Operating System Concepts

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

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

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.

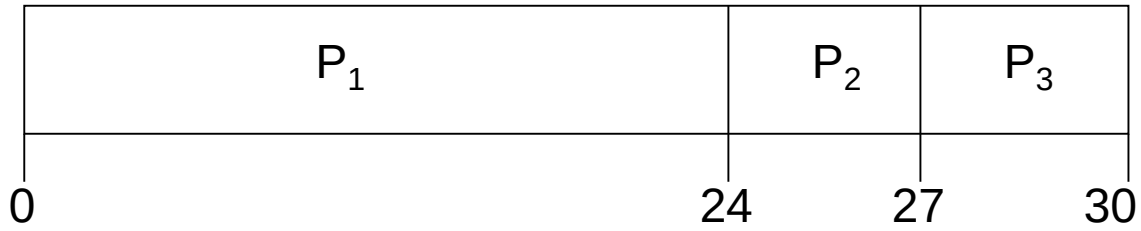
Execution Time and Burst Time

- 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 .

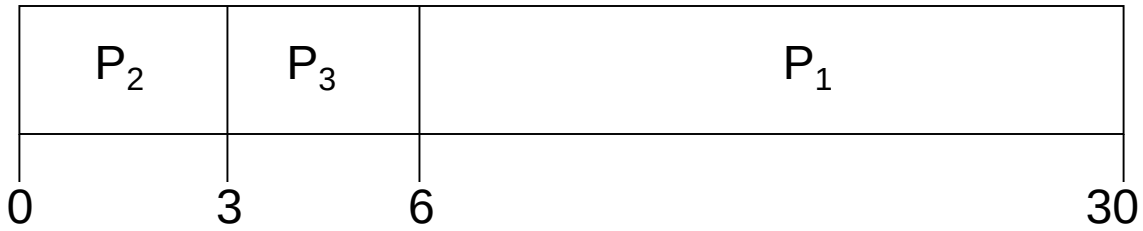
The Gantt chart for the schedule is:

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

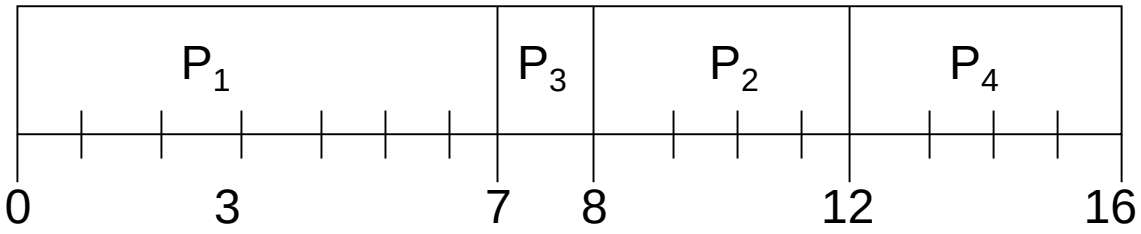
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - **non preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **preemptive** – if a new process arrives with 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

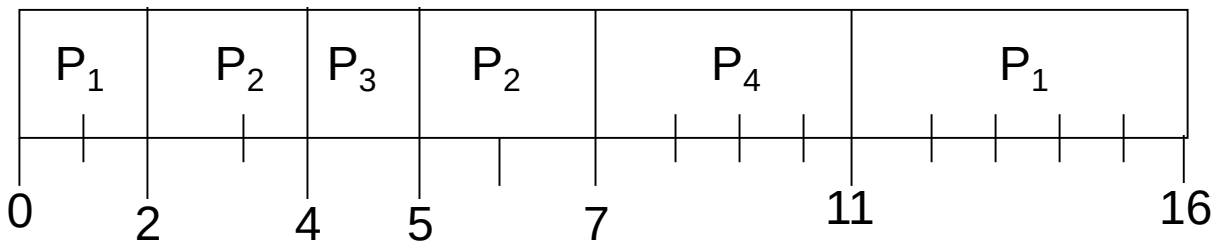
P1 waiting time = 0
P2 waiting time = 6 (8-2)
P3 waiting time = 3(7-4)
P4 waiting time = 7(12-5)



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

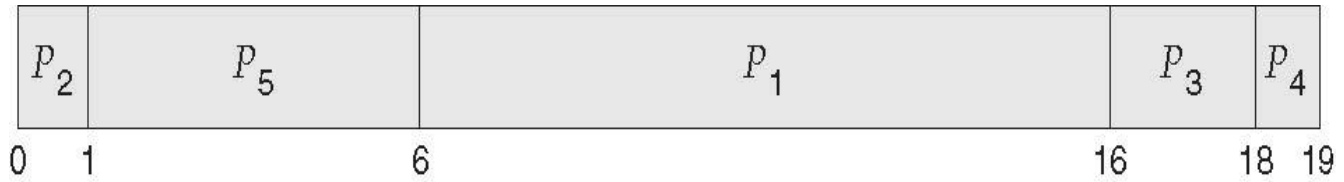
Problem ≡ **Starvation** – low priority processes may never execute

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

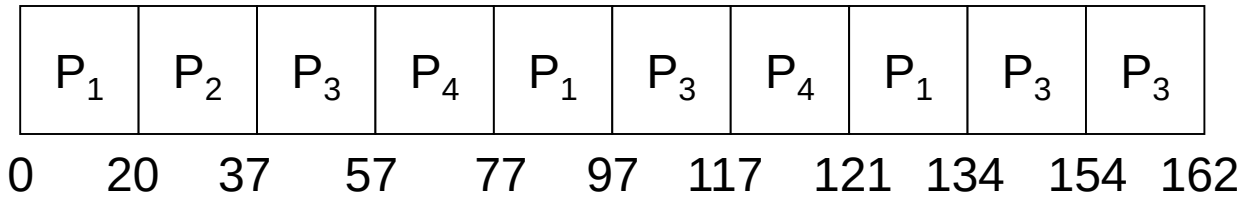
Round Robin (RR)

- 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:

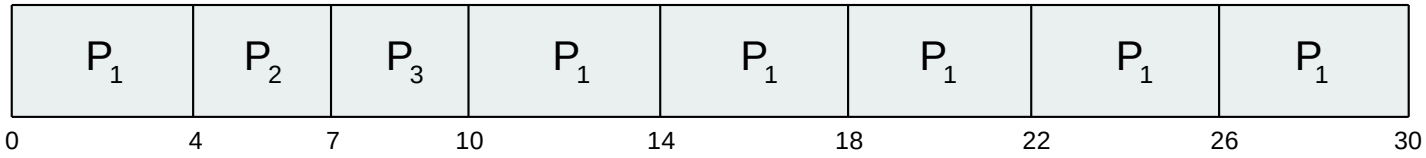


Typically, higher average turnaround than SJF, but better *response*.

Round Robin Example : Time Quantum = 4

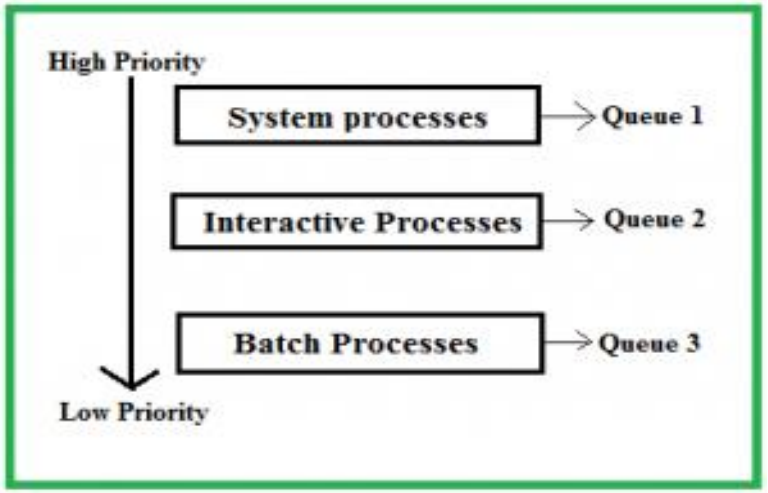
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



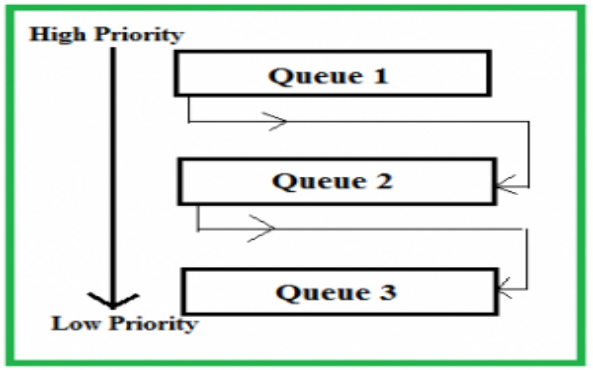
P1 waits for 6 millisconds (10- 4)
P2 waits for 4 millisconds
P3 waits for 7 millisconds.
Thus, the average waiting time is 17/3 = 5.66 mss.

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

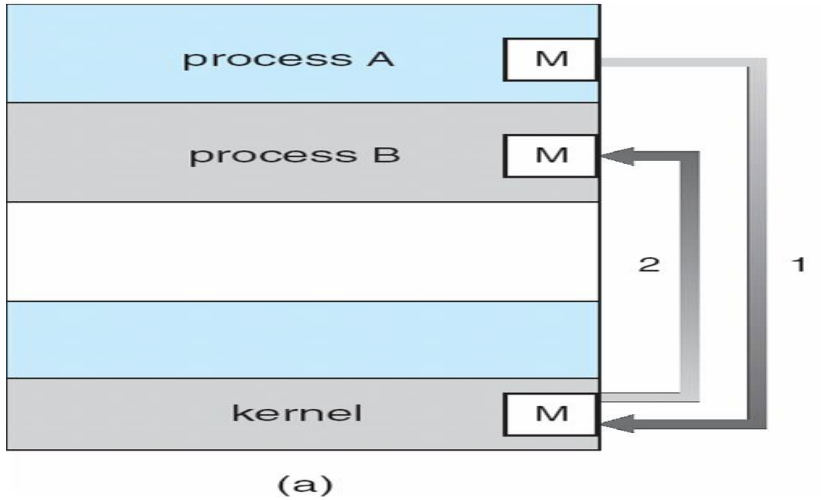
- do not get affected by the execution of another process

Cooperating process

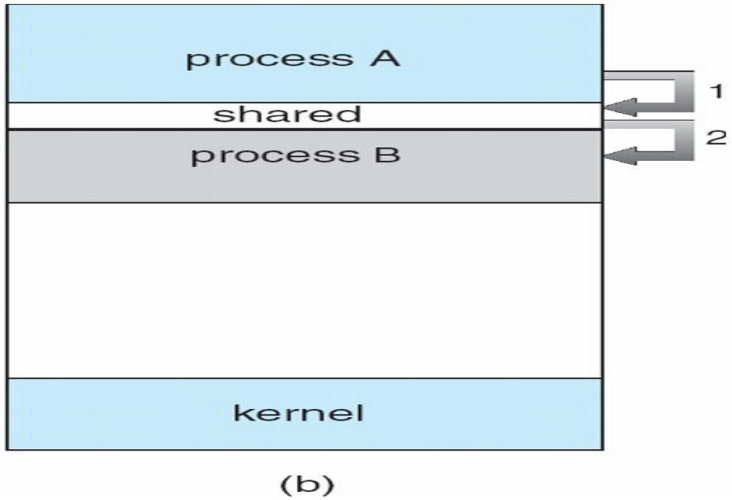
- get affected by the execution of another process

- 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.

Message Passing Mechanisms

Pipe

- Allowing two processes to communicate in unidirectional way
- by using pipe one process can send message to another process at a time

Message Queue

- It is bidirectional communication
- processes communicate with each other through message primitives.
  - **send***(destination, message*) or **send**(*message*)
  - **receive***(source, message*) or **receive**(*message*)

Signal

- Processes communicates with each other by means of signals.
- Signals have predefined meaning
- Eg. SIGTERM (Terminate) SIGKILL, SIGSTOP, SIGCONT etc....

Socket

- A **socket** is defined as an *endpoint for communication*
- e.g.Host to Webserver
- socket = ip addr + port number
- e.g. chat application

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;
```

Solution to Critical-Section Problem

Mutual Exclusion.

- If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

Progress

- If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

Bounded Waiting

- A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

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.**

What is Semaphore?

- **Semaphore** is simply a variable that is non-negative and shared between threads.
- A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread.
- It uses two atomic operations, 1) Wait, and 2) Signal for the process synchronization.
- A semaphore either allows or disallows access to the resource, which depends on how it is set up.

Example on Semaphore

Example 1 : A counting semaphore S is initialized to 10. Then, 6 P operations and 4 V operations are performed on S. What is the final value of S?

P operation also called as wait operation decrements the value of semaphore variable by 1.
V operation also called as signal operation increments the value of semaphore variable by 1.

Final value of semaphore variable S
= 10 – (6 x 1) + (4 x 1)
= 10 – 6 + 4
= 8

Example 2 : A counting semaphore S is initialized to 7. Then, 20 P operations and 15 V operations are performed on S. What is the final value of S?
Final value of semaphore variable S
= 7 – (20 x 1) + (15 x 1)
= 7 – 20 + 15
= 2