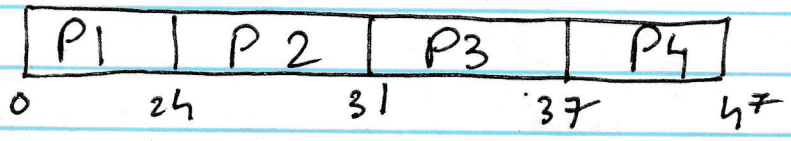
Assignment 01 :

Q1)

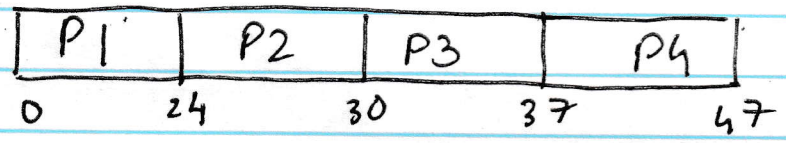
Process	Burst time	Arrival time	Priority
P1	24	0	5
P2	7	3	3
P3	6	5	2
P4	10	10	1

i) Gantt chart : FCFS, SJF, Priority (Preemptive & Round Robin (Quantum = 4)

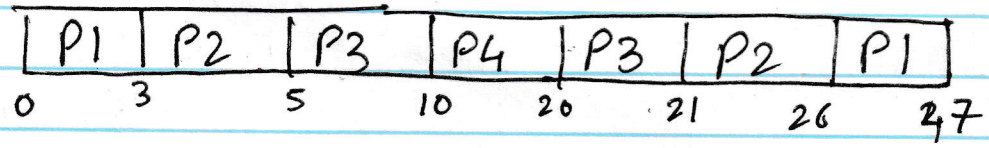
A) FCFS :



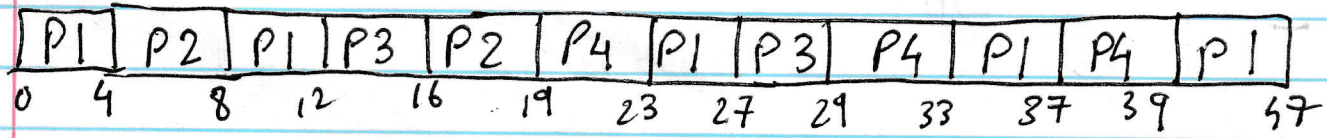
B) SJF:



c) Priority (Preemptive) :



D) Round Robin



ii) Average Waiting time:

A) FCFS : 18.5 ms

B) SJF : 18.25 ms

C) Priority (Preemptive) : 12.5 ms

D) Round Robin (TQ=4) : 17.25 ms

iii) Turnaround Time:

A) FCFS : 30.25 ms

B) SJF : 30 ms

C) Priority : 24 ms

D) Round Robin : 29 ms.

Q2) Requirements of Mutual Exclusion:

- If many processes want to execute in the critical section then only one process should be allowed to execute in that critical section for the same resources or shared object.
- If the process halts in its remainder section (other than its critical section) then its allowed to do so without ~~interpreting~~ interfering with other processes.

- If a process is waiting to enter in critical section then it should not be delayed for an indefinite period that is should not lead to deadlock & starvation

- If the critical section is empty any process not executing in critical section, then if any process that requests entry to its critical section must be allowed to enter without delay.

- Assumption about relative process speed or number of processes are not made

- Any process will not remain inside its critical section per indefinite time. It should stay there for a finite time only.

Semaphore Usage:

- Binary Semaphores behave similarly to mutex locks. In fact, on systems that do not provide mutex locks, binary semaphores can be used instead for providing mutual exclusive.

- Counting semaphores can be used to control access to a given resource consisting of a finite number of instances.

- The semaphore is initialized to the number of resources available.

- Each process that wishes to use a resources performs a wait() call on the semaphore.

- When that process releases that resource it calls signal() on the semaphore.

- This is how mutual exclusion is provided by semaphores.

Q3) Kernel:

- It is the core of an operating system
- Whenever a system starts, the kernel is the first program that is loaded after the bootloader because the kernel has to handle the rest of the things of the system for the OS.

- It remains in the main memory until the OS shuts down
- It is responsible for low-level tasks such as disk management, memory management, task management etc.
- Provides an interface between the user & the hardware components of the system
- It contains most frequently used functions

Kernel Designing:

1) Monolithic Approach:

- Here there is no specific structure for the modules.

- No restrictions on module calls
- Delivers good performance as there are very few interfaces between the application program & the underlying hardware.
- It is extremely difficult to enhance or maintain - modifications to one module may adversely affect other modules.
- Errors are more difficult to isolate & high risk of damage.

2) Layered Approach:

- OS is broken into number of layers
- The bottom layer is the hardware & the top layer is the user interface.
- Construction is simple & bugs are easy to catch.
- Layers are arranged in a way that each layer uses functions & services of only the lower layers
- Verification is simplified
- Layer definition is difficult

3) Microkernel Approach:

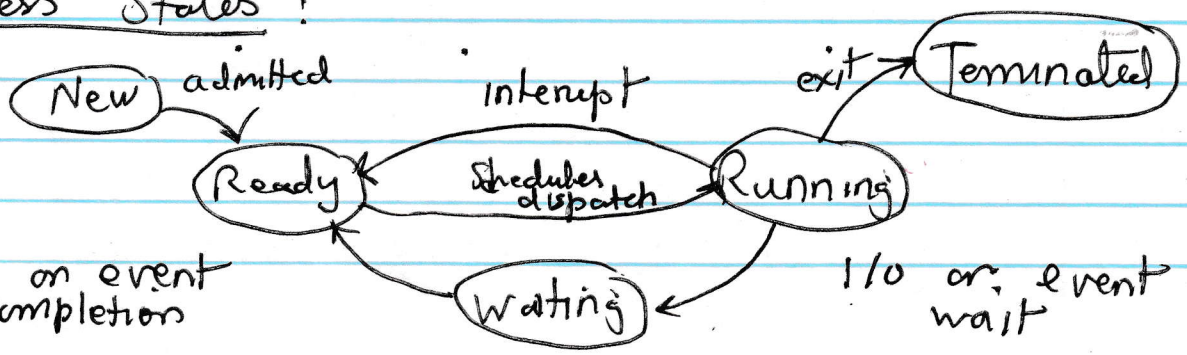
- The kernel only provides only the most essential OS functions - process management communication primitives & low level memory management.
- System programs or user level programs are implemented outside the kernel.

- These are called ~~servers~~ severs.
- Hence, the size of the kernel is reduced by a huge margin.

Q4) Process Management:

- In a multi-programming environment, single CPU is shared among multiple processes.
- If many processes remain busy in completing I/O, CPU is allocated to any one process at a given point of time.
- Here, some policy is required to allocate CPU to process, called as CPU scheduling.
- If multiple users are working on a system, OS switches the CPU from one ~~proc~~ process to another.
- Process communication, deadlock handling, suspension & resumption of processes & creation & ~~deletion~~ deleting of the process are the activities performed in Process management.

Process States:



- During execution, process may change its state. Process state shows the current activity of the process. Process state contains five states

- Each process remains in one of the five states. There is a queue associated with each state of the process:

(1) New State: A new process being created

(2) Ready State: A process is ready to ~~run~~ run but is waiting for CPU to be assigned.

(3) Running State: A process has been allocated CPU

(4) Waiting State: A process can't continue the ~~execution~~ ~~be~~ ~~can~~ because it is waiting for event to happen such as I/O completion

(5) Terminated state: The process has completed execution