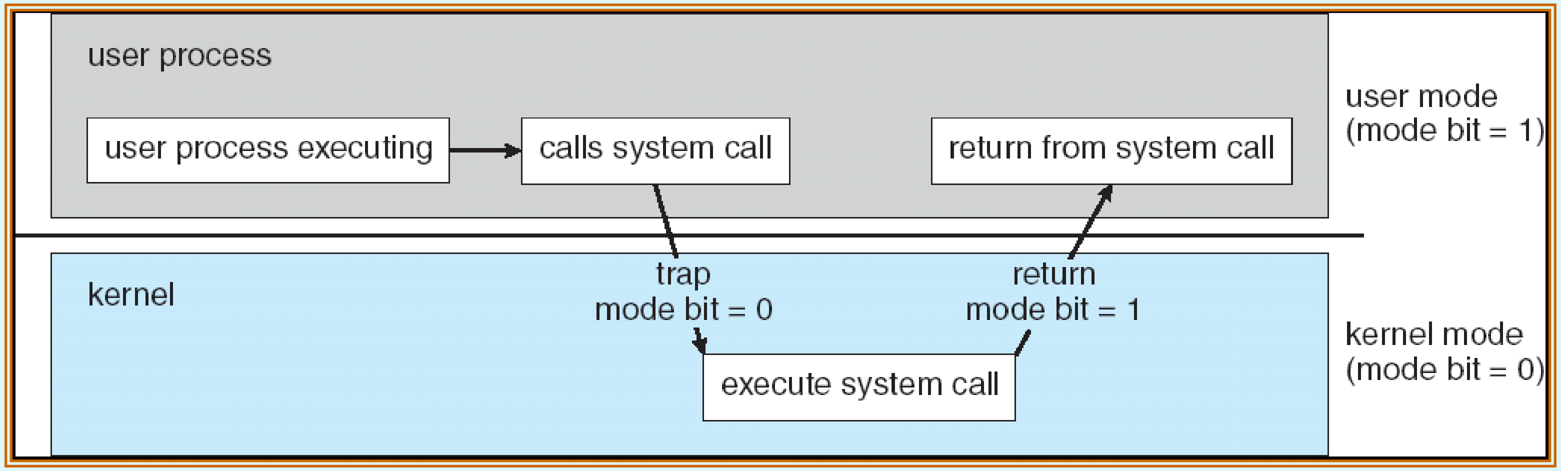
Operating System
Day 2

BY:
MRS. AKSHITA. S. CHANCHLANI
SUNBEAM INFOTECH
AKSHITA.CHANCHLANI@SUNBEAMINFO.COM

Dual Mode Operation

- Allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user mode or kernel mode
    - Some instructions designated as **privileged**, only executable in kernel mode
- To perform privileged operations, must transit into OS through well defined interfaces
  - System calls
  - Interrupt handlers



CPU Modes

System Mode/ Kernel Mode

- privileged mode/master mode/supervisor mode
  - Can execute any instruction
  - Can access any memory locations, e.g., accessing hardware devices,
  - Can enable and disable interrupts
  - Can change privileged processor state
  - Can access memory management units
  - Can modify registers for various descriptor tables

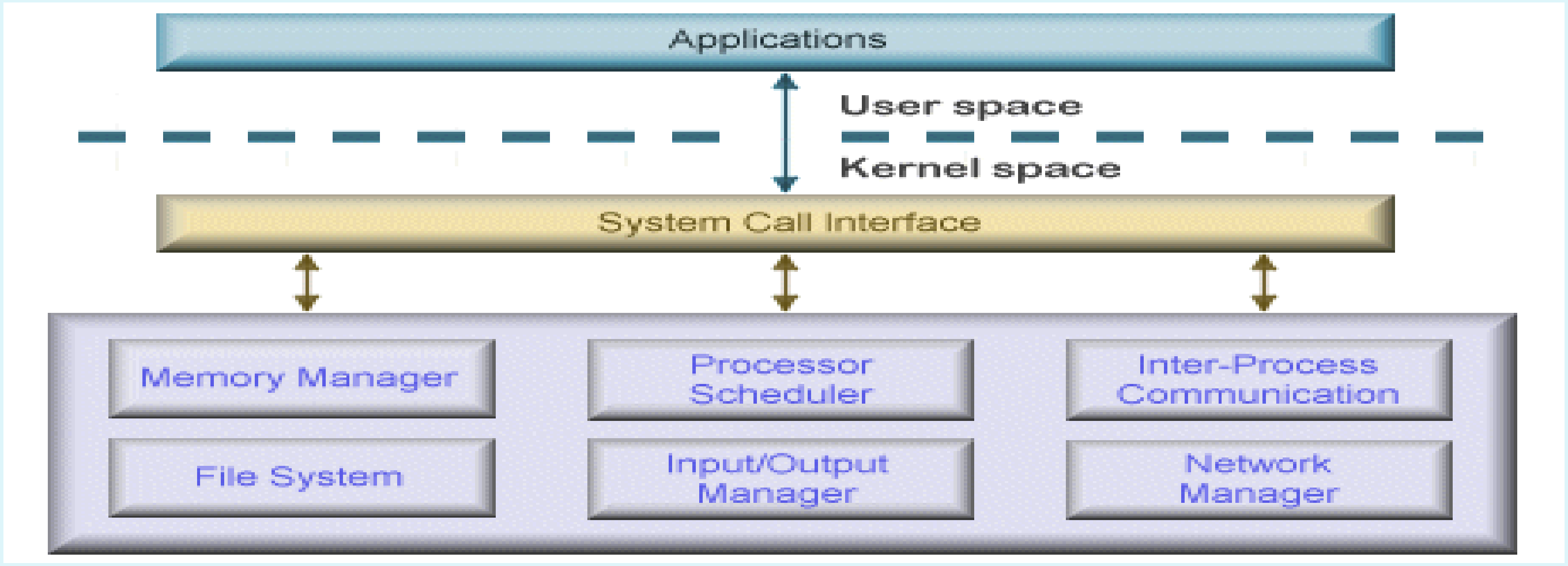
User Mode

- Unprivileged Mode
  - Access to memory is limited,
  - Cannot execute some instructions
  - Cannot disable interrupts,
  - Cannot change arbitrary processor state,
  - Cannot access memory management units

Transition from user mode to system mode must be done through well defined call gates (system calls)

Kernel space vs User space

- Part of the OS runs in the kernel model
  - known as the OS kernel
- Other parts of the OS run in the user mode, including service programs , user applications, etc.
  - they run as processes
  - they form the user space (or the user land)



User Mode VS Kernel Mode

User Mode vs Kernel Mode	
User Mode is a restricted mode, which the application programs are executing .	Kernel Mode is the privileged mode, which the computer enters when accessing hardware resources.
Modes	
User Mode is considered as the slave mode or the restricted mode.	Kernel mode is the system mode, master mode or the privileged mode.
Address Space	
In User mode, a process gets their own address space.	In Kernel Mode, processes get single address space.
Interruptions	
In User Mode, if an interrupt occurs, only one process fails.	In Kernel Mode, if an interrupt occurs, the whole operating system might fail.
Restrictions	
In user mode, there are restrictions to access kernel programs. Cannot access them directly.	In kernel mode, both user programs and kernel programs can be accessed.

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 Management

- Operating systems provide fundamental services to processes including:
  - Creating processes
  - Destroying processes
  - Suspending processes
  - Resuming processes
  - Changing a process's priority
  - Blocking processes
  - Dispatching processes
  - Inter Process communication (IPC)

Memory Layout of Program and Process

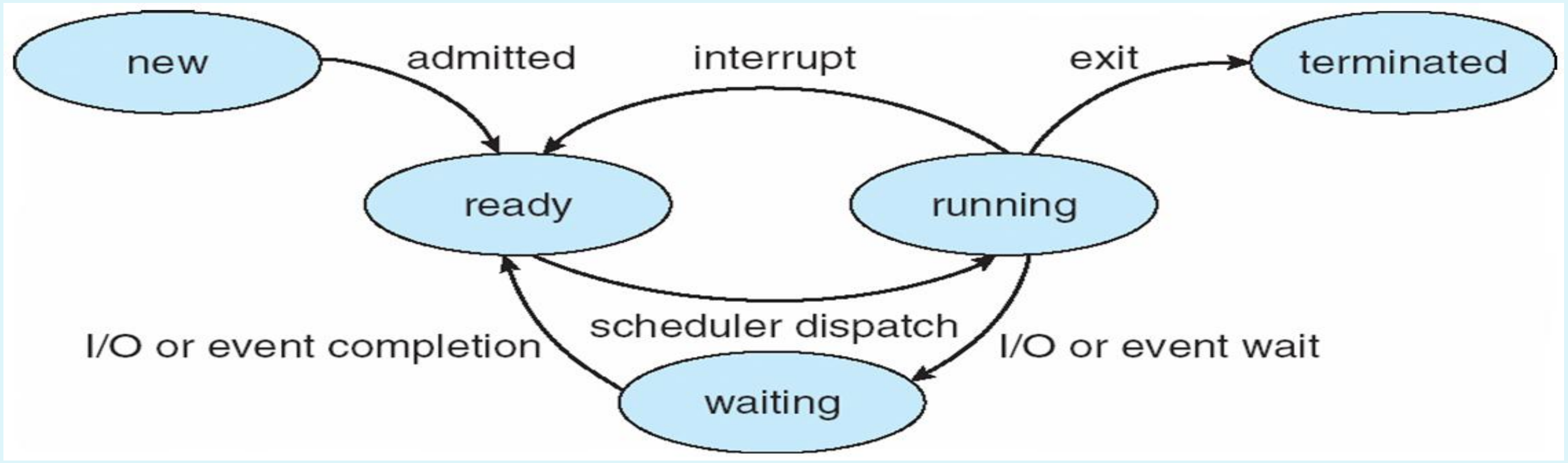
Program Consist of
exe header/primary header
Block started by symbol (bss) section (un initialized static / global variables)
Data section (initialized static / global variables)
Rodata Section(Constant/literals)
code/text section (contains executable instructions)
Symbol Table

Process Consist of
Skipped
Block started by symbol (bss) section (un initialized static / global variables)
Data section (initialized static / global variables)
Rodata Section(Constant/literals)
code/text section (contains executable instructions)
Skipped
Stack Section
Heap Section

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.
- 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



- As a process executes, it changes *state*
  - **new**: The process is being created
  - **ready:** The process is waiting to be assigned to a processor.
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **terminated**: The process has finished execution

Data Structures Maintained by Kernel at the time of process Execution

- Job Queue
- Ready Queue
- Waiting Queue

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

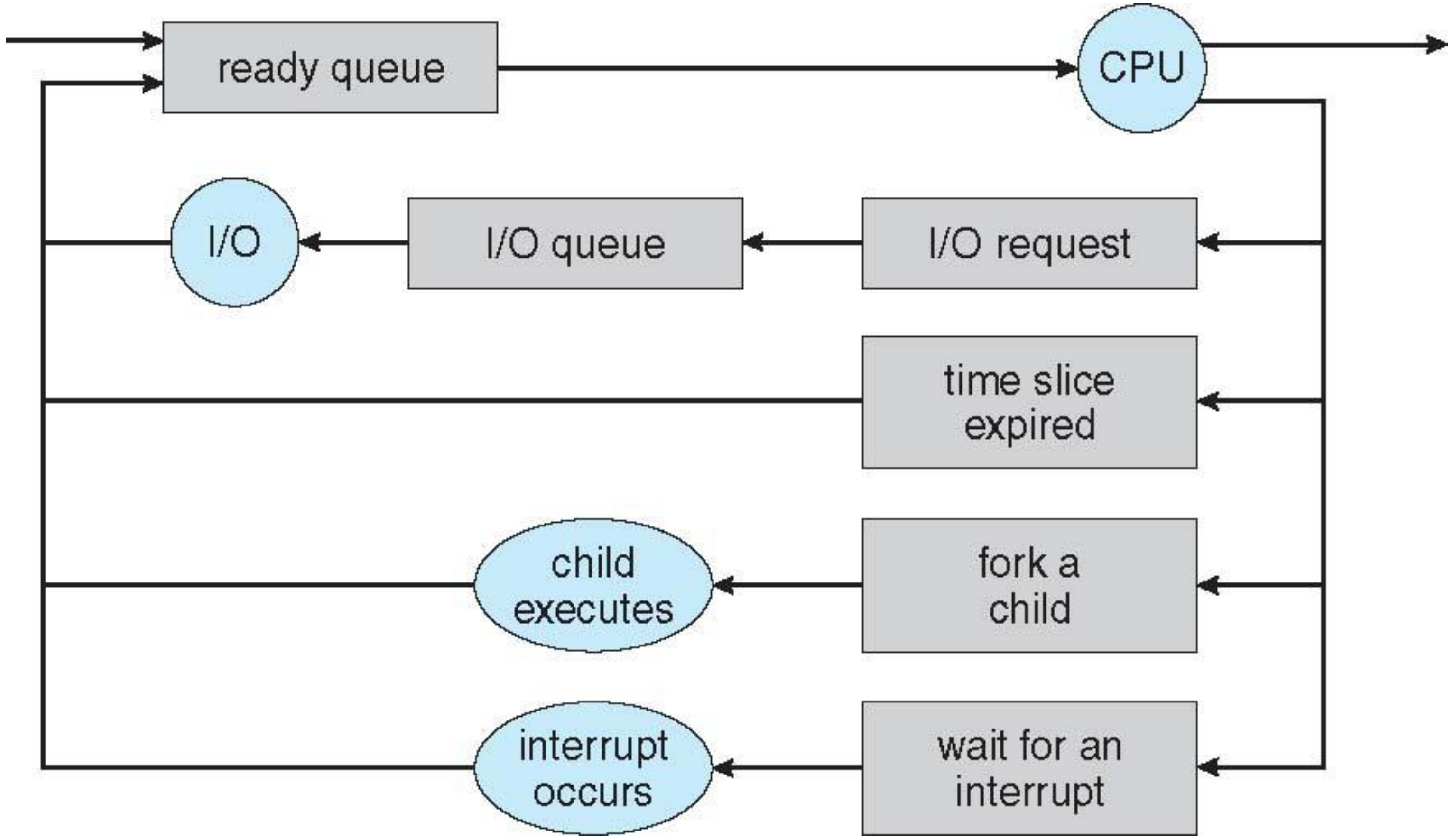
Schedulers

- Job Scheduler/long term schedulers :
  - Selects which processes should be brought into the ready queue
- CPU Scheduler/Short term schedulers
  - Process from ready queue to load into CPU
  - selects which process should be executed next and allocates CPU

Dispatcher

- Gives control of the CPU to the process which is scheduled by the CPU scheduler
- time taken by the dispatcher to stops execution and one process and starts execution of another process is called as "dispatcher latency".

Representation of Process Scheduling



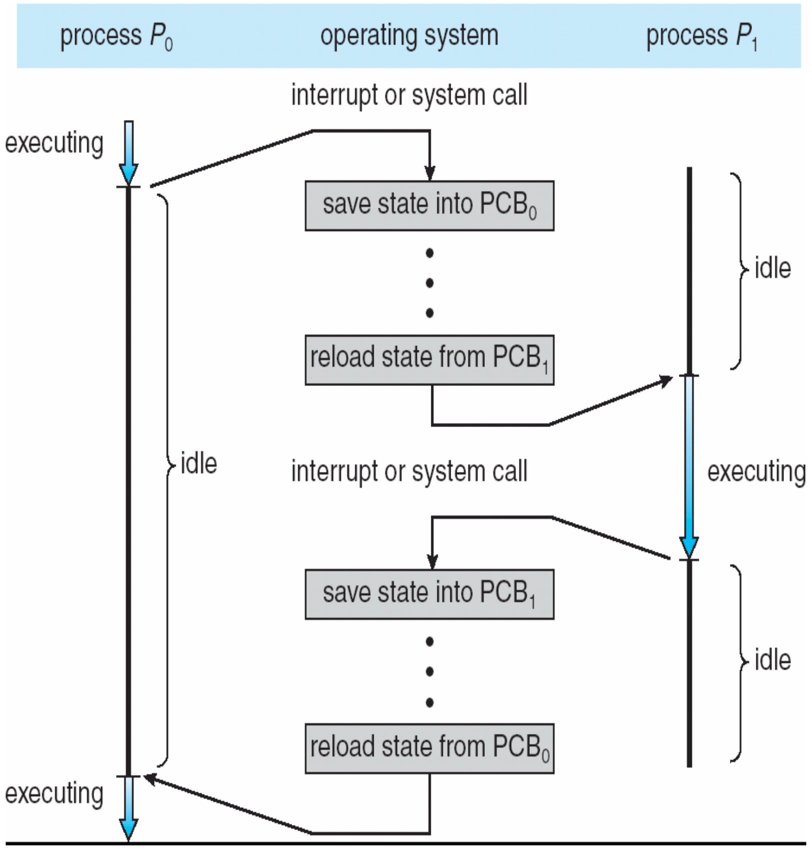
Context Switch

- 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.

Context Switching



CPU scheduler should get called in following four cases:
1.running --> terminated
2.running --> waiting
3.. running --> ready
4. waiting --> ready

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.

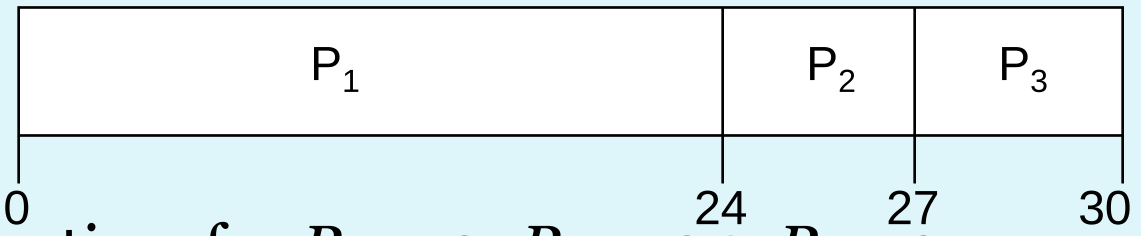
Execution Time/Running Time/CPU Burst Time

- It is the total amount of time spent by the process onto the CPU 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: $(0 + 24 + 27)/3 = 17$

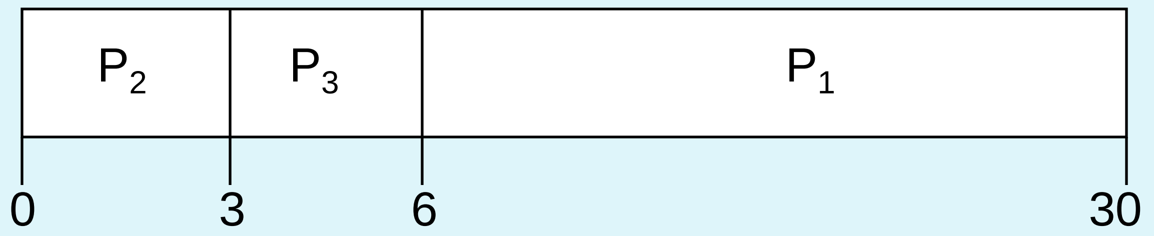
convoy effect
as all the other processes wait for the one big process to get off the CPU.

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

- The Gantt chart for the schedule is:



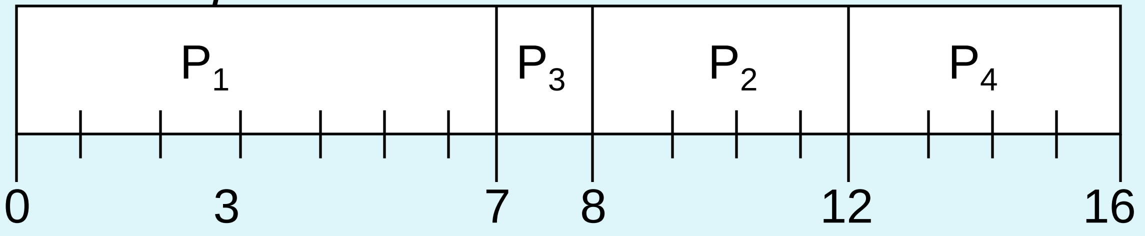
- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.
- *Convoy effect* short process behind long process

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

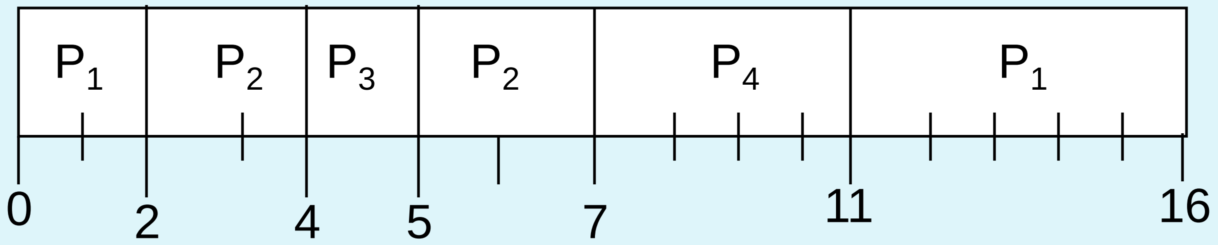
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- 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

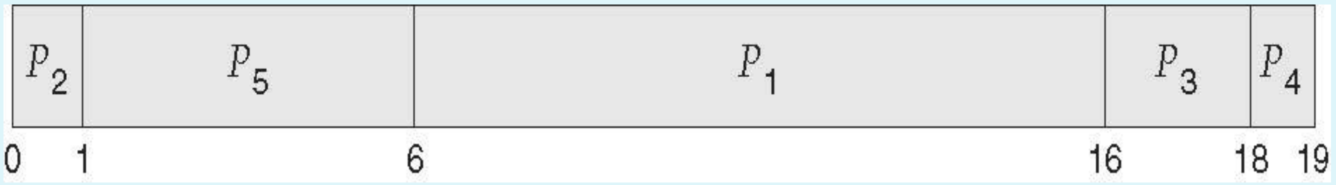
Priority Scheduling

- 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

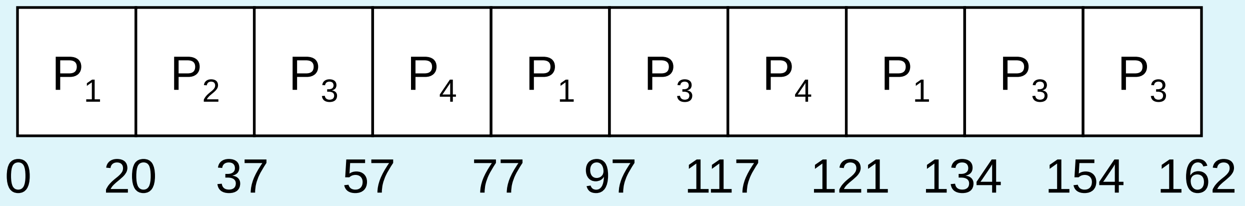
Round Robin (RR)

- 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:

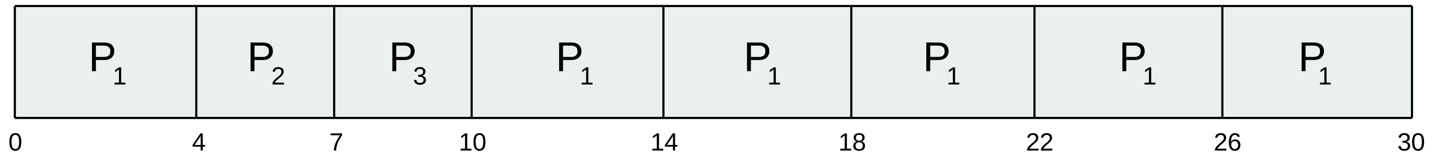


- Typically, higher average turnaround than SJF, but better *response*.

Round Robin Example : Time Quantum = 4

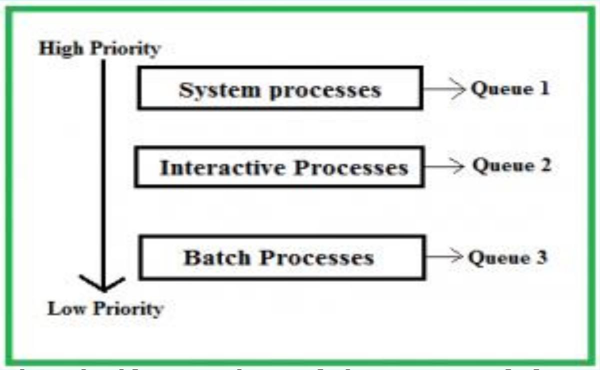
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



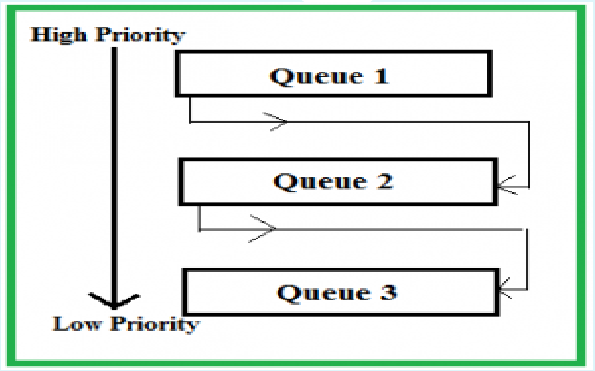
P1 waits for 6 millisconds (10- 4)
P2 waits for 4 millisconds
P3 waits for 7 millisconds.
Thus, the average waiting time is 17/3 = 5.66 mss.

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.

Features of OS

multi-programming

- system in which execution more than one programs/processes can be starts at a time.

multi-tasking/time sharing

- system in which the CPU can run/execute more than one programs processs/tasks concurrently/simultaneously.

multi-threading

- system in which the CPU can execute multiple threads of either same process or of different processes concurrently/simultaneously.

multi-user

- system in which the CPU can execute programs of multiple users concurrently

multi-processor

- system can run on such a machine in which more than one CPU's/processors are connected in a closed circuit.

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

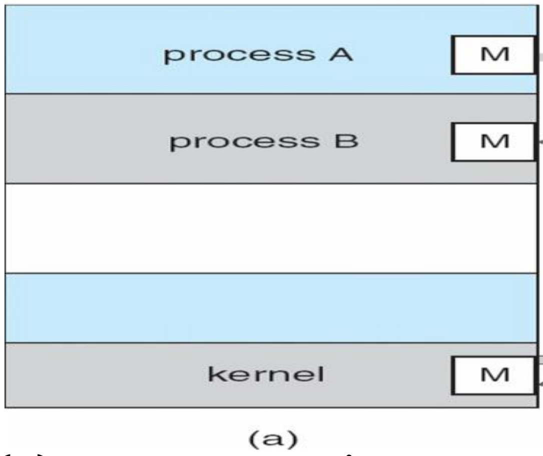
- **get affected by the execution of another process**

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

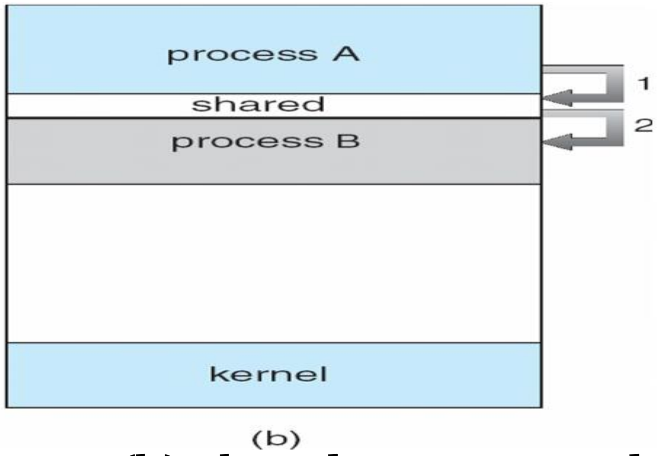
IPC Requirements

- If *P* and *Q* wish to communicate, they need to:
  - establish communication link between them.
  - exchange messages via send/receive.
- Implementation of communication link:
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

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
  - unnamed pipe --- two related processes can communicate with other.
  - **"related process"** -- processes which are of same parent
  - named pipe -- non-related processes can also communicate with each other.

Message Queue

- It is bidirectional communication
- processes communicate with each other through message primitives. (Send / Receive)

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

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.
- 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**

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

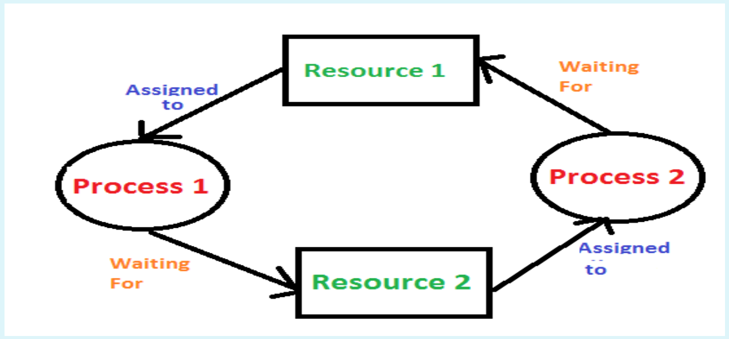
- If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

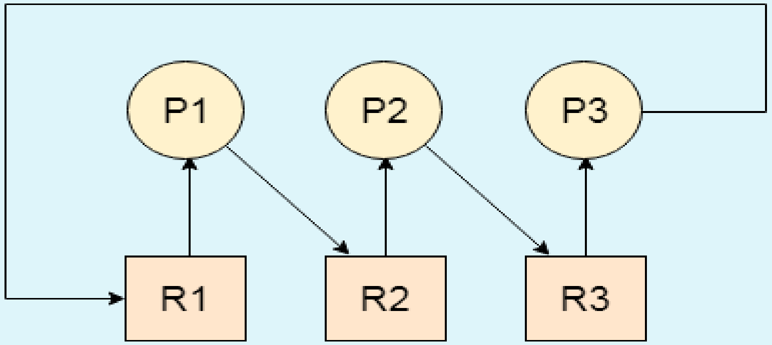
Bounded Waiting

- 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.

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.

- only the process can release its held resource
- A resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular wait

- A set { Po , 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 Detection
2. Deadlock Recovery
3. Deadlock Prevention
4. Deadlock Avoidance

Deadlock Detection

- Deadlock can be detected by the resource scheduler as it keeps track of all the resources that are allocated to different processes.
- After a deadlock is detected, it can be handed using the given methods:
  1. All the processes that are involved in the deadlock are terminated. This approach is not that useful as all the progress made by the processes is destroyed.
  2. Resources can be preempted from some processes and given to others until the deadlock situation is resolved.

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.

Deadlock Prevention

Deadlock prevention ensures that deadlock is excluded from the beginning by invalidate at least one of the four necessary conditions, however deadlock prevention is often impossible to implement.

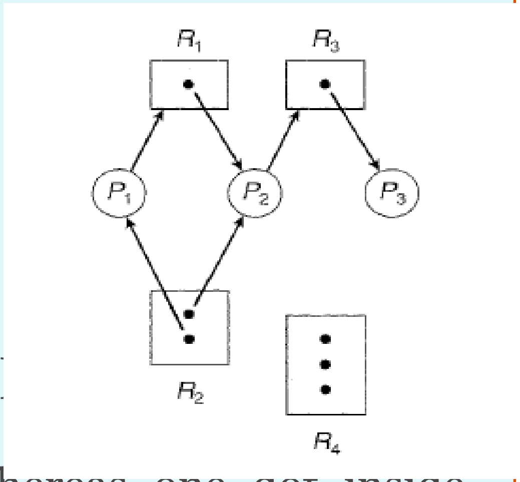
- **No Mutual Exclusion Condition**
- **No Hold and Wait**
- **Preemption**
- **No Circular Wait**

Deadlock Avoidance

- Deadlock avoidance algorithm ensures that a processes will never enter into unsafe or deadlock.
- The system requires additional **prior information** regarding potential use of each resource for each process that means each process declare the maximum number of resources of each type that it may need, number of available resources, allocated resources, maximum demand of the processes.
- If we allocated resources in a order such that requirement can be satisfied for each process and deadlock can not be occur then this state is called as **safe state.**

Deadlock Avoidance algorithms

- Resource allocation graph(RAG)
  - states resources is held by which process and which process is waiting for a resource of a particular type.
    - G(V,E)
    - V:
    - P = { P1, P2, P3 },
    - R = { R1, R2, R3 }
    - E:
  - request edge: { P1 -> R3, P2 -> R1, P3 -> R2 }
  - assignment edge: { R1 <- P1, R2 <- P2, R3 <- P3
  - process can be shown by using a "circle“
  - resources can be shown by using "rectangle", whereas one dot inside rectangle indicates only one instance of a resource is available, and two dots indicate two instances of the resource are available.



- Bankers algorithm
  - tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

THANK YOU!!