Operating System Concepts

SunBeam Institute of Information & Technology, Hinjwadi, Pune & Karad.

Trainer: Akshita Chanchlani
Email: akshita.chanchlani@sunbeaminfo.com

Process Management

-When we say an OS does process management it means an OS is responsible for process creation, to provide environment for an execution of a process, resource allocation, scheduling, resources management, inter process comminication, process coordination, and terminate the process.

Q. What is a Program?

-**User view:** Program is a set of instructions given to the machine to do specific task.

-**System view:** Program is an executable file divided into sections like exe header, bss section, data section, rodata section, code section, symbol table.

Process and Program

- A **process** is an instance of a program in execution.
- Running program is also knows as **Process.**
- When a program gets loaded in to memory is also known as **Process.**
- A **Program** is a set of instructions given to the machine to do specific task.
  - Three types of Programs:
    - User Programs (c/java program)
    - Application Programs (ms office)
    - System Programs (device drivers, interrupt handlers etc)

Process Control Block/ Process Descriptors

- When an execution of any program is started one structure gets created for that program/process to store info about it, for controlling its execution, such a structure is known as PCB: Process Control Block.
- Per process one PCB gets created and PCB remains inside the main memory throughout an execution of a program, upon exit PCB gets destroyed from the main memory.
- It has information about process like:
  - process id – pid
  - process state - current state of the process
  - memory management info
  - CPU scheduling info
  - Program Counter -- address of next instruction to be executed
  - Exit status
  - Execution Context
  - I/O devices info etc...

Five State Process Diagram

 # Process States:

-Throughout execution, process goes through different states out of which at a time it can be only in a one state.

-States of the process:

1. New state: upon submission or when a PCB for a process gets created into the main memory process is in a new state.

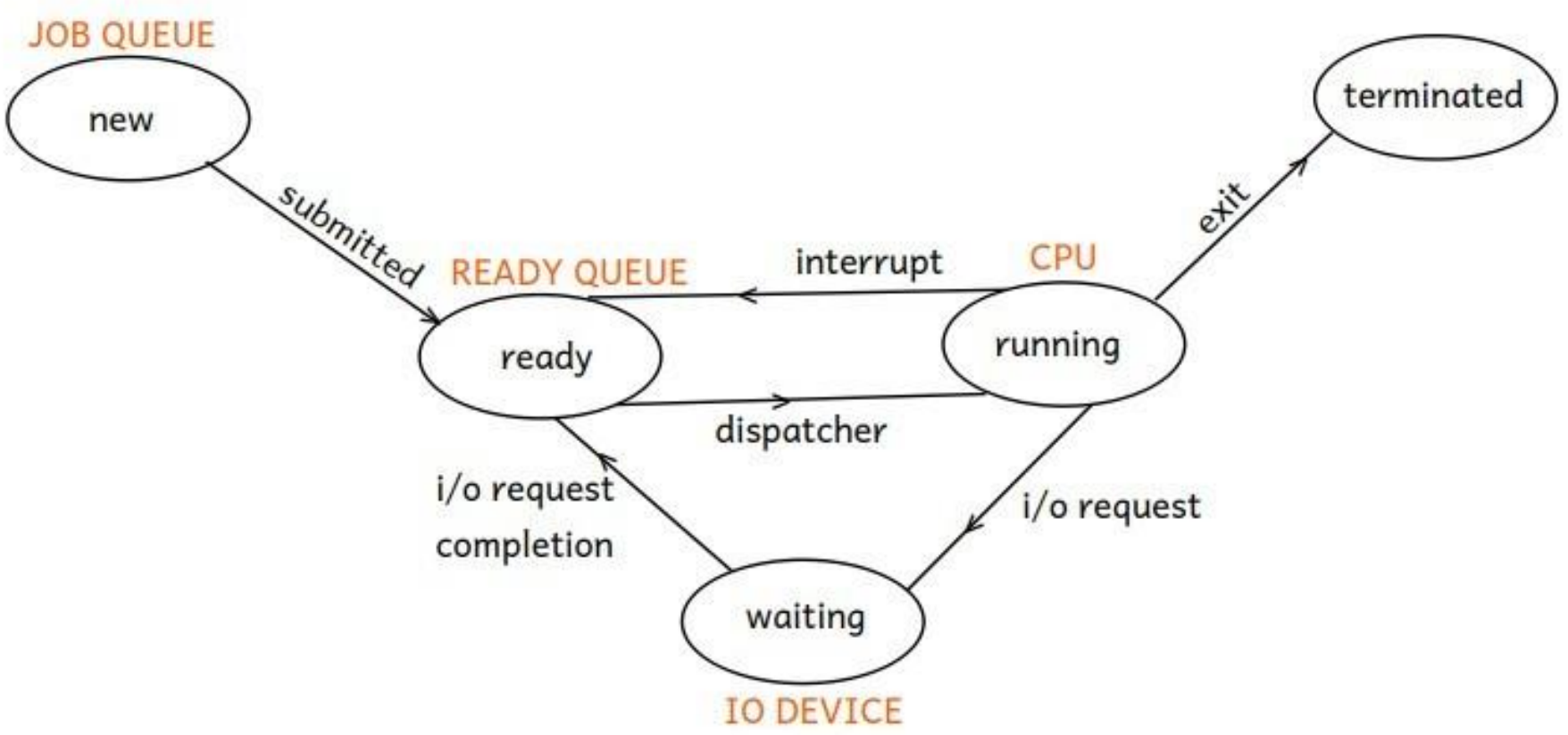
2. Ready state: after submission, if process is in the main memory and waiting for the CPU time, it is in a ready state.

3. Running state: if currently the CPU is executing any process then state of that process is considered as a running state.

4. Waiting state: if a process is requesting for any i/o device then state of that process is considered as a waiting state.

5. Terminated state: upon exit, process goes into terminated state and its PCB gets destroyed from the main memory.

Five State Process Diagram



PROCESS STATE DIAGRAM

Data Structures Maintained by Kernel at the time of process Execution

 # Process States:

-To keep track on all running programs, an OS maintains few **data structures** reffered as **kernel data structures:**

1. Job queue: it contains list of PCB's of all submitted processes.

2. Ready queue: it contains list of PCB's of processes which are in the main memory and waiting for the CPU time.

3. Waiting queue: it contains list of PCB's of processes which are requesting for that particular device.

Process May be in one of the state at a time

New

- when program execution is started or upon process submission process
- when a PCB of any process is in a job queue then state of the process is referred as a new state.

Ready

- When a program is in a main memory and waiting for the cpu
- when a PCB of any process is in a ready queue then state of the process is referred as a ready state.

Running

- When a CPU is executing a process

Exit

- when a process is terminated

Waiting

- when a process is requesting for any i/o device then process change its change from running to waiting state
- When a PCB of any process is in a waiting queue of any device

Schedulers

- **Job Scheduler/long term schedulers :**
  It is a system program which selects/schedules jobs/processes from job queue to load them onto the ready queue.
- **CPU Scheduler/Short term schedulers**
  - It is a system program that selects / schedules process from ready queue to load into CPU
  - selects which process should be executed next and allocates CPU
- **Dispatcher**
  - It is a system program that loads a process onto the CPU that is scheduled by the CPU scheduler.
  - Time required for the dispatcher to stops execution and one process and starts execution of another process is called as **"dispatcher latency".**

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.
- **Context** of a process represented in the PCB.
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch.

Context Switch = state-save + state-restore

"state-save" -- to save execution context of suspended process into its PCB
"state-restore" -- to restore execution context of the process which is scheduled by the cpu scheduler onto the cpu registers.

When a high priority process arrived into the ready queue, low priority process gets suspended by means of sending an interrupt, and control of the CPU gets allocated to the high priority process, and its execution gets completed first, then low priority process can be resumed back, i.e. the CPU starts executing suspended process from the point at which it was suspended and onwards.

Process Scheduling

- Process scheduling decides which process to dispatch (to the Run state) next.
- In a multiprogrammed system several processes compete for a single processor
- **Preemptive**

  a process **can be removed** from the Run state before it completes or blocks (timer expires or higher priority process enters Ready state).
- **Non preemptive**

  a process **can not be removed** from the Run state before it completes or blocks.

CPU Scheduling algorithms

1. FCFS : First Come First Served
2. SJF: Shortest Job First
3. Priority Scheduling
4. Round Robin
5. Multi-level Queue
6. Multi-level Feedback Queue

CPU Scheduling Algorithm Optimization Criteria's

CPU Utilization

- utilization of the CPU must be as max as possible.

Throughput

- It is the total work done per unit time.
- Throughput must be as max as possible.

Waiting time

- It is the total amount of time spent by the process in a ready queue for waiting to get control of the CPU from its time of submission.
- waiting time must be as min as possible.

CPU Scheduling Algorithm Optimization Criteria's

Turn Around Time

- It is the total amount of time required for the process to complete its execution from its time of submission., turn around time must be as min as possible.
- **Turn around time = waiting time + execution.**
- turn around time is the sum of periods spent by the process in a ready queue and onto the CPU to completes its execution.

Response time

- It is a time required for the process to get first response from the CPU from its time of submission.
- One need to select such an algorithm in which response time must be as min as possible.

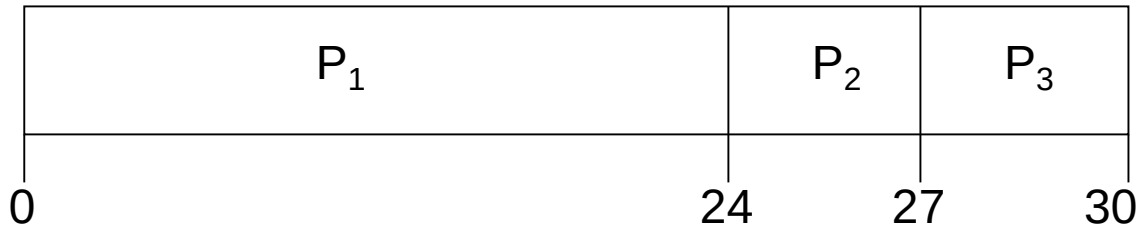
Execution Time and Burst Time

- Execution time/Running time
  - It is the total amount of time spent by the process onto the CPU to complete its execution.
- CPU burst time
  - Total no. of CPU cycles required for the process to complete its execution.

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 for the schedule is:



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

convoy effect
as all the other processes wait for the one big process to get off the CPU.

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

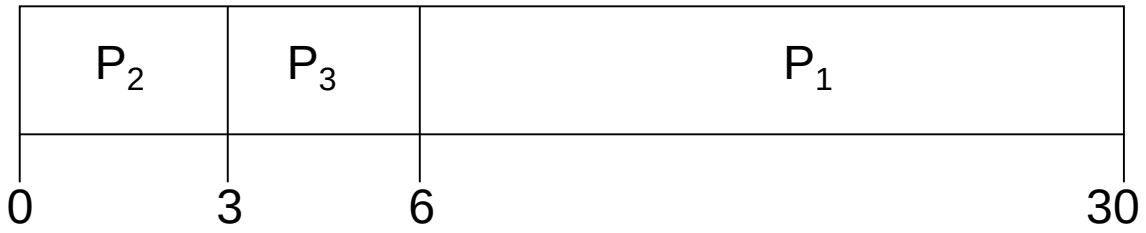
The Gantt chart for the schedule is:

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.

Convoy effect short process behind long process



Shortest-Job-First (SJF) Scheduling

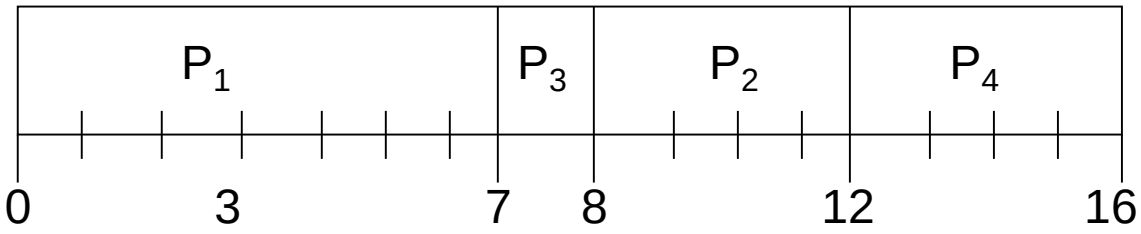
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **non preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as **Shortest-Remaining-Time-First (SRTF).**
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

Average waiting time = (0 + 6 + 3 + 7)/4 = 4

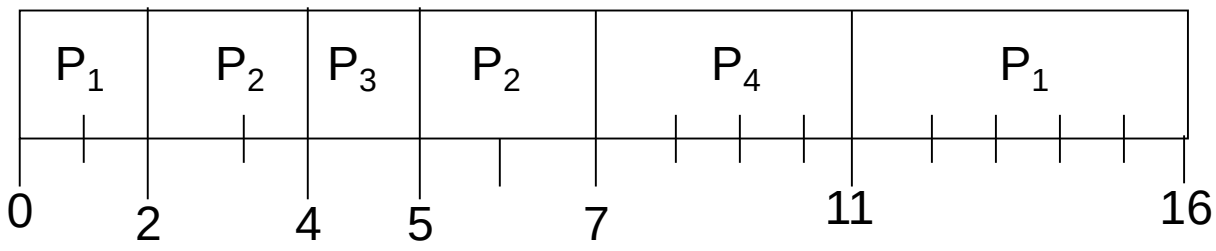
P1 waiting time = 0
P2 waiting time = 6 (8-2)
P3 waiting time = 3(7-4)
P4 waiting time = 7(12-5)



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

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

Average waiting time = (11 + 1 + 0 +2)/4 = 3



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

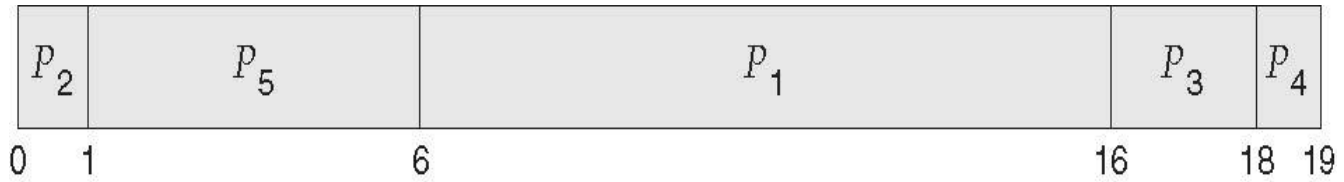
Problem ≡ **Starvation** – low priority processes may never execute

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



Average waiting time = 8.2 msec

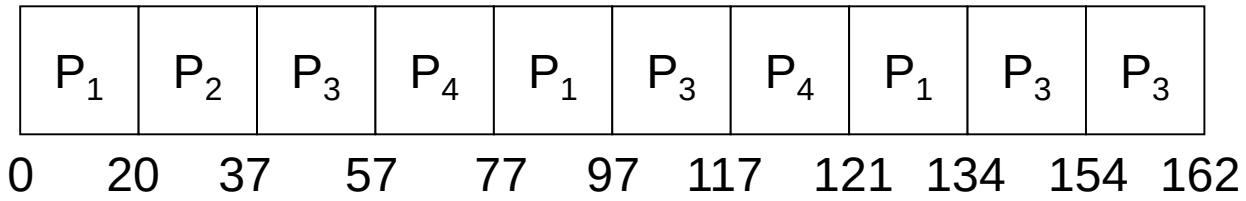
Round Robin (RR)

- 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.

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:

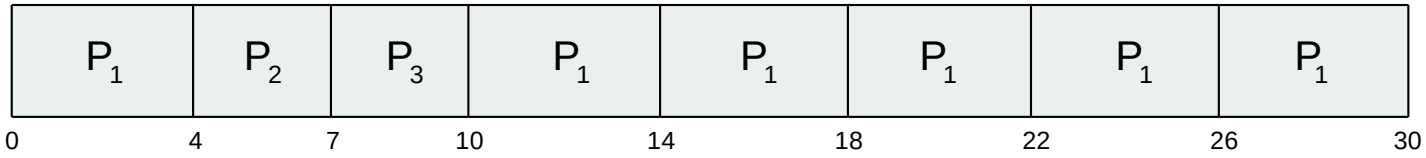


Typically, higher average turnaround than SJF, but better *response*.

Round Robin Example : Time Quantum = 4

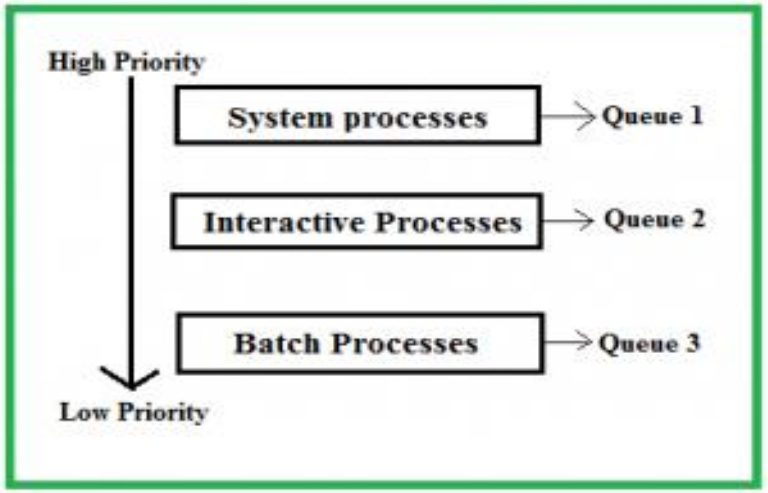
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



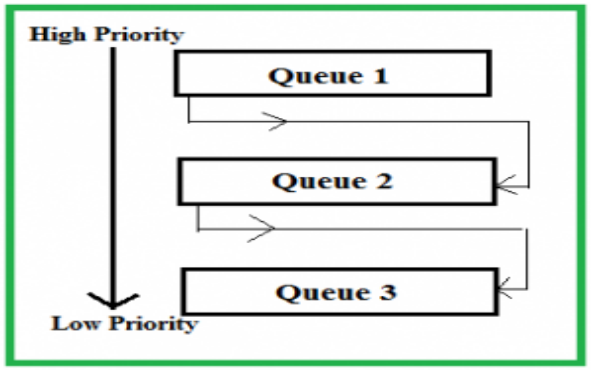
P1 waits for 6 millisconds (10- 4)
P2 waits for 4 millisconds
P3 waits for 7 millisconds.
Thus, the average waiting time is 17/3 = 5.66 mss.

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

Multilevel feedback queue scheduling



- keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.
- If a process uses too much CPU time then it will be moved to the lowest priority queue.
- This leaves I/O bound and interactive processes in the higher priority queue.
- Similarly, the process which waits too long in a lower priority queue may be moved to a higher priority queue.
- The **multilevel feedback queue scheduler** has the following parameters:
- The number of queues in the system.
- The scheduling algorithm for each queue in the system.
- The method used to determine when the process is upgraded to a higher-priority queue.
- The method used to determine when to demote a queue to a lower - priority queue.
- The method used to determine which process will enter in queue and when that process needs service.

Inter process Communication (IPC)

- Passing information between processes
- Used to coordinate process activity
- Processes within a system may be **independent** or **cooperating**

Independent process

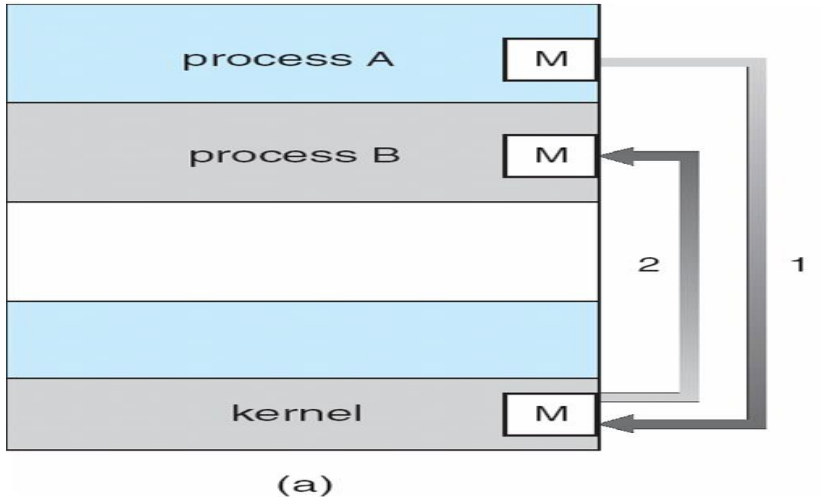
- do not get affected by the execution of another process

Cooperating process

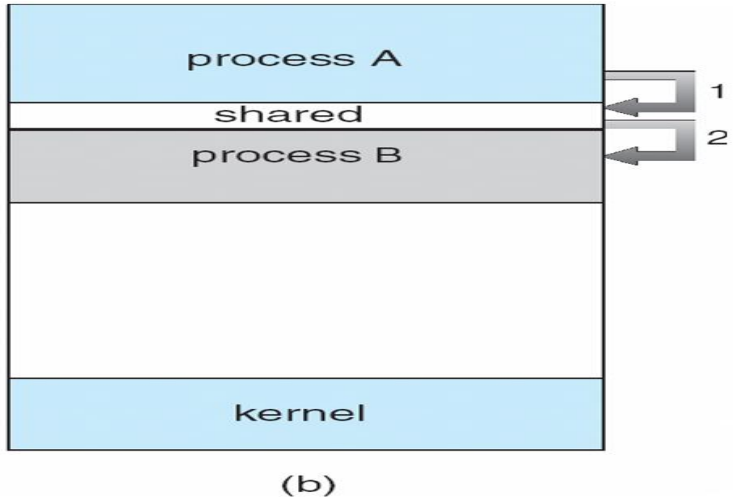
- get affected by the execution of another process

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

Two models/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.

Process Synchronization

- Concurrent access to shared data may result in **data inconsistency.**
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
- **Race condition**: The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.

- To prevent race conditions, concurrent processes must be **synchronized**.

The Critical Section / Region Problem

- Occurs in systems where multiple processes all compete for the use of shared data.
- Each process includes a section of code (the **critical section**) where it accesses this shared data.
- The problem is to ensure that **only one process at a time is allowed** to be operating in its critical section.

Critical-Section Problem

- Consider system of n processes $\{p_0, p_1, \ldots p_{n-1}\}$
- Each process has **critical section** segment of code
  - Process may be changing common variables, updating table, writing file, etc
  - When one process in critical section, no other may be in its critical section
- Each process must ask permission to enter critical section in **entry section**, may follow critical section with **exit section**, then **remainder section**
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

```
repeat
    entry to section
            critical section
    exit section
            remainder of program
until false;
```

Synchronization Tools:

1. Semaphore: there are two types of semaphore

i. Binary semaphore: can be used when at a time resource can be acquired by only one process.

- It is an integer variable having either value is 0 or 1.

ii. Counting / Classic semaphore: can be used when at a time resource can be acquired by more than one processe

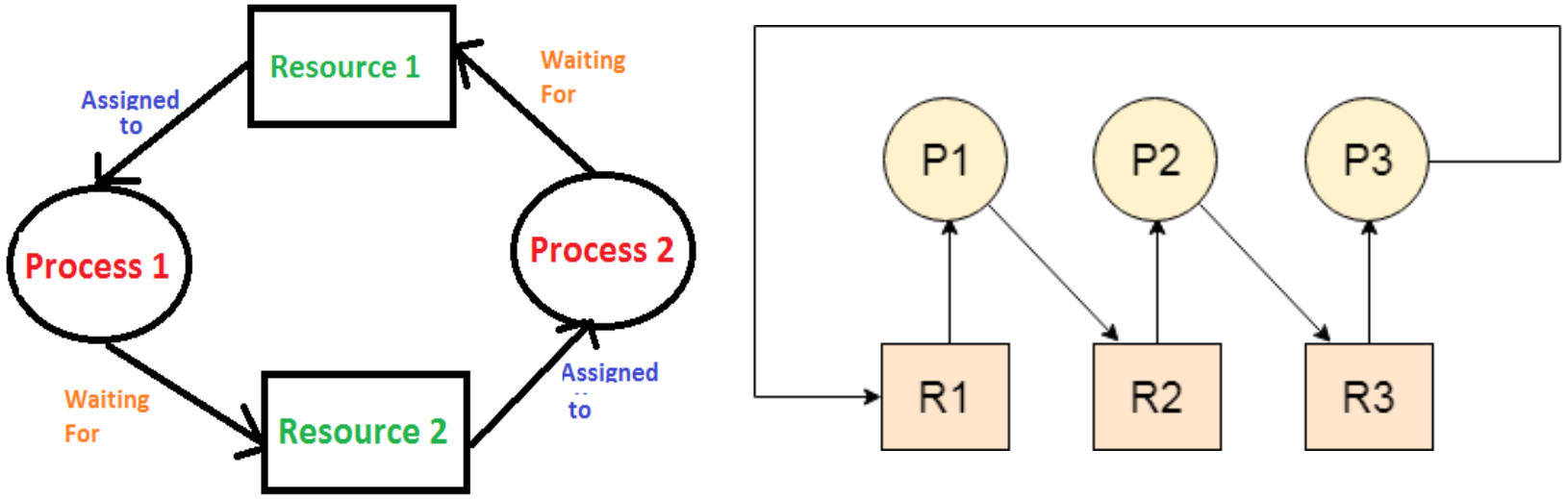
2. Mutex Object: can be used when at a time resource can be acquired by only one process.

- Mutex object has two states: **locked & unlocked**, and at a time it can be only in a one state either locked or unlocked.

- Semaphore uses **signaling mechanism**, whereas mutex object uses **locking and unlocking mechanism.**

Deadlock

- ***Deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- A set of processes is in a ***deadlock state*** when every process in the set is waiting for a resource that can only be released by another process in the set.



Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.

Conditions for Deadlock

Mutual exclusion

- At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource.
- If another process requests that resource, the requesting process must be delayed until the resource has been released.

Hold and wait

- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

No preemption.

- control of the resource cannot be taken away forcefully from any process.

Circular wait

- A set { P0 , Pl, ... , Pn } of waiting processes must exist such that
- Po is waiting for a resource held by P1,
- P1 is waiting for a resource held by P2, ... ,
- Pn-1 is waiting for a resource held by Pn,
- Pn is waiting for a resource held by Po.

Methods for handling deadlock

1. Deadlock Prevention
2. Deadlock Detection and Avoidance
3. Deadlock Recovery

Handling Deadlock

1. Deadlock Prevention: deadlock can be prevented by discarding any one condition out of four necessary and sufficient conditions.

2. Deadlock Detection & Avoidance: before allocating resources for processes all input can be given to deadlock detection algorithm in advanced and if there are chances to occur deadlock then it can be avoided by doing necessary changes.

- There are two deadlock detection & avoidance algorithms:

1. Resource Allocation Graph Algorithm

2. Banker's Algorithm

Deadlock Recovery

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

For Resource

Preempt the resource

- 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.

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.

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.