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

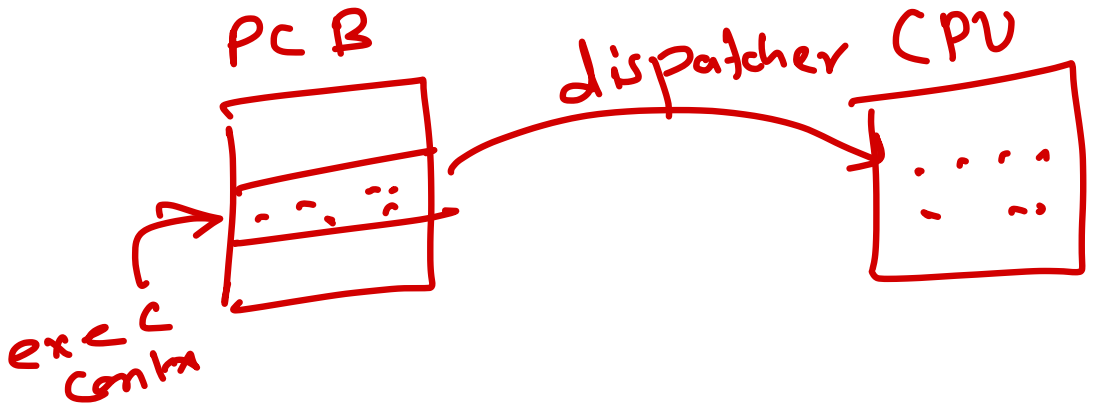
Q. Process which is in the main memory waiting for the CPU time considered in a _____.

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

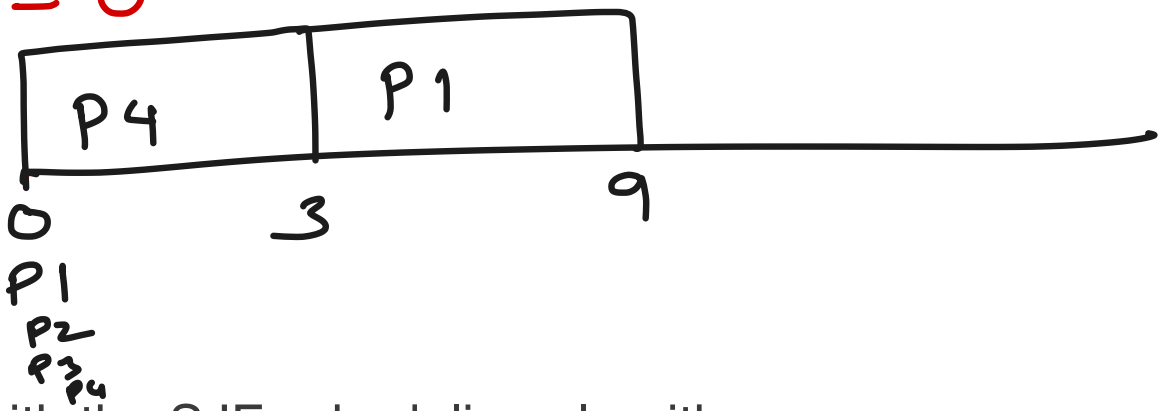


Q. Which of the following CPU scheduling algorithm leads to starvation?

A. FCFS
B. Priority
C. Round Robin
D. None of the above
E. All of the above

Q. Consider the following set of processes, the length of the CPU burst time given in milliseconds.

Process	Burst time	arrival time = 0
P1	6	0
P2.	8	0
P3	7	0
P4	3	0



Assuming the above process being scheduled with the SJF scheduling algorithm.

a) The waiting time for process P1 is 3ms
b) The waiting time for process P1 is 0ms
c) The waiting time for process P1 is 16ms
d) The waiting time for process P1 is 9ms

Quiz

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

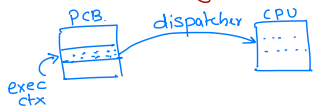
c) Running state

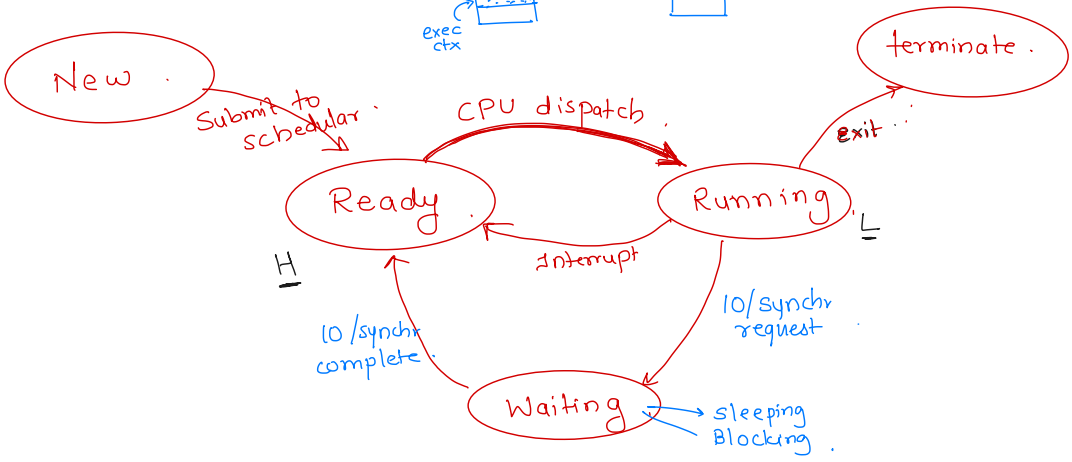
d) Ready state

Q. FCFS scheduling is
(a) Fair-Share scheduling
(b) Deadline scheduling
(c) Non-preemptive scheduling
(d) Pre-emptive scheduling

Process State diagram

PCB → Execution Context (values of CPU).





CPU Scheduler → decide which process to run next.

CPU Dispatch → dispatch the selected process on CPU.

CPU sched invoke.

1. Running → terminated
2. Running → waiting
3. Running → Ready
4. Waiting → Ready

Sched Criteria.

1) CPU Utilization - max
2) Throughput - **max.**
3) Waiting time - **min.**
4) Turn around time - **min**
5) Response time. - **min**

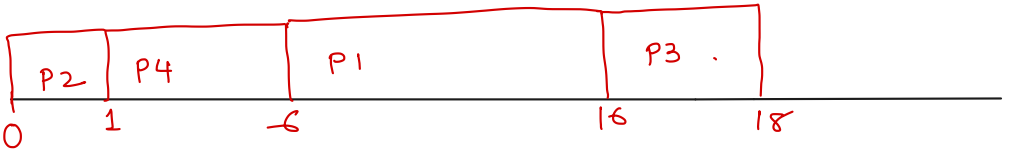
CPU Scheduling Algorithms

- **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 Preemptive scheduling
  - Minimum waiting time
- **Priority**
  - Each process is associated with some priority level. Usually lower the number, higher is the priority.
  - Preemptive scheduling or NonPreemptive scheduling

Priority

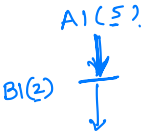
non-pre-emptive

Process	Arrival	CPU Burst	Priority	Waiting time
P1	0	10	3	6
P2	0	1	1 (highest prio)	0
P3	0	2	4 (lowest prio)	16
P4	0	5	2	1



P1
P2
P3
P4

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



Process arrival order (non pre-emptive)

P1(8) → P1
P2(10) → P3
P3(3) → P4
P4(5) → P5
P5(7) → P6
P6(9) → P7
P7(2) → P2

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

Starvation — A process is not getting enough CPU time for it execution
Reason → lowest Priority

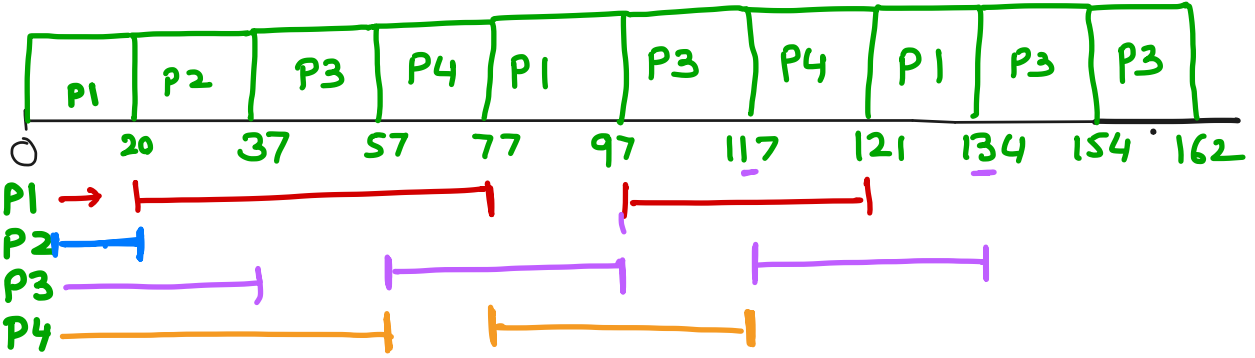
Round Robin :

pre-emptive

Process	CPU Burst	Remaining time	Waiting time	Response time
P1	53	~~33~~ ~~13~~ 0	57+24 = 81	0
P2	17	0	20	20
P3	68	~~48~~ ~~28~~ 8	37+40+17 = 94	37
P4	24	~~4~~ 0	57+40 = 97	57

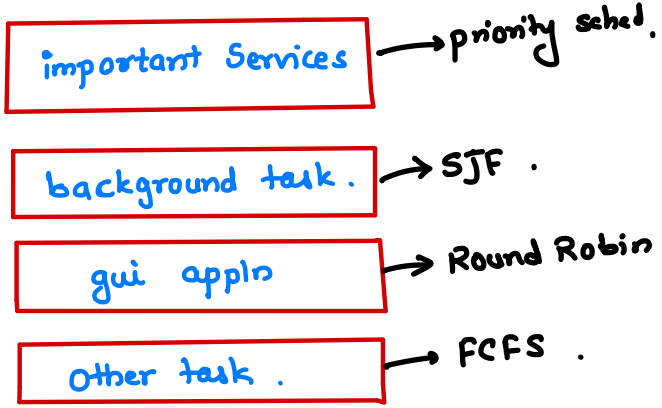
Time Quantum = 20.

$$\text{avg waiting time} = \frac{81+20+94+97}{4} = 73.$$



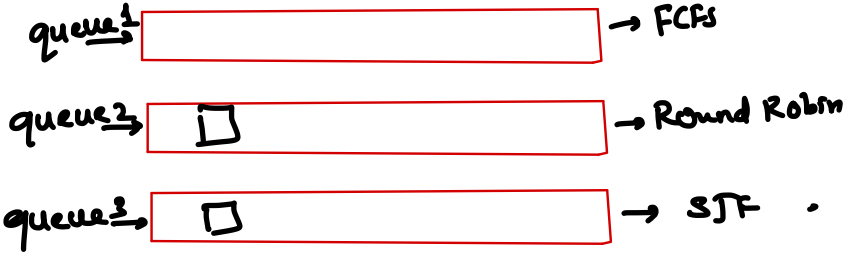
CPU Scheduling Algorithms

- **Round-Robin**
  - Preemptive scheduling
  - Process is assigned a time quantum/slice. Once time slice is completed/expired, then process is forcibly preempted and other process is scheduled.
  - Min response time.
  - If time quantum is too high, this algorithm works like FCFS.
  - If time quantum is too low, CPU scheduler will be invoked frequently to schedule next process. More system time is wasted in switching from one process to another. This reduces efficiency.
- 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.



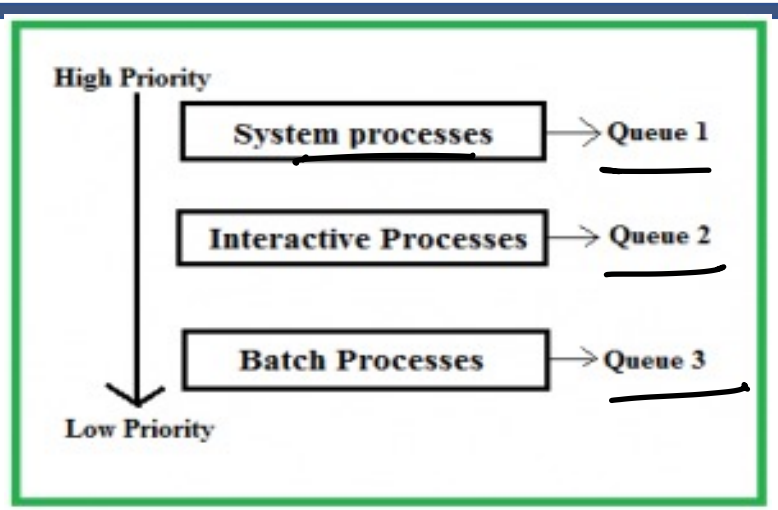
Multilevel Queue.

Ready queue is divided into multiple Sub-queues. Different Sub-queue can have different algorithm.



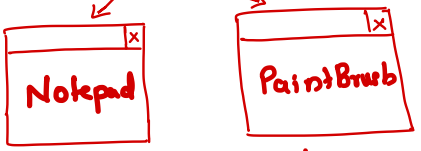
In multi-level feedback queue, if a process not getting enough cpu time in a queue, OS can transfer it to another queue.

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

Multi-tasking.

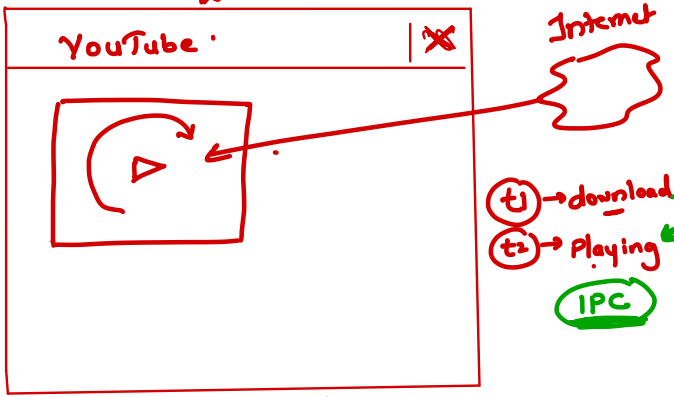


Independent

Multi-tasking.

↓

Web Browser

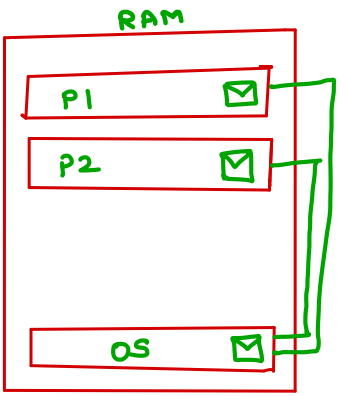


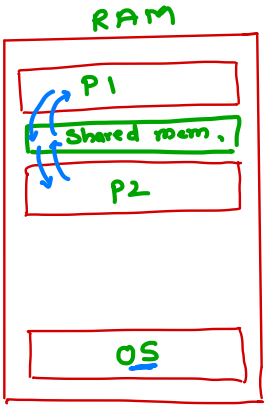
Co-operating process.

IPC 2 Type:

① Message Passing

(2) Shared Memory





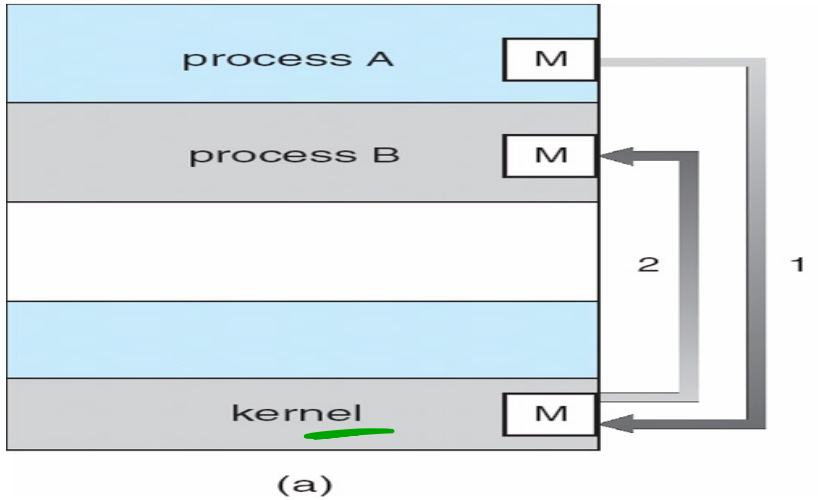
✓

UNIX/LINUX IPC mech.

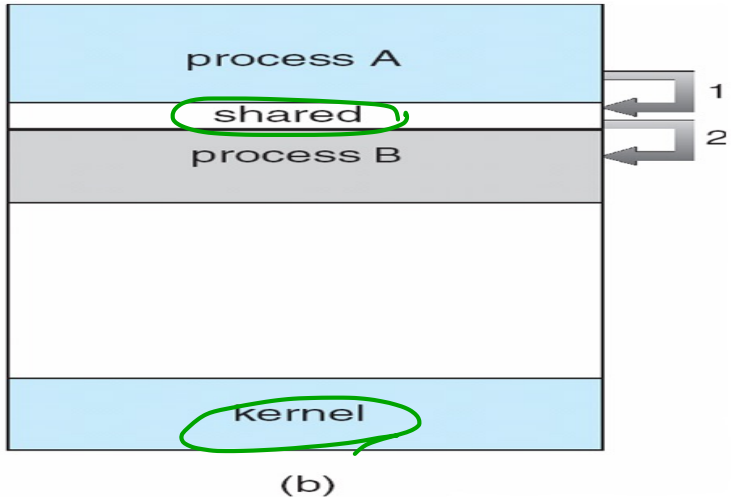
① Signal
② Message
③ Pipe
④ Socket
⑤ Shared Memory → Shared memory

} Message passing

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.

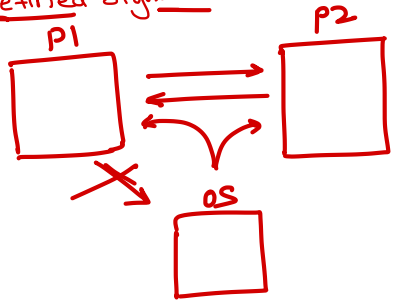
Mechanisms of IPC

Unix/Linux IPC mechanisms

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

① Signals

- predefined signals.



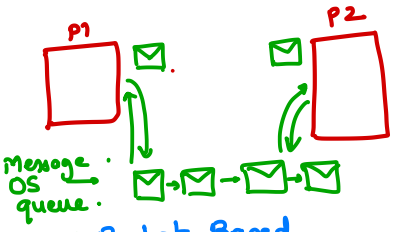
64 signals..

② SIGINT : cntrl + C → process terminated.

⑨ SIGKILL : shutdown.

⑪ SIGSEGV : dangling pointer
Segment violation.
ie seg fault → abnormal terminate.

② Message queue :

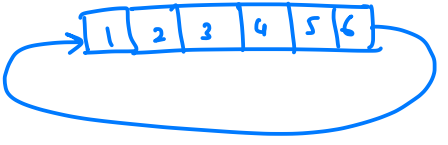


→ Packet Based.
→ Bi-directional

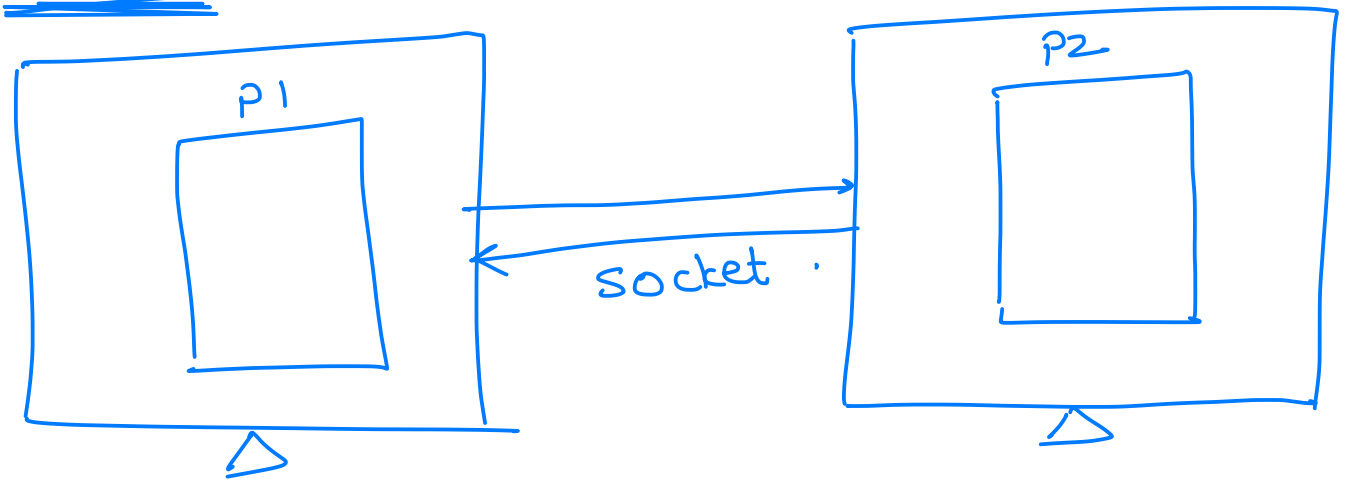
③ Pipe

→ Stream based.
→ Uni-directional.

Circular queue.



④ Socket



→ Socket is comm^n endpoint

→ Socket are used for comm^n b/w two processes on same computer or diff comp.

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

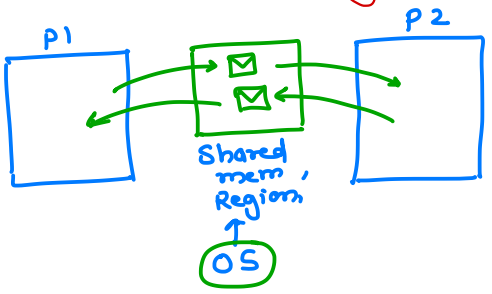
Socket

- 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

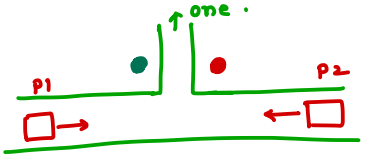
- OS creates a memory region that is accessible to multiple processes.
- Multiple processes accessing a shared memory need to be synchronized to handle race condition problem.
- Fastest IPC mechanism.
- Both processes have direct access to shared memory(no os invoked)

⑤ Shared Memory

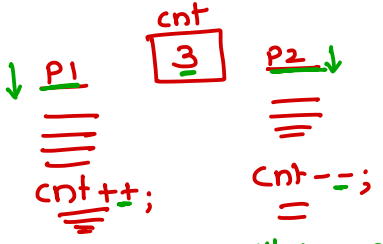


→ Shared mem is fastest IPC mech.

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

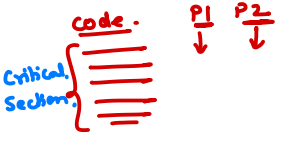


Syn mechanism → ensure that only one process will access resource at a time and thus data in-consistency is avoided.



expected : count will be unchanged (3)

But, if race condition occurs, then count may be 2 or 4. This is data inconsistency



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

OS provide some sync objects.

1. Semaphore. ✓
2. Mutex. ✓

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

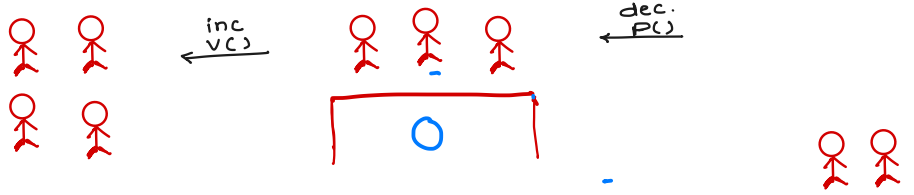
Semaphore

- 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 :

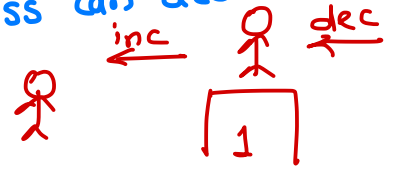
Semaphore is a counter

cnt < 0
↳ process - block.



If semaphore count is -n, number of waiting processes are n.

Semaphore
- Counting/Classic — when multiple processes can access a resource at a time.
- Binary (0 or 1).
  ↳ when a single process can access a resource (at a time).



Mutex (Mutual Exclusion) : when only one process access resource like binary semaphore.
Mutex is not counter:D has two states is unlocked & locked state.

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