Operating Systems

Quiz

Q. Which of the following statement/s is/are false about a process?

A. Process is a running entity of a program

B. Program in main memory is referred as a process

C. One program may has multiple running instances i.e. processes.

D. Program in execution is referred as a process.

E. All of the above

F. None of the above

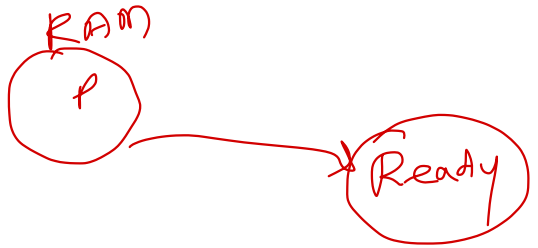
Q. Process which is in the main memory waiting for the CPU time considered in a _____.

A. waiting state

B. new state

C. ready state

D. running state



Quiz

Q. _____ copies an execution context of a process which is scheduled by the scheduler from its PCB and restores it onto the CPU registers.

A. Loader

B. Interrupt Handler

C. Dispatcher

D. Job Scheduler

Q. In a timeshare operating system, when the time slot assigned to a process is completed, the process switches from the current state to?

A. Suspended state

B. Terminated state

C. Ready state

D. Blocked state

Quiz

Q. CPU scheduling is the basis of _____

a) multiprogramming operating systems

b) larger memory-sized systems

c) multiprocessor systems

d) none of the mentioned

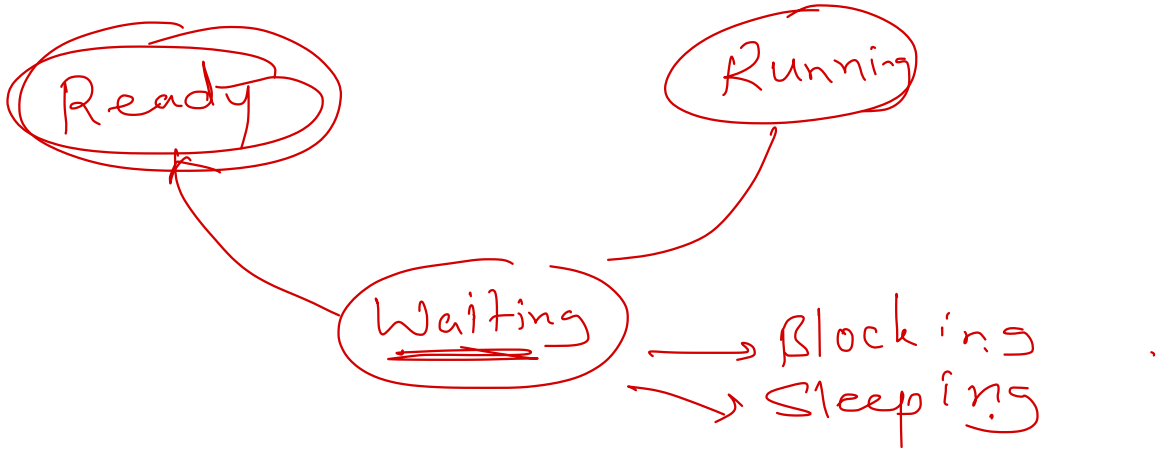
Q. When a process is in a “Blocked” state waiting for some I/O service. When the service is completed, it goes to the _____

a) Terminated state

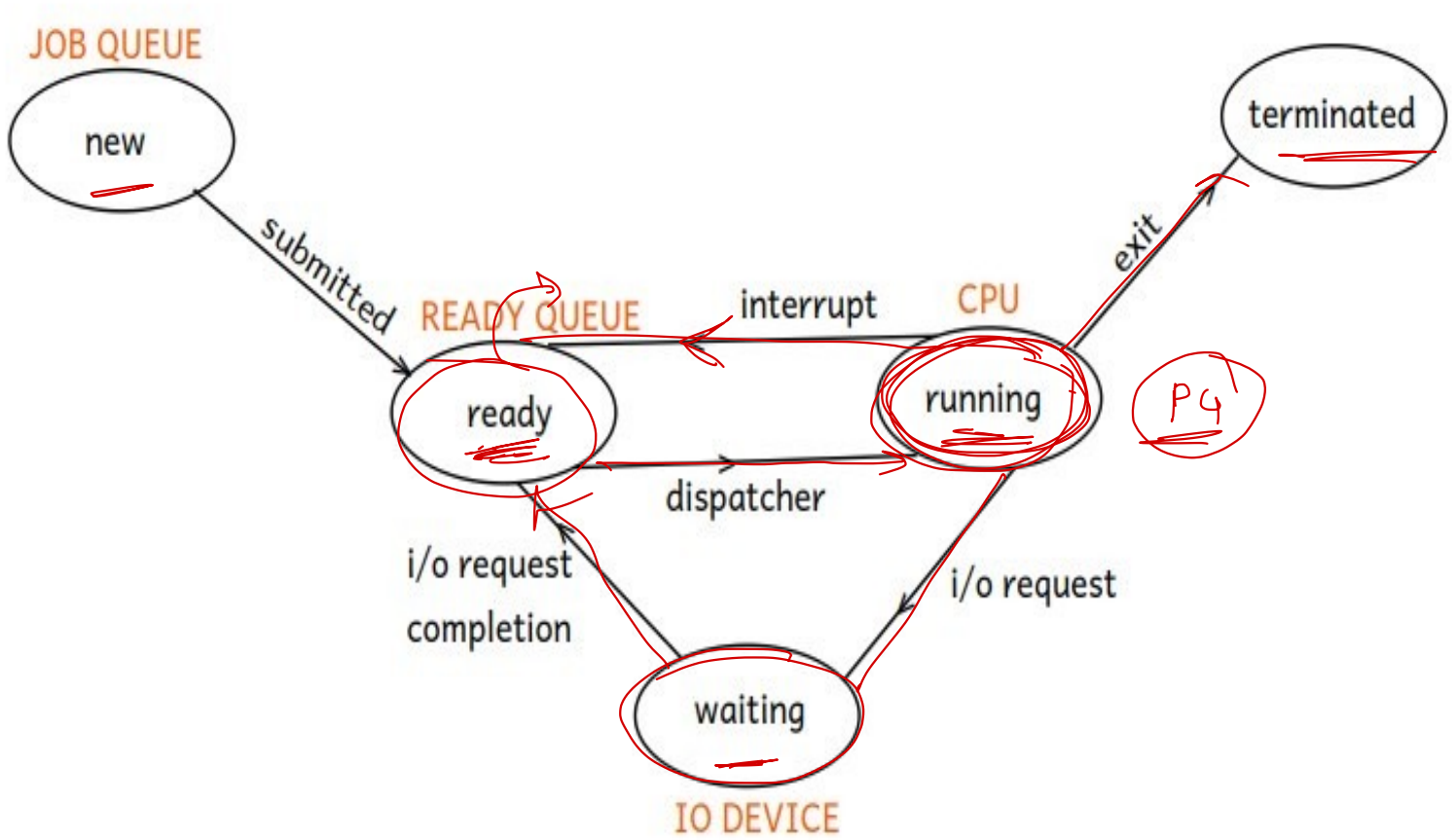
b) Suspended state

c) Running state

d) Ready state



Process State Diagram



CPU scheduler is invoked

1. Running -> Terminated
2. Running -> Waiting
3. Running -> Ready
4. Waiting -> Ready

CPU Scheduling Criteria

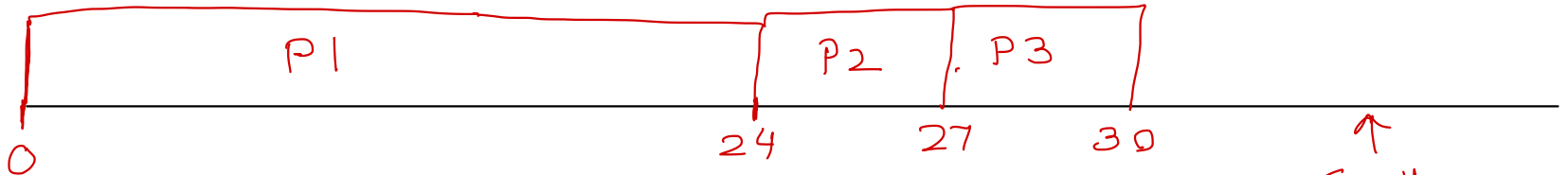
Scheduling criteria

- **CPU utilization**: Ideal - max
  - On server systems, CPU utilization should be more than 90%.
  - On desktop systems, CPU utilization should around 70%.
- **Throughput**: Ideal - max
  - The amount of work done in unit time.
- **Waiting time**: Ideal - min
  - Time spent by the process in the ready queue to get scheduled on the CPU.
  - If waiting time is more (not getting CPU time for execution) -- Starvation.
- **Turn-around time**: Ideal - CPU burst + IO burst – min
  - Time from arrival of the process till completion of the process.
  - CPU burst + IO burst + (CPU) Waiting time + IO Waiting time
- **Response time**: Ideal - min
  - Time from arrival of process (in ready queue) till allocated CPU for first time.

① FCFS → First Come First Serve.

Process	Arrival	CPU Burst	waiting time	turn around time	Response time
1 → P1	0	24	0	24	0
2 P2	0	3	24	27	24
3 P3	0	3	27	30	27

non-pre-emptive



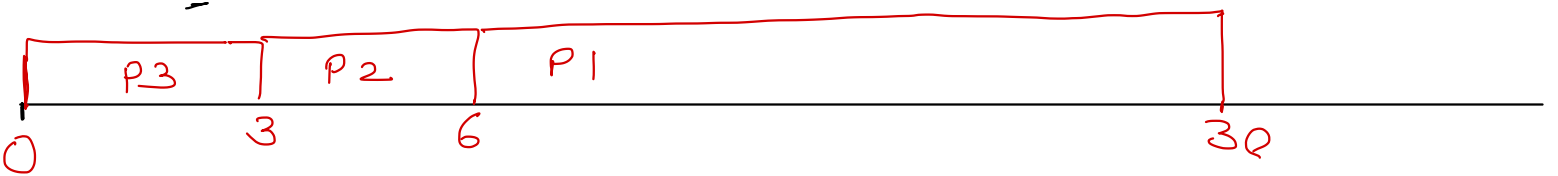
P1
P2
P3

avg waiting time = $\frac{0+24+27}{3} = \frac{17}{}$

avg turnaround time = $\frac{24+27+30}{3} = \frac{27}{}$

FCFS

Process	Arrival	CPU Burst	waiting time
P3	0	3	0
P2	0	3	3
P1	0	24	6



✓P3
✓P2
✓P1

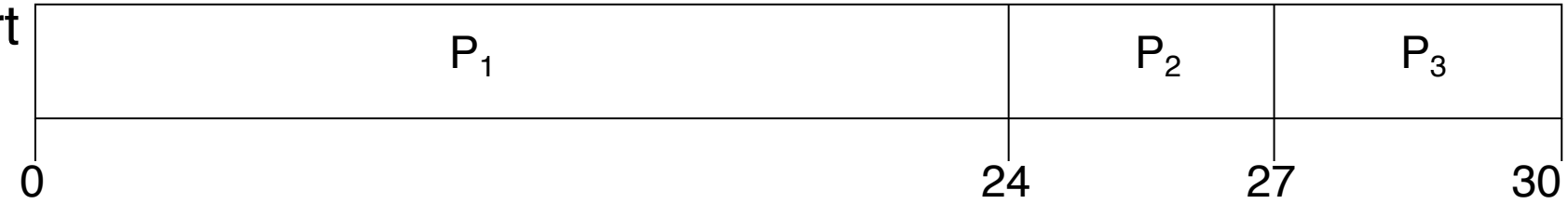
avg waiting time = $\frac{0+3+6}{3} = \frac{3}{=}$

Convey effect → if bigger process comes first than waiting time increases.

First-Come, First-Served (FCFS) Scheduling

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
- The Gantt Chart



Waiting time for P_1 = 0; P_2 = 24; P_3 = 27

- Average waiting time: (0 + 24 + 27)/3 = 17

.

Convoy effect -> If bigger processes arrive first, average waiting time increases

FCFS Scheduling

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

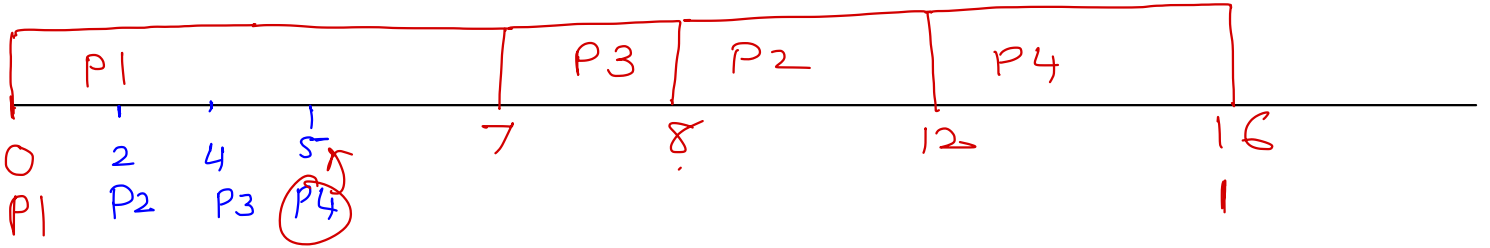
The Gantt chart for the schedule is:

CPU burst time -> p3 = 3, p2 = 3 , p1 = 24
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.
Convoy effect short process behind long process

SJF → Shortest Job first

non-preemptive

Process	arrival	CPU Burst	wait time	turnaround time
P1	0	7	0	7
P2	2	4	8-2=6	10
P3	4	1	7-4=3	4
P4	5	4	12-5=7	11

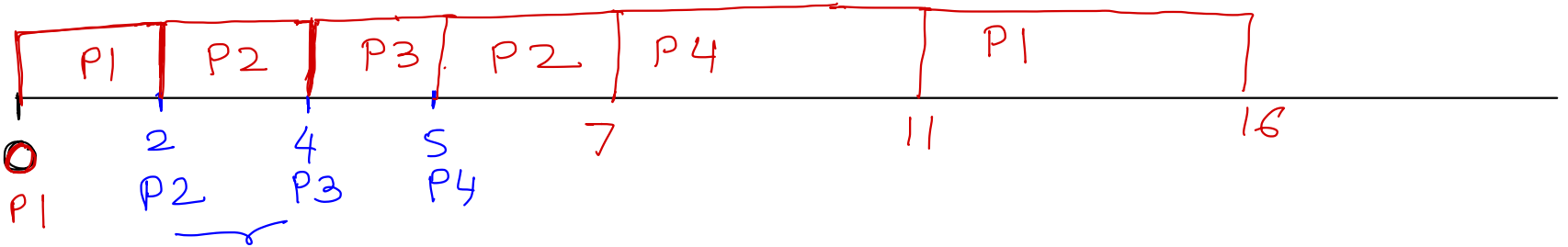


avg waiting time = $\frac{0+6+3+7}{4} = 4$

SRTF → Shortest Remaining time first.

pre-emptive

Process	arrival	CPU Burst	remain time	wait time
P1	0	7	7-2=5	11-2=9
P2	2	4	4-2=2	5-4=1
P3	4	1	1-1=0	0
P4	5	4		7-5=2

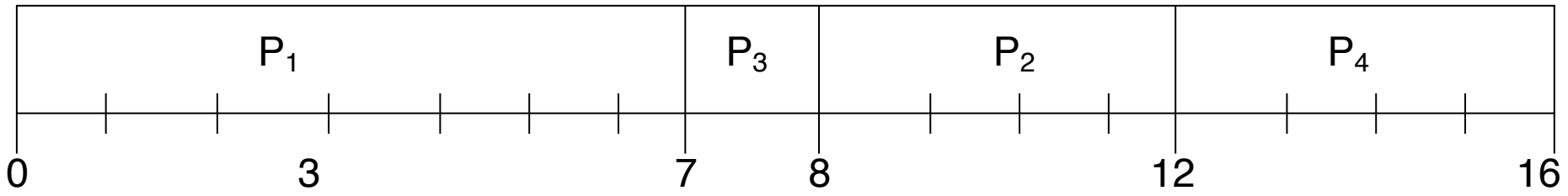


avg waiting time = $\frac{9+1+0+2}{4} = 3$.

Example of SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

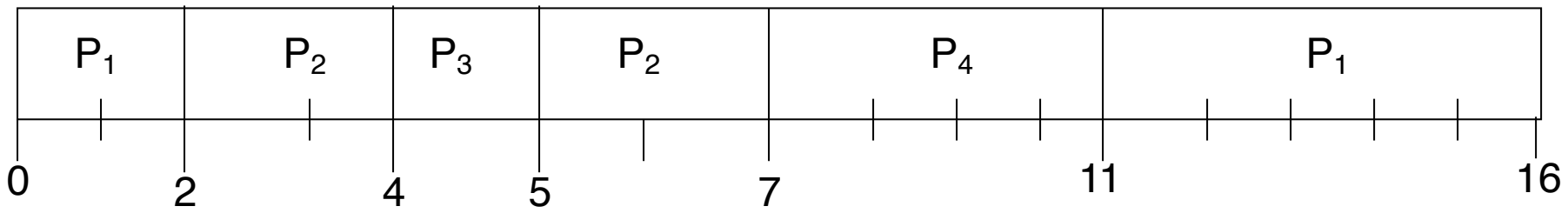
- Average waiting time = (0 + 6 + 3 + 7)/4 = 4
  - *P1 waiting time = 0*
  - *P2 waiting time = 6 (8-2)*
  - *P3 waiting time = 3(7-4)*
  - *P4 waiting time = 7(12-5)*



Example of Preemptive SJF (Shortest Remaining Time First [SRTF])

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

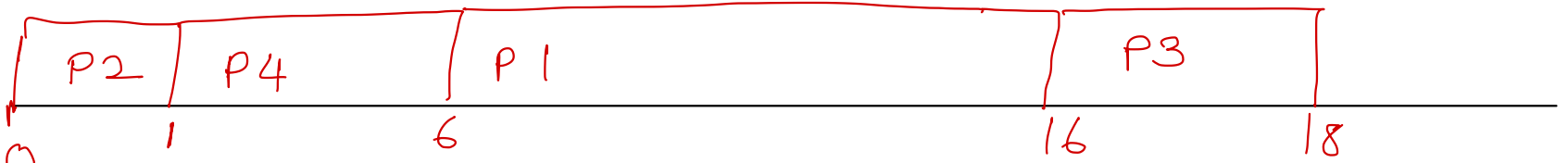
Remaining time = p1 = 5, p2 = 2, p3 =1, p4 = 4
Waiting time = p1 = 9; p2 =1 , p3=0 , p4 = 2
Average waiting time = (9 + 1 + 0 +2)/4 = 3



Priority

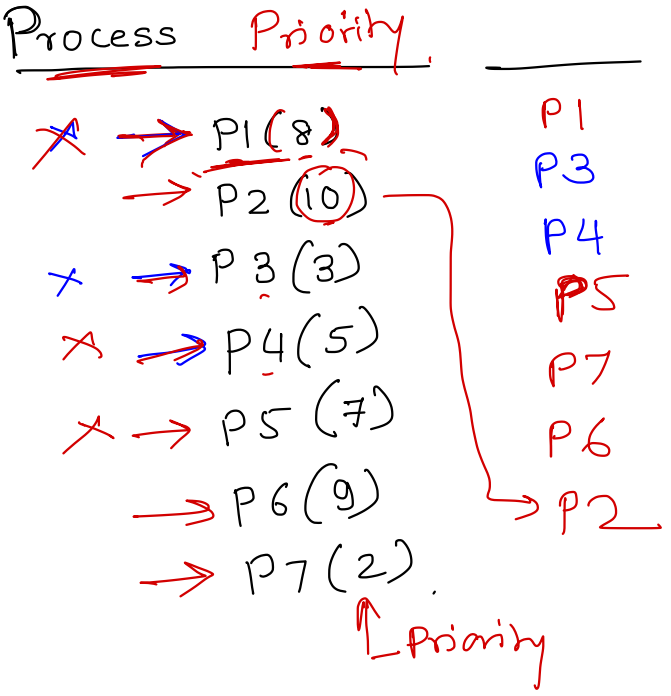
non-pre-emptive

Process	Arrival	CPU Burst Time	Priority	Waiting time
P_1	0	10	3	6
P_2	0	1	1 (high priority)	0
P_3	0	2	4 (low priority)	16
P_4	0	5	2	1



P1
P2
P3
P4

avg waiting time $= \frac{6+0+16+1}{4} = 5.75$



Starvation : A process is not getting enough CPU time for its execution.
Reason :- Low Priority

Soln → Aging :- The process spending long time in ready queue, increase its priority dynamically.

Starvation
⇕
Ageing

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
  - Non-preemptive
- **Starvation** – A process is not enough CPU time for its execution.(waiting in the ready queue)
- Solution of starvation is Aging – The process spending long time in ready queue, increase its priority dynamically.

Example of Priority Scheduling

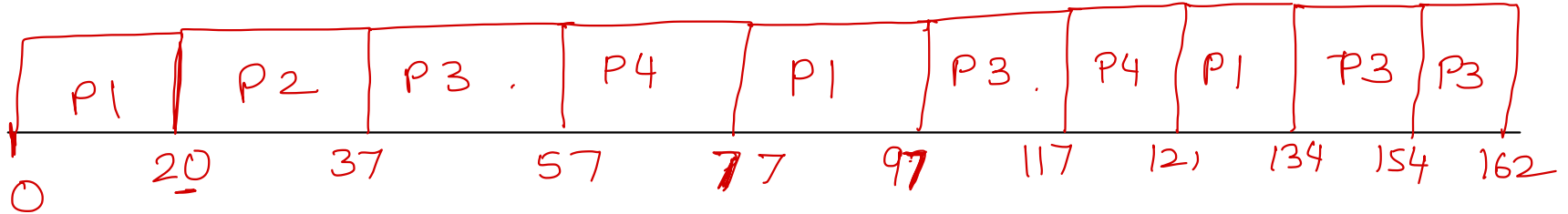
Process	Burst Time	Priority
P_1	10	3
P_2	1	1(high priority)
P_3	2	4(low priority)
P_5	5	2

P2→P5 →P1→ p3
Waiting time (p1 = 6, p2 =0 , p3=16 , p5= 1)

Average waiting time =(6 +0+ 16 + 1) /3= 5.75

Round Robin. → time slot (Quantum) = 20.

Arrival	Process	Burst Time	Remain time	waiting time
0	P_1	53	53-20=33-20=13	(77-20)=57 ; (121-97)=24 ; 20
0	P_2	17	0	20
0	P_3	68	68-20=48-20=28-20=8	37+40+17=94
0	P_4	24	24-20=4	97



P1
P2
P3
P4

pre-emptive.

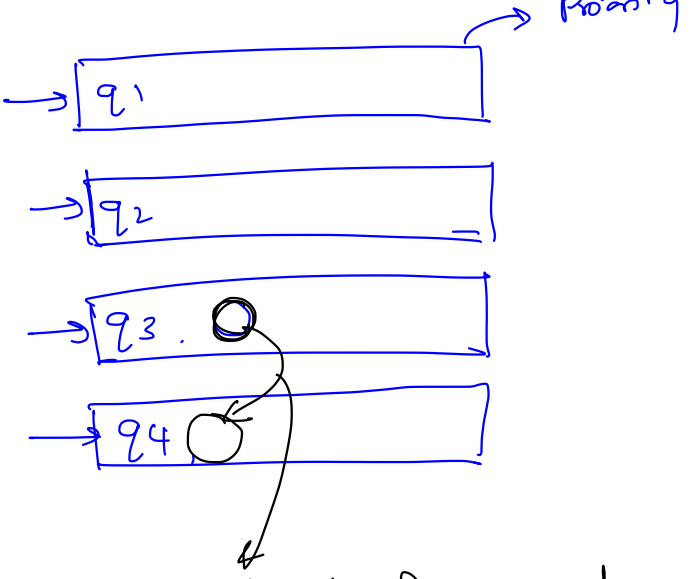
important services → Priority.

background task → SJF.

gui task → RR.

other task → FCFS.

Multilevel queue.



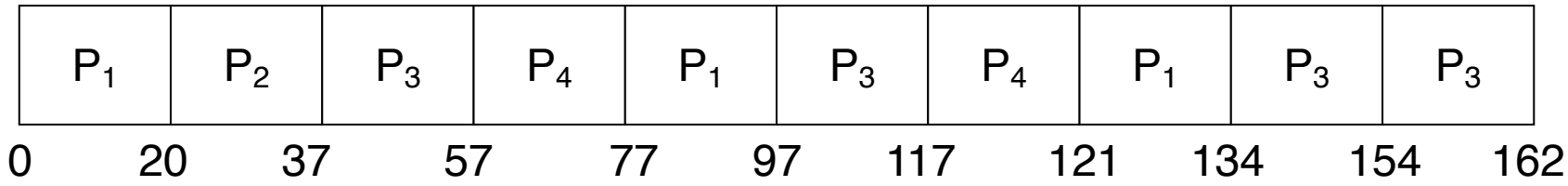
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.
- Round Robin algorithm is a preemptive .

Example of RR with Time Quantum = 20

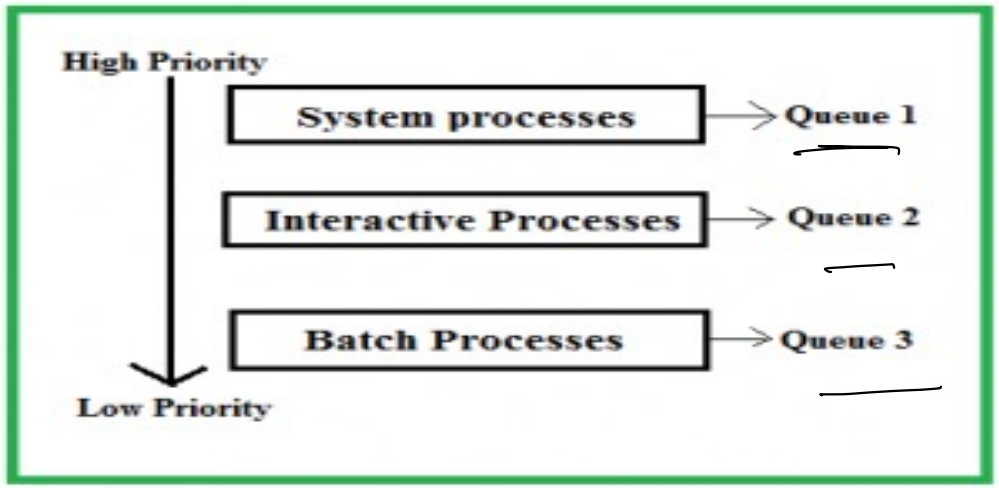
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



Typically, higher average turnaround than SJF, but better *response*.

Multilevel Queue Scheduling Algorithm



- 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.
- processes are permanently stored in one queue in the system and **do not move between the queue.**
- separate queue for foreground or background processes
- **For example:** A common division is made between foreground(or interactive) processes and background (or batch) processes.
- These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes

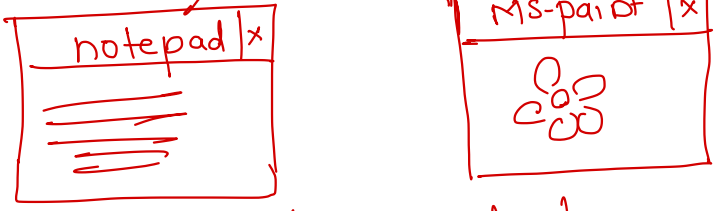
CPU Scheduling Algorithms

- **FCFS**
  - Process added first in ready queue should be scheduled first.
  - Non-preemptive scheduling
  - Scheduler in invoked when process is terminated, blocked or gives up CPU is ready for execution. Convoy Effect: Larger processes slow down execution of other processes.
- **SJF**
  - Process with lowest burst time is scheduled first.
  - Non-preemptive scheduling
  - Minimum waiting time
- **SRTF -Shortest Remaining Time First**
  - Similar to SJF - but Pre-emptive scheduling
  - Minimum waiting time
- **Priority**
  - Each process is associated with some priority level. Usually lower the number, higher is the priority.
  - Pre-emptive scheduling or Non-Preemptive scheduling

CPU Scheduling Algorithms

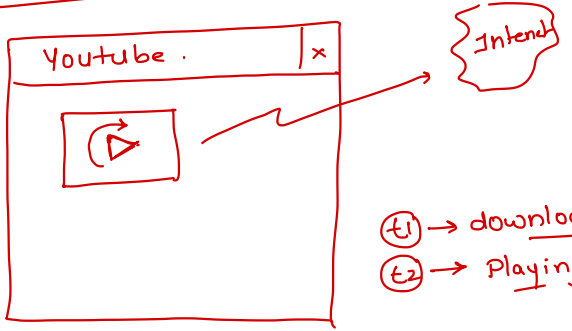
- Starvation
  - Problem may arise in priority scheduling.
  - Process not getting CPU time due to other high priority processes.
  - Process is in ready state (ready queue).
  - May be handled with aging -- dynamically increasing priority of the process.
- **Round-Robin**
  - Pre-emptive scheduling
  - Process is assigned a time quantum/slice. Once time slice is completed/expired, then process is forcibly
  - Pre-empted and other process is scheduled.
  - Min response time.
- **Multi-level queue**
  - In modern OS, the ready queue can be divided into multiple sub-queues and processes are arranged in them depending on their scheduling requirements. This structure is called as "Multi-level queue".
  - If a process is starving in some sub-queue due to scheduling algorithm, it may be shifted into another sub-queue. This modification is referred as "Multi-level feedback queue".
  - The division of processes into sub-queues may differ from OS to OS.

Multitasking



Independent

Multitasking



t1 → download → data → IPC

t2 → Playing

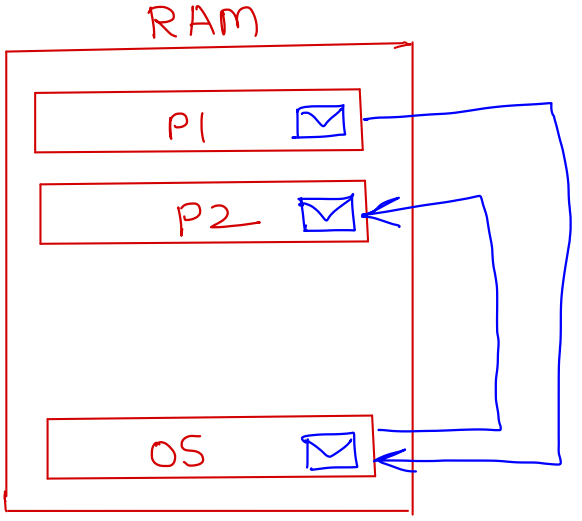
Inter Process Communication

Co-operating process

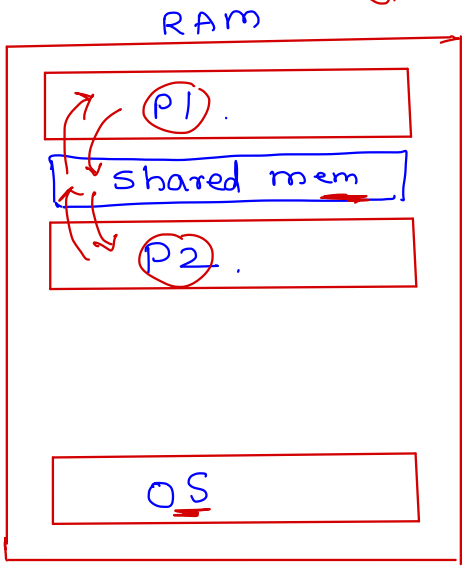
PCB → IPC

IPC Model.

(1) Message Passing



(2) Shared Memory.



LINUX / Unix IPC mechanism.

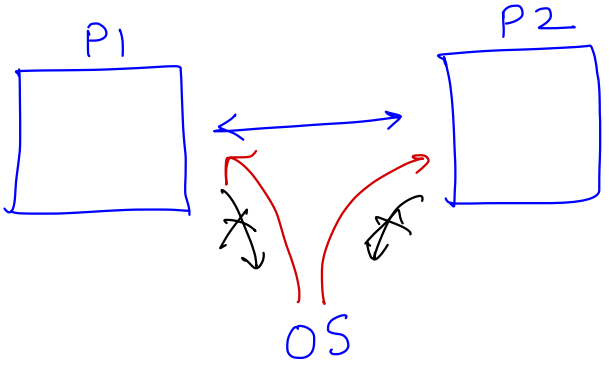
1. Signal.
2. Message Queue.
3. Pipe.
4. Socket.

→ Message passing

5. Shared Memory. → Shared memory

① Signal

→ Predefine Signal.



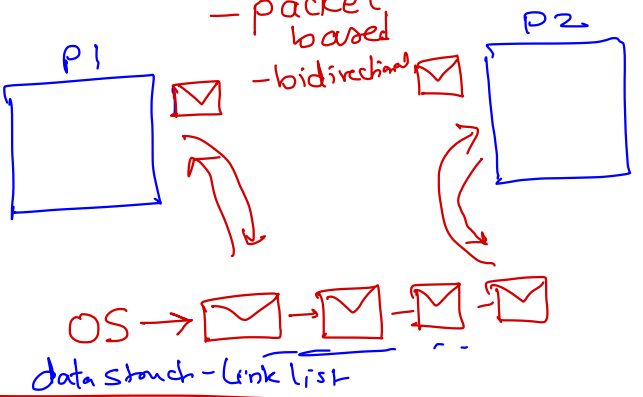
① SIGINT → Cntl+C .

② SIGKILL → .

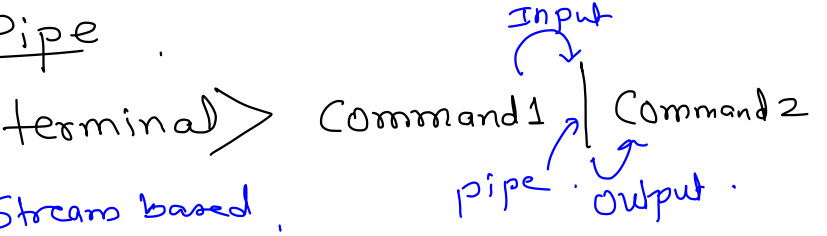
③ SIGSEGV

② Message Passing

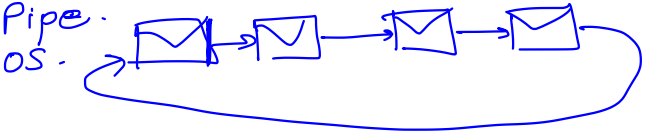
- packet based
- bidirectional

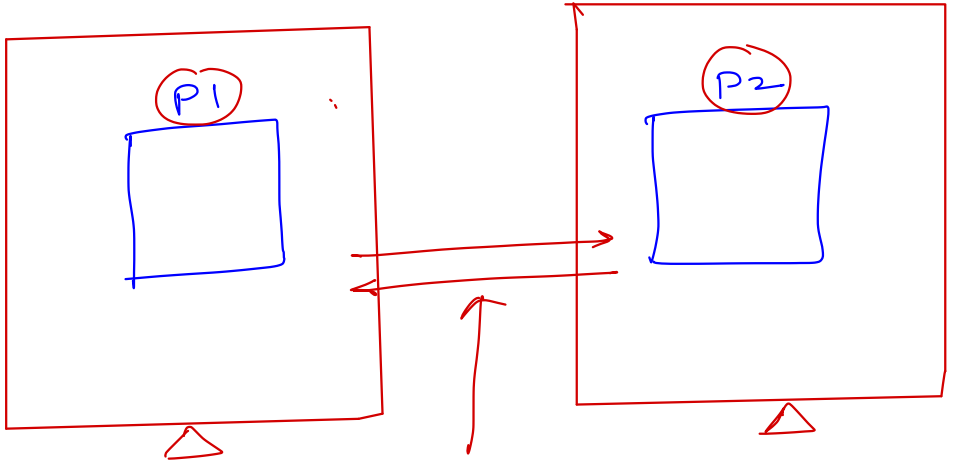


③ Pipe



- Stream based .
- Unidirection .

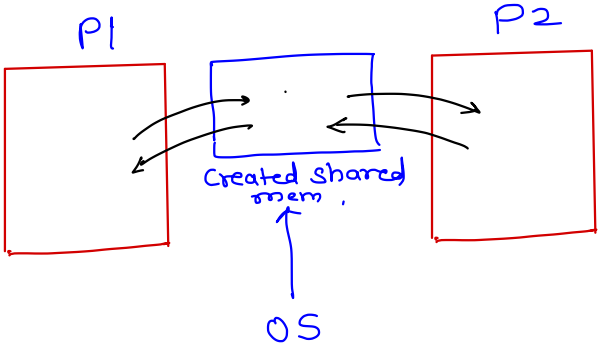




Socket. → same nfw.

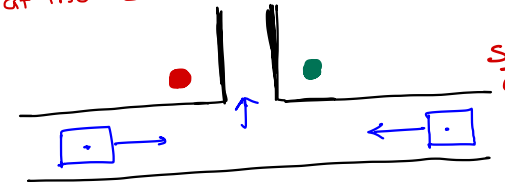
(LAN, WAN)

⑤ Shared Memory

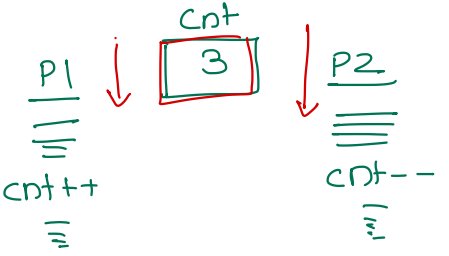


Shared memory is fastest.

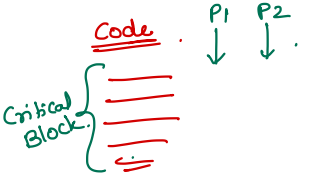
If two process try to access same resource at the same time, it is referred as "race condition"



Sync mechanism.
ensure that only one process will access resource at a time and thus data inconsistency is avoided.



Expected: Cnt will be unchanged (3)
But, if race condition occurs, then cnt may be 2 or 4. This is data inconsistency.



A block of code, executed by multiple processes at same time cause data inconsistancy

Inter process Communication (IPC)

- A process cannot access of memory of another process directly. OS provides IPC mechanisms so that processes can communicate with each other.

Independent process

- **do not get affected by the execution of another process**

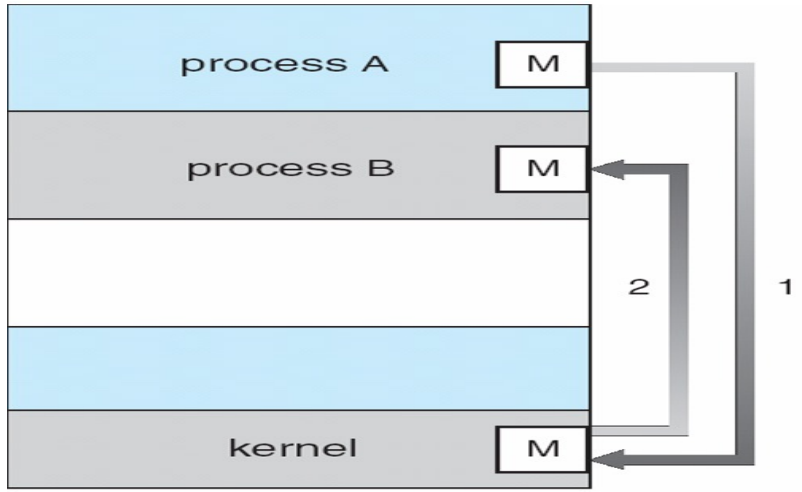
Cooperating process

- **get affected by the execution of another process**

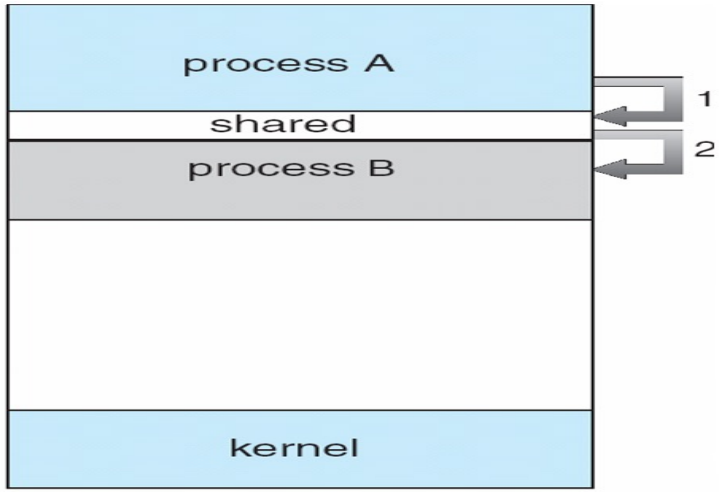
Reasons for cooperating processes:

- Information sharing
- Computation speedup
- Modularity
- Convenience
- Cooperating processes need **inter process communication** (**IPC**)

Mechanisms of IPC



(a)Message Passing



(b) Shared Memory Model

Message Passing

- communication takes place by means of messages exchanged between the cooperating processes
- Uses two primitives : Send and Receive

Shared Memory

- In the shared-memory model, a region of memory that is shared by cooperating processes is established.
- Processes can then exchange information by reading and writing data to the shared region.

Mechanisms of IPC

Unix/Linux IPC mechanisms

- Signals
- Shared memory
- Message queue
- Pipe
- Socket

Mechanisms of IPC

Signals

- OS have a set of predefined signals, which can be displayed using command
  - terminal> kill -l
- A process can send signal to another process or OS can send signal to any process.

Important Signals

- SIGINT (2): When CTRL+C is pressed, INT signal is sent to the foreground process.
- SIGKILL (9): During system shutdown, OS send this signal to all processes to forcefully kill them. Process cannot handle this signal.
- SIGSTOP (19): Pressing CTRL+S, generate this signal which suspend the foreground process. Process cannot handle this signal.
- SIGCONT (18): Pressing CTRL+Q, generate this signal which resume suspended the process.
- SIGSEGV (11): If process access invalid memory address (dangling pointer), OS send this signal to process causing process to get terminated. It prints error message "Segmentation Fault".

Mechanisms of IPC

Message Queue

- Used to transfer packets of data from one process to another.
- It is bi-directional IPC mechanism.
- Internally OS maintains list of messages called as "message queue" or "mailbox".

Pipe

- Pipe is used to communicate between two processes.
- It is stream based uni-directional communication.
- Pipe is internally implemented as a kernel buffer, in which data can be written/read.
- There are two types of pipe:
  - Unnamed Pipe
  - Named Pipe (FIFO)

Mechanisms of IPC

Socket

- Socket is defined as communication endpoint.
- Sockets can be used for bi-directional communication.
- Using socket one process can communicate with another process on same machine (UNIX socket) or different machine (INET sockets) in the same network.
- Sockets can also be used for communication over Bluetooth, CAN, etc.

Shared memory

- OS creates a memory region that is accessible to multiple processes.
- Multiple processes accessing a shared memory need to be synchronized to handle race condition problem.
- Fastest IPC mechanism.
- Both processes have direct access to shared memory(no os invoked)

Synchronization

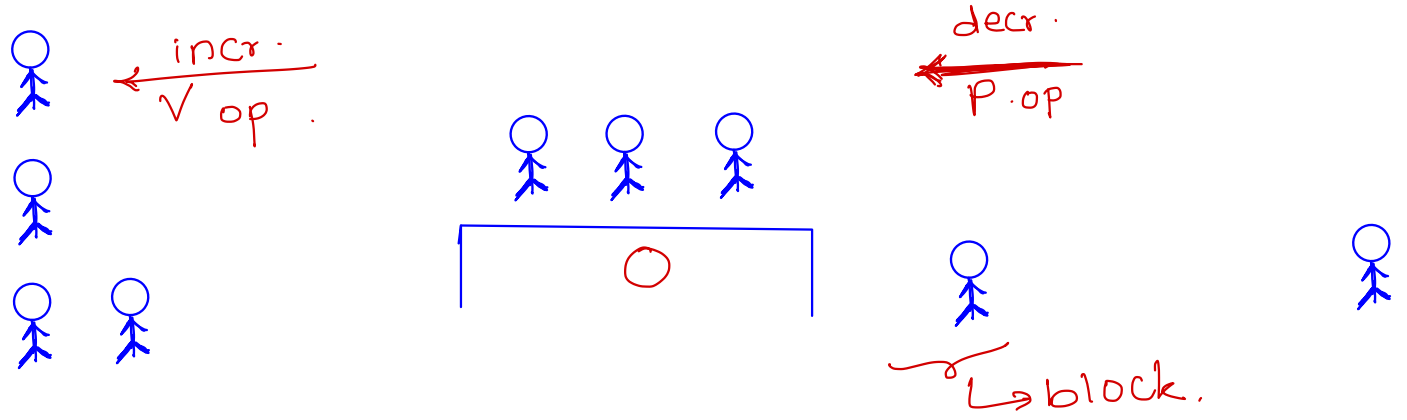
- If two/multiple processes try to access same resource at same time , it is referred as “race condition”
- When race condition occurs, resource may get corrupted (unexpected results).
- Peterson's problem: If two processes are trying to modify same variable at the same time, it can produce unexpected results.
- Synchronization Mechanism ensure that only one process will access resource at a time and thus data inconsistency is avoided.
- A block of code ,executed by multiple processes at same time cause data inconsistency. Such kind of code block is called Critical section.
- To resolve race condition problem, one process can access resource at a time. This can be done using sync objects/primitives given by OS.
- OS provide some Synchronization objects

1) Semaphore
2) Mutex

Semaphore

- Semaphore was suggested by Dijkstra scientist (dutch math)
- Semaphore is a counter
- On semaphore two operations are supported:
- wait operation: decrement op: P operation:
  - semaphore count is decremented by 1.
  - if cnt < 0, then calling process is blocked(block the current process).
  - typically wait operation is performed before accessing the resource.
- signal operation: increment op: V operation:
  - semaphore count is incremented by 1.
  - if one or more processes are blocked on the semaphore, then wake up one of the process.
  - typically signal operation is performed after releasing the resource.
- Q. If sema count = -n, how many processes are waiting on that semaphore?
  - Answer: "n" processes waiting

Semaphore :



Semaphore
- Counting/Classic.
  - multipeople can use same resource
- Binary
  - one use resource.

Semaphore

- Counting Semaphore/classic
  - When multiple processes can access a resource.
  - Allow "n" number of processes to access resource at a time.
  - Or allow "n" resources to be allocated to the process.
- Binary Semaphore
  - When a single process can access a resource at a time.
  - Allows only 1 process to access resource at a time or used as a flag/condition