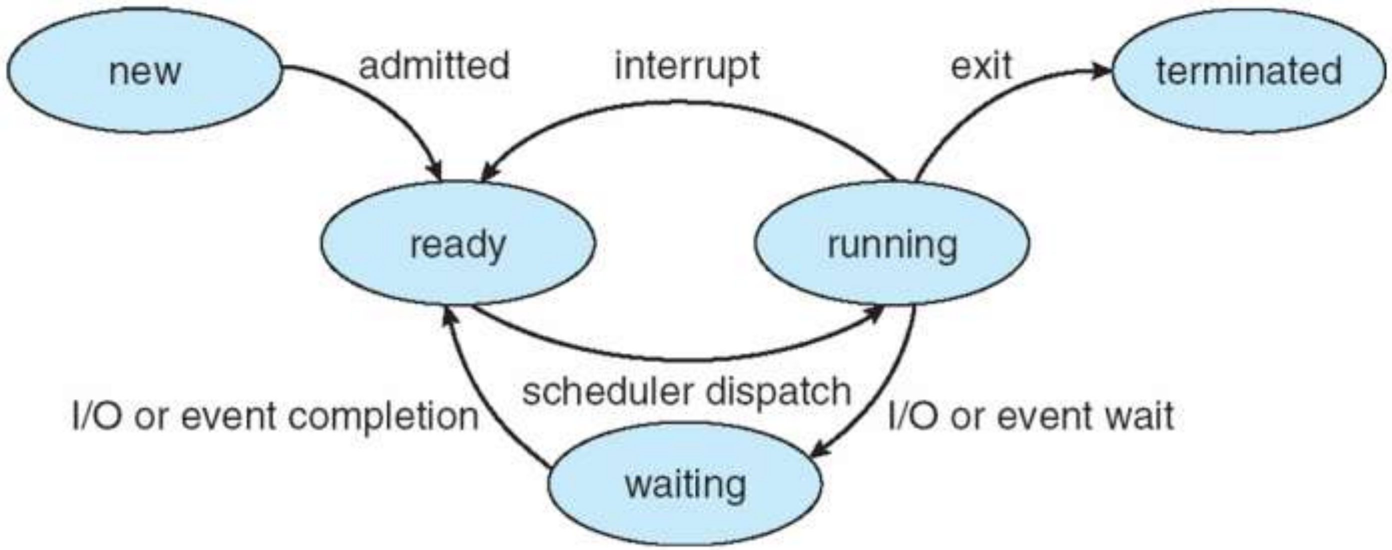
# Chapter 2:
# Process and Process Scheduling

# Process Concepts

- **Process** – a program in execution; process execution must progress in sequential fashion
- Process needs certain resources to accomplish its task
- Resources are allocated either while process creation or while process execution
- Process vs Program
- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
- Program becomes process when executable file loaded into memory
- Process is unit of work

# Diagram of Process State



# Process State

- 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

# Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

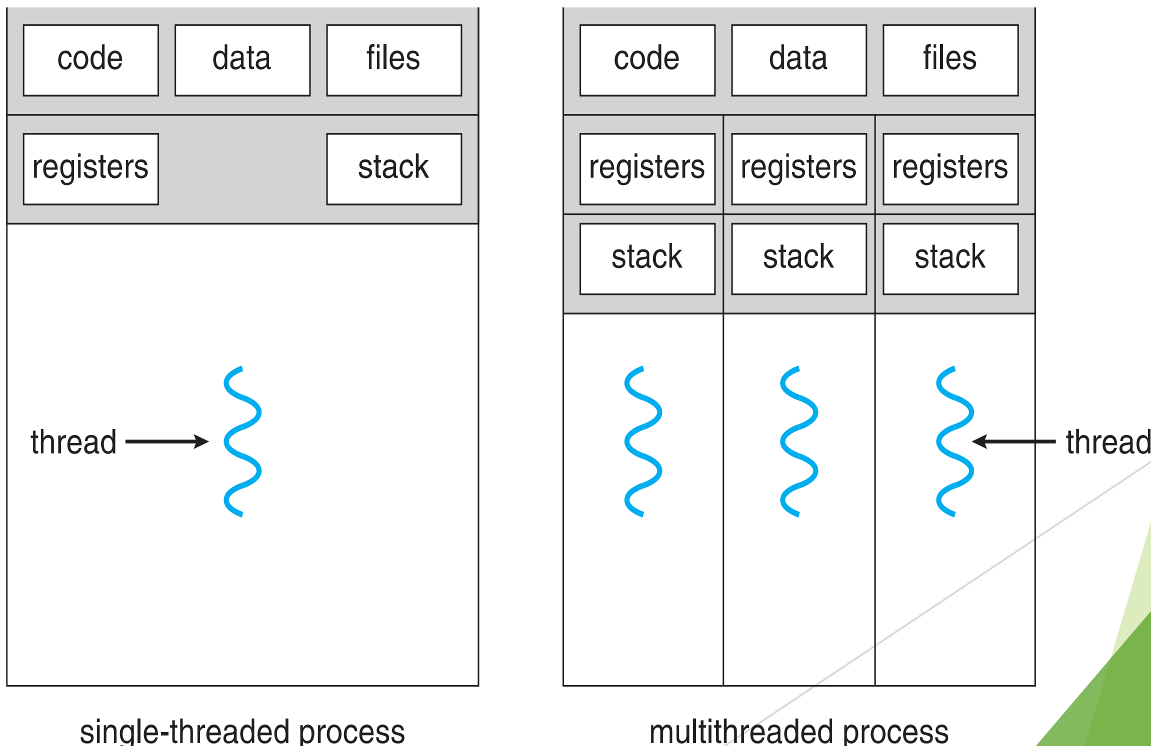
- Pointer – points to another PCB
- Process state – running, waiting, etc
- Process no. – unique id for each process
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Event info – info about the event for which the process is waiting.
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

# Threads

- basic unit of CPU utilization
- it comprises a thread ID, a program counter, a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- Lightweight process
- If a process has multiple threads of control, it can perform more than one task at a time.
- Most software applications that run on modern computers are multithreaded
- Kernels are also generally multithreaded
- **Each thread belongs to exactly one process but a process can have multiple threads.**
- Threads of same process runs in a shared memory space.
- Every process starts with a single thread called as primary thread but can create additional threads as required.

# Threads

- Threads run within application
- Before threads different processes were created to perform different tasks.
- Now, multiple threads are created within a same process to handle different tasks.
- This is called as multithreading.
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency



# Benefits

- **Responsiveness** - Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. This quality is especially useful in designing user interfaces.
- **Resource Sharing** - threads share resources of process to which they belong by default. It is easier than shared memory or message passing. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
- **Economy** - Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.
- **Scalability** - The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.

# User Threads and Kernel Threads

**User threads** – thread management done by application & user-level threads library.

- Kernel unaware of thread's existence
- Faster to create & manage, easier thread switching.
- Can run on any OS
- System calls are blocking

**Kernel threads** - thread management done by kernel

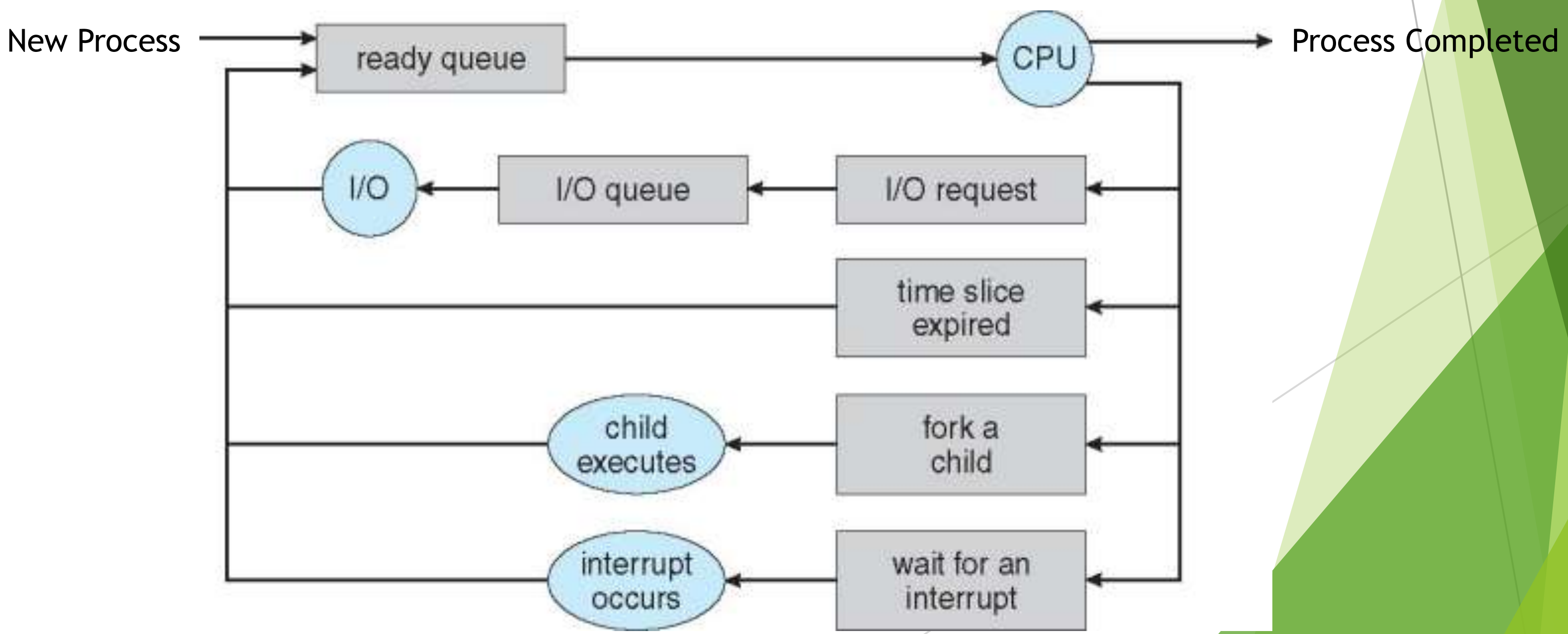
- Directly supported by OS
- Slower to create & manage
- No blocking
- Windows, Linux, Mac OS support kernel threads

# Process Scheduling

- In a single processor system only one process can run at a time. All other process must wait for CPU to be free
- Multiprogramming maximizes CPU utilization by scheduling another process to run when the running process is waiting (for I/O or completion of some event)
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- Scheduling is a fundamental OS functionality.
- All computer resources are scheduled before use
- **Process scheduler** selects among available processes for next execution on CPU

# Scheduling Queues

- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues
- **Queueing diagram** represents queues, resources, flows



# Long Term Schedulers

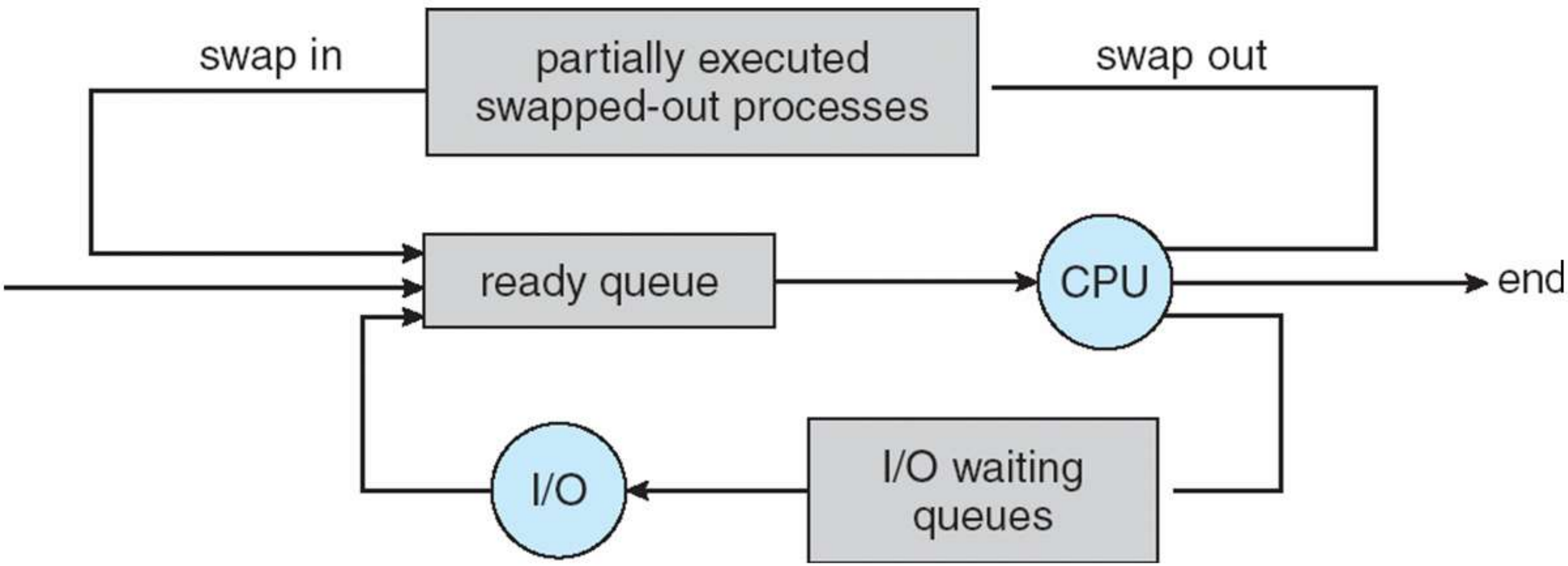
- In a batch system more process are submitted than can be executed immediately.
- These processes are spooled to a mass storage device, where they are kept for later execution.
- The long term scheduler or job scheduler select processes from this pool and loads them into the memory for execution.
- It also controls degree of multi programming.
- Avg rate of process creation = avg departure rate of processes leaving the system
- Executes less frequently
- Long term scheduler is used when process changes state from new to ready state

# Short Term Schedulers

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
- Short term scheduler is used when process changes state from ready to running state
- Also known as Dispatcher
- Executes most frequently
- It is fast

# Medium Term Scheduler

- **Medium-term scheduler** is an intermediate level of scheduling that may be introduced by some time sharing OS
- Key idea – Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**
- Swapping is done by Medium Term Scheduler



# Context Switching

- 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
- State save and state restore is performed
- State save is performed of current process while state restore is performed of some other process.
- Context-switch time is overhead; the system does no useful work while switching

# Preemptive and Non-Preemptive Scheduling

- Different Scheduling algos. Use different criterias for selecting process from ready queue
- Scheduling algos. can be preemptive or no –preemptive
- Four circumstances –

1) Process switches from running to waiting state
2) Process switches from running to ready state
3) Process switches from waiting to ready state
4) Process terminates

- Circumstances 1) and 4) – Non-Preemptive
- Circumstances 2) and 3) – Preemptive
- In **Preemptive Scheduling** running process is forcibly terminated and replaced by higher priority process at any time
- In **Non-Preemptive Scheduling** once allocated, the CPU is released by the process either on termination or by switching to waiting state. There is no forced termination.

# Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – Number of processes that complete their execution per unit time
- **Turnaround time** – amount of time taken from submission of process to the time of its completion
- **Waiting time** – amount of time a process spends waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced.
- **Priority -** Preferential treatment to processes with higher priorities
- **Balanced Utilization -** Overall utilization of all the resources
- **Fairness -** All processes should be given equal opportunity to execute

# Scheduling Algorithms

- Used to decide which process in the ready queue is to be allocated CPU
- Various scheduling Algorithms –

1) FCFS
2) SJF
3) SRT (Preemptive SJF)
4) Priority Scheduling (Non Preemptive)
5) Priority Scheduling (Preemptive)
6) Round Robin

# 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 ms ; $P_2$ = 24 ms ; $P_3$ = 27 ms
- Average waiting time: (0 + 24 + 27)/3 = 17 ms
- Average Turnaround time (Burst time + waiting time) : (24+27+30)/3 = 27 ms

# 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 ms
- Average Turnaround time : (30 + 3 + 6)/3 = 13 ms
- Much better than previous case
- **Convoy effect** - short process behind long process

Q) Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time when the processes arrive in the following order:

P1,P2,P3,P4,P5

| PROCESS | BURST TIME |
|---|---|
| P1 | 4 |
| P2 | 7 |
| P3 | 3 |
| P4 | 3 |
| P5 | 5 |

Waiting Time: 9.2
TAT: 13.60

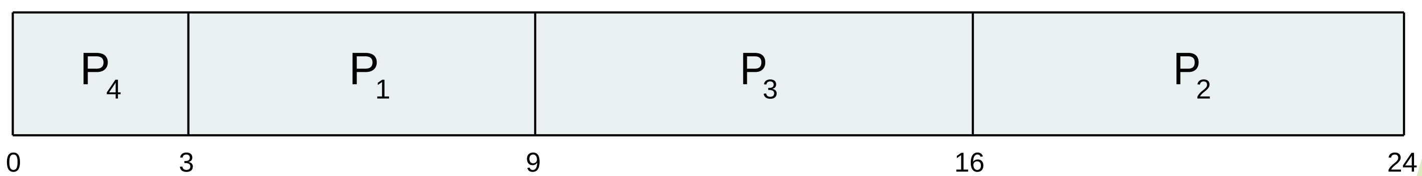
# Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

# Example of SJF

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |

- SJF scheduling chart




- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7
- Average Turnaround time = (9 + 24 + 16 + 3) / 4 = 13

Q) Calculate the average waiting time and turnaround time.

| PROCESS | BURST TIME |
| --- | --- |
| P1 | 5 |
| P2 | 2 |
| P3 | 6 |
| P4 | 4 |

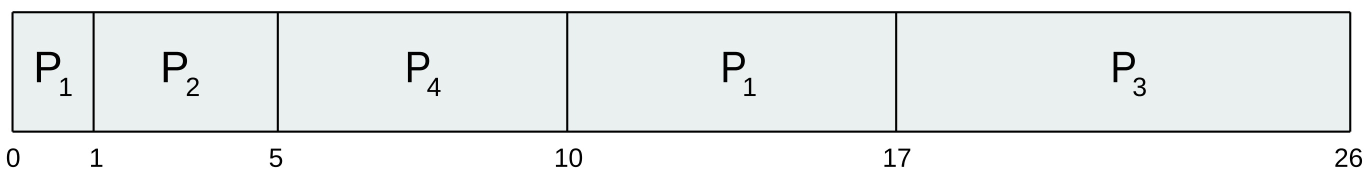
Waiting Time: 4.75
TAT: 9

# Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

| Process | *Arrival* Time | Burst Time |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

- *Preemptive* SJF (i.e. SRT) Gantt Chart




- Average waiting time = [((10-1)-0)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 msec
- Average Turnaround time = (17+4+24+7)/4 = 52/4 = 13 msec

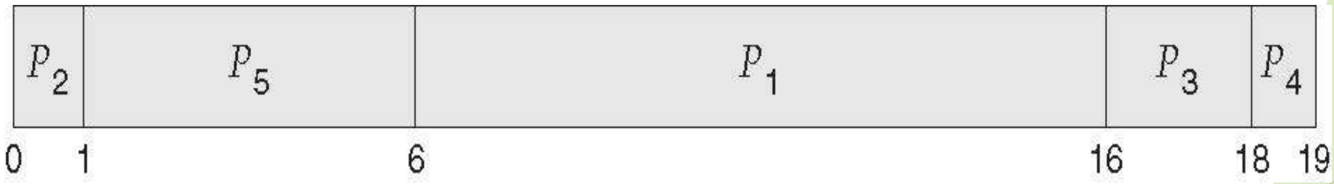
# Priority Scheduling

- 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

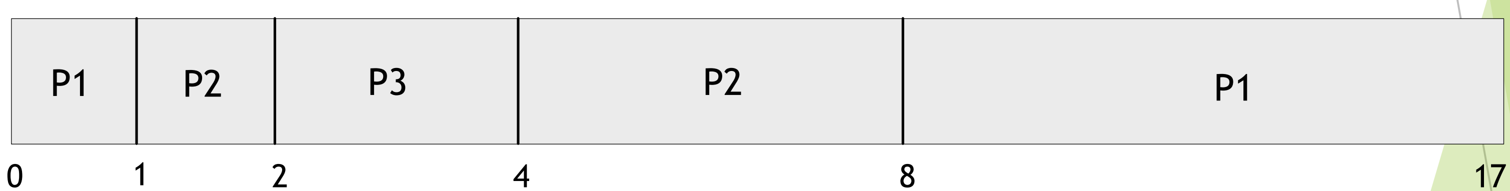



- Average waiting time = 8.2 msec

# Preemptive Priority Scheduling

| Process | Burst Time | Arrival Time | Priority |
|---|---|---|---|
| P1 | 10 | 0 | 3 |
| P2 | 5 | 1 | 2 |
| P3 | 2 | 2 | 1 |

- Priority scheduling Gantt Chart




- Average waiting time = [((8-1)-0)+(4-1-1)+(2-2)]/3 = 9/3 = 3 msec
- Average Turnaround time = (17+7+2)/3 = 26/3 = 8.6 msec

# Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** $q$), usually 10-100 milliseconds.
- After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are $n$ processes in the ready queue and the time quantum is $q$, then each process gets $1/n$ of the CPU time in chunks of at most $q$ time units at once.
- No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Ready queue is maintained as a fifo queue
- Performance
  - $q$ large $\Rightarrow$ FIFO
  - $q$ small $\Rightarrow$ $q$ must be large with respect to context switch, otherwise overhead is too high

# Example of RR

| <u>Process</u> | <u>Burst Time</u> |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Time Quantum = 4 msec
- The Gantt chart is:

| $P_1$ | $P_2$ | $P_3$ | $P_1$ | $P_1$ | $P_1$ | $P_1$ | $P_1$ |
|---|---|---|---|---|---|---|---|

0 4 7 10 14 18 22 26 30

- Average Waiting Time = ((10-4)+4+7)/3 = 17/3 = 5.6 msec
- Average Turnaround Time = (30+7+10)/3 = 47/3 = 15.6 msec

# Comparison of Scheduling Algorithms

| Algorithm | Policy type | Advantages | Disadvantages |
| --- | --- | --- | --- |
| FCFS | Non-Preemptive | 1. Easy to implement<br>2. Minimum Overhead | 1. Unpredictable turnaround time<br>2. Avg waiting time is more |
| RR | Preemptive | 1. Provides fair CPU allocation<br>2. Provides reasonable response time to interactive users | 1. Requires selection of optimum time slice |
| Priority | Preemptive or non-preemptive | 1. Ensures fast completion of important jobs | 1. Indefinite postponement of some jobs<br>2. Starvation problem |
| SJF | Preemptive or non-preemptive | 1. Minimize average waiting time<br>2. It is an optimal algorithm | 1. Indefinite postponement of some jobs<br>2. Unknown burst times of all jobs |

Q) Calculate the average waiting time and turnaround time for FCFS, SJF, Non-Preemptive Priority & Round Robin Process Scheduling Algorithms. Processes arrived in the order P1, P2, P3, P4, P5 at time 0. Time Quantum = 1ms

| PROCESS | BURST TIME | PRIORITY |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

Q) Calculate the average waiting time and turnaround time for FCFS, SJF, Non-Preemptive Priority & Round Robin Process Scheduling Algorithms. Processes arrived in the order P1, P2, P3, P4, P5 at time 0. Time Quantum = 1ms

| PROCESS | BURST TIME | PRIORITY |
|---|---|---|
| P1 | 10 | 3 |
| P2 | 1 | 1 |
| P3 | 2 | 3 |
| P4 | 1 | 4 |
| P5 | 5 | 2 |

| ALGORITHM | AVG. WT | AVG. TAT |
|---|---|---|
| FCFS | 9.6 | 13.4 |
| SJF | 3.2 | 7 |
| Priority | 8.2 | 12 |
| Round Robin | 5.4 | 9.2 |

Q) Calculate the average waiting time and turnaround time for FCFS, SRT & Round Robin Process Scheduling Algorithms. Time Quantum = 1ms

| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

Q) Calculate the average waiting time and turnaround time for FCFS, SRT & Round Robin Process Scheduling Algorithms. Time Quantum = 1ms

| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

| ALGORITHM | AVG. WT | AVG. TAT |
|---|---|---|
| FCFS | 8.75 | 15.25 |
| SRT | 6.5 | 13 |
| Round Robin | 12.75 | 19.25 |

Q) Calculate the average waiting time and turnaround time for Preemptive Priority Scheduling Algorithm.

| PROCESS | ARRIVAL TIME | BURST TIME | PRIORITY |
|---|---|---|---|
| P1 | 0 | 6 | 4 |
| P2 | 3 | 5 | 2 |
| P3 | 3 | 3 | 6 |
| P4 | 5 | 5 | 3 |

Q) Calculate the average waiting time and turnaround time for Preemptive Priority Scheduling Algorithm.

| PROCESS | ARRIVAL TIME | BURST TIME | PRIORITY |
|---|---|---|---|
| P1 | 0 | 6 | 4 |
| P2 | 3 | 5 | 2 |
| P3 | 3 | 3 | 6 |
| P4 | 5 | 5 | 3 |

| ALGORITHM | AVG. WT | AVG. TAT |
|---|---|---|
| Preemptive Priority | 6.5 | 11.25 |
| Preemptive Priority (Largest number highest priority) | 4.5 | 9.25 |