waiting time for this schedule.
WT of P1 = TAT – BT = 30-24 = 6
WT of P2 = 7-3 = 4
WT of P3 = 10-3 = 7
Thus, the average waiting time is 17/3 = 5.66 milliseconds

Round Robin Scheduling

TQ=3

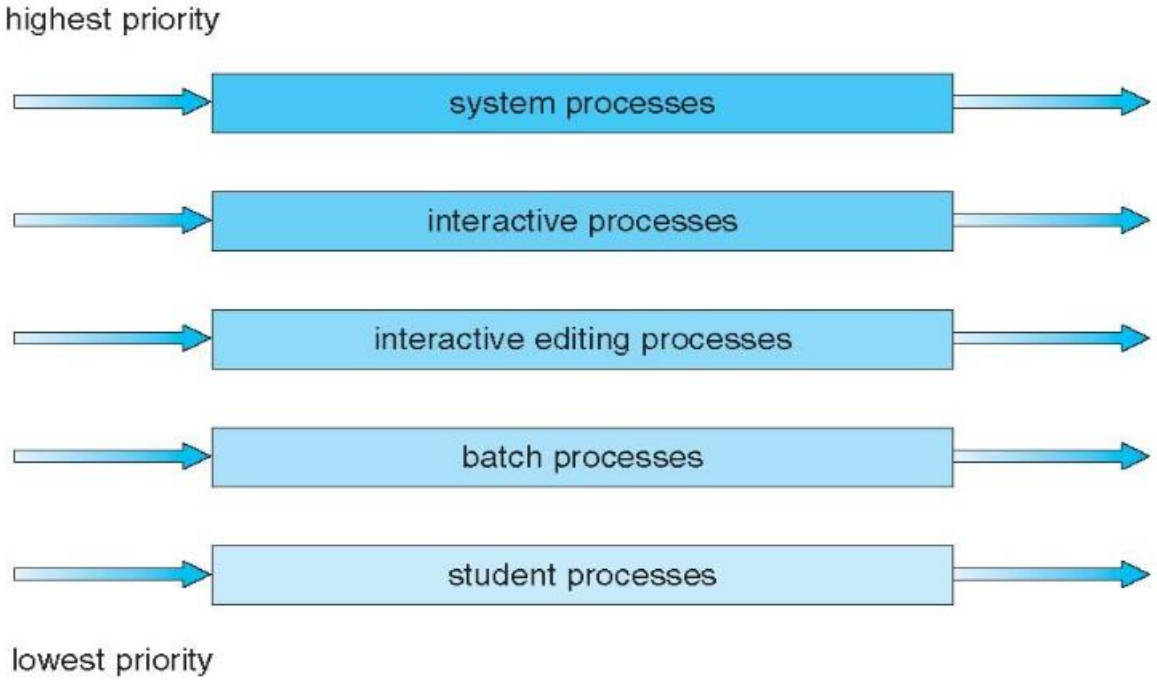
PNO	AT	BT
1	5	5
2	4	6
3	3	7
4	1	9
5	2	2
6	6	3



Multilevel Queue Scheduling

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue. 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.

Multilevel Queue Scheduling



Multilevel Feedback Queue Scheduling

- The multilevel feedback queue scheduling algorithm, allows a process to move between queues.
- The idea is to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.
- This form of aging prevents starvation.
- 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

Example of Multilevel Feedback Queue Scheduling

- A process entering the ready queue is put in queue 0.
- A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.