

# OPERATING SYSTEMS

## UNIT 2 — Detailed Notes

*Processes, Thread & Process Scheduling*

Topics covered: Process Definition & Relationship • Process States • PCB • Context Switching • Thread (States, Benefits, Types, Multithreading) • Scheduling Foundation & Objectives • Types of Schedulers • Scheduling Criteria • FCFS, SJF, RR (with Numericals)

 Green box = Sticky Note |  Red box = VVI Hotpoint |  Pink box = Numerical Practice Space |  Yellow box = Diagram Suggestion

# 1. Process

## 1.1 What is a Process? (Definition)

A Process is a program that is currently being executed (a program in action). A program is just a passive set of instructions saved on disk (like a .exe file), but the moment you run it, the OS loads it into memory, gives it resources (CPU time, memory, files), and it becomes a Process — an active entity.

● **Sticky Note:** Program = Passive (just code sitting on disk). Process = Active (code running, using CPU and memory right now). This exact line-difference is asked very commonly in exams.

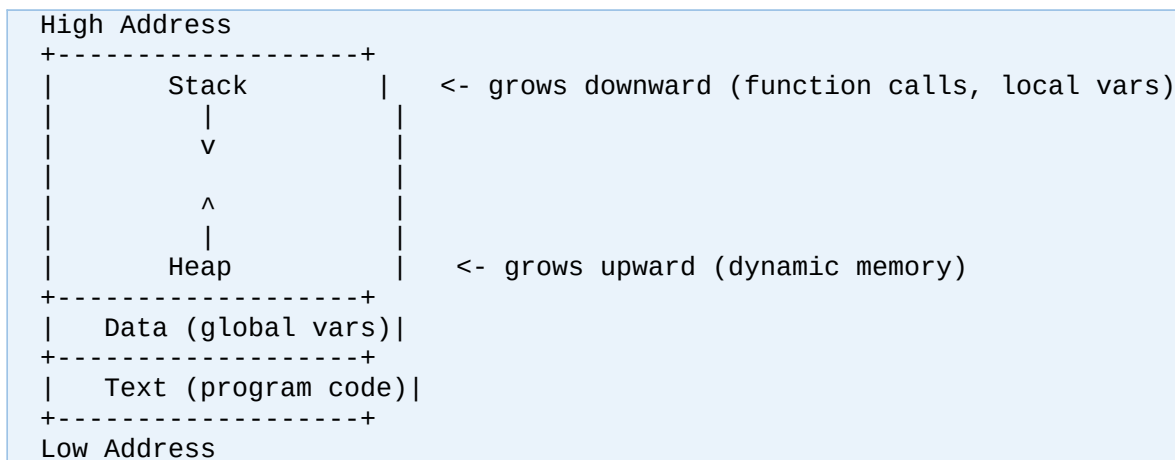
## 1.2 Process vs Program

| Basis     | Program                                     | Process                                                 |
|-----------|---------------------------------------------|---------------------------------------------------------|
| Nature    | Passive entity                              | Active entity                                           |
| Storage   | Stored on disk (secondary memory)           | Loaded into main memory (RAM)                           |
| Lifespan  | Exists permanently until deleted            | Exists only for a limited time (created and terminated) |
| Resources | Does not need CPU or memory to just 'exist' | Needs CPU, memory, I/O devices to execute               |
| Example   | notepad.exe file on your hard disk          | Notepad actually running and open on your screen        |

## 1.3 Structure of a Process in Memory

When a process is loaded into memory, it is divided into four main sections:

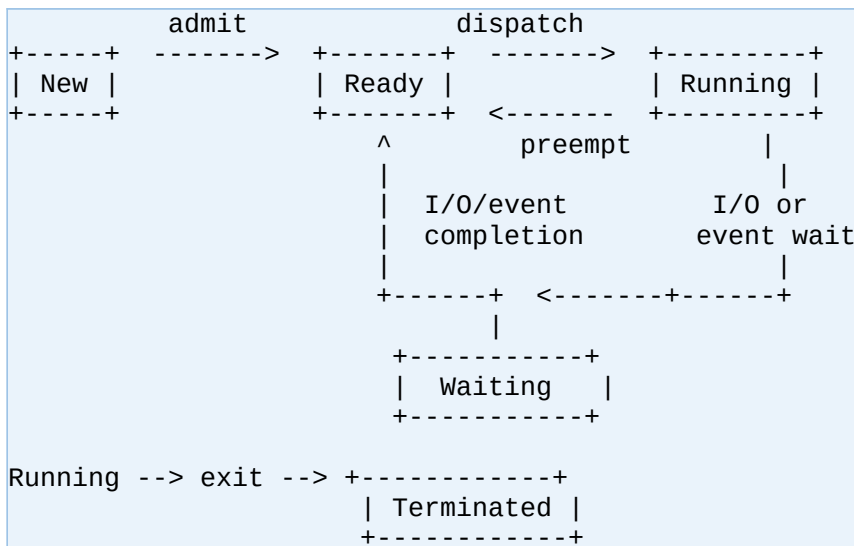
- **Text section:** Contains the actual compiled program code (the instructions).
- **Data section:** Contains global variables that are used throughout the program's life.
- **Heap:** Memory that is allocated dynamically while the program runs (e.g., using malloc() in C).
- **Stack:** Contains temporary data such as function parameters, return addresses, and local variables.



## 1.4 Process States (Process Lifecycle / Process Relationship)

As a process runs, it moves through different states depending on what it is doing. This movement is called the process life cycle:

| State             | Meaning                                                                                                |
|-------------------|--------------------------------------------------------------------------------------------------------|
| New               | The process is being created (just started).                                                           |
| Ready             | Process is in memory, waiting to be assigned to the CPU by the scheduler.                              |
| Running           | Process's instructions are actually being executed by the CPU.                                         |
| Waiting (Blocked) | Process is paused, waiting for some event to complete (like I/O finishing, or waiting for a resource). |
| Terminated (Exit) | Process has finished execution, or was killed, and is being removed from memory.                       |



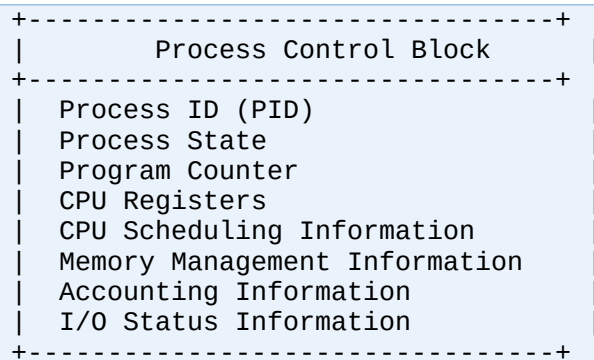
★ **VVI Hotpoint (Exam Important):** Process State Diagram (New -> Ready -> Running -> Waiting -> Terminated) is one of the MOST asked diagrams in exams. Practice drawing it by heart with all arrow labels (admit, dispatch, timeout, I/O wait, I/O completion, exit).

## 1.5 Process Control Block (PCB)

The Process Control Block (PCB) is a data structure maintained by the Operating System for every single process. It stores all the important information about that process, so the OS can pause it and resume it later exactly from where it stopped. It is sometimes also called a 'Task Control Block'.

Contents of a PCB:

- **Process ID (PID):** A unique number that identifies the process.
- **Process State:** Current state of the process — New, Ready, Running, Waiting, or Terminated.
- **Program Counter (PC):** Address of the next instruction to be executed for this process.
- **CPU Registers:** Values of all CPU registers (accumulator, index registers, etc.) used by the process — needed to resume execution correctly.
- **CPU Scheduling Information:** Process priority, pointers to scheduling queues.
- **Memory Management Information:** Base and limit register values, page tables, or segment tables used by this process.
- **Accounting Information:** Amount of CPU used, time limits, process number, etc.
- **I/O Status Information:** List of I/O devices allocated to this process, list of open files.

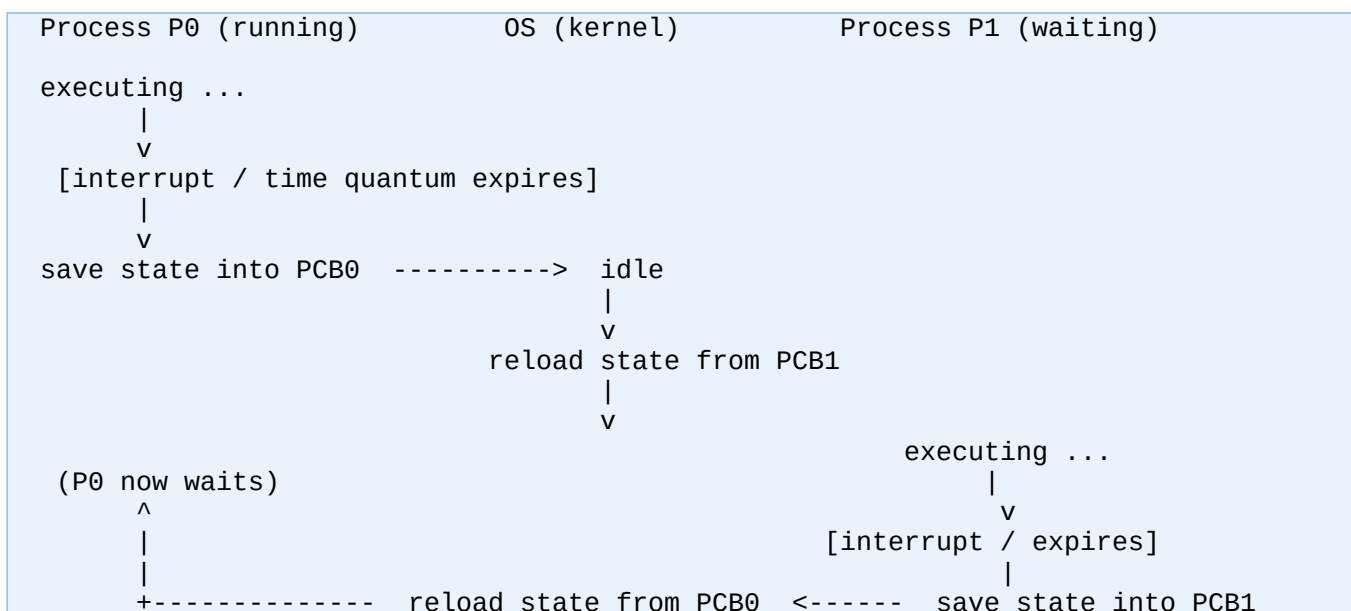


**Sticky Note:** Easy way to remember PCB contents: think 'PSPC-MAI' — Pid, State, PC, CPU registers, Memory info, Accounting info, I/O info. PCB is basically the 'identity card + progress card' of a process.

## 1.6 Context Switching

Context Switching is the process of saving the state (context) of a currently running process into its PCB, and loading the saved state of another process from its PCB, so that CPU can switch from one process to another.

- **Why it happens:** It happens whenever the CPU switches from one process to another — for example, when a time quantum expires, a higher priority process arrives, or the running process needs to wait for I/O.
- **How it works:** The OS saves the current process's Program Counter, CPU registers, and state into its PCB. Then it loads the next process's saved Program Counter and registers from its PCB, and CPU resumes that process exactly where it had left off.
- **Overhead:** Context switching itself does not do any useful work for the user — it is pure overhead (wasted time), so the OS tries to minimize how often it happens.



★ **VVI Hotpoint (Exam Important):** Context Switch time is considered PURE OVERHEAD because the CPU is not doing any actual process work during a switch — only saving/loading registers. Faster hardware = less context switch time = better system performance. This 'why it is overhead' point is commonly asked.

## 2. Thread

### 2.1 What is a Thread? (Definition)

A Thread is the smallest unit of execution inside a process. It is often called a 'lightweight process'. A single process can have multiple threads, and all threads of the same process share the same memory (code, data, files) but each thread has its own program counter, register set, and stack.

● **Sticky Note:** Simple way to think: a Process is like a whole factory, and Threads are like the workers inside that factory. All workers (threads) share the same factory building and resources (memory), but each worker does their own task at their own pace (own PC, own stack).

### 2.2 Thread States

A thread moves through similar states as a process: New (created), Ready (waiting for CPU), Running (executing on CPU), Waiting/Blocked (waiting for an event), and Terminated (finished execution). The state diagram is almost identical to the process state diagram shown earlier, applied at the thread level instead of the whole process.

### 2.3 Benefits of Threads

- **Responsiveness:** If one thread of a program is blocked (waiting), other threads of the same program can keep running, so the application stays responsive to the user.
- **Resource Sharing:** Threads share the memory and resources of the process they belong to, so sharing data between threads is easy and fast (no need for complex inter-process communication).
- **Economy (Low Cost):** Creating a new thread and switching between threads is much cheaper (faster, less overhead) than creating/switching a full new process, because threads don't need a separate memory space.
- **Scalability / Utilization of Multiprocessor Architecture:** On a multi-core CPU, different threads of the same process can run in parallel on different cores, making the program run faster.

### 2.4 Types of Threads

| Type                      | Description                                                                                                                                                    | Advantage                                                                                                                                         | Disadvantage                                                                                                            |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| User-Level Thread (ULT)   | Managed entirely by a user-level thread library, without kernel involvement. The kernel is not even aware individual threads exist — it only sees one process. | Fast to create and switch (no kernel mode switch needed); portable across OS.                                                                     | If one thread makes a blocking system call, the ENTIRE process blocks (since kernel does not know about other threads). |
| Kernel-Level Thread (KLT) | Managed directly by the Operating System kernel. The kernel is aware of, creates, schedules, and manages each thread individually.                             | If one thread blocks, other threads of the same process can still run, since kernel manages each separately; can use multiple CPU cores properly. | Slower to create/switch, since every thread operation needs a kernel mode switch (more overhead).                       |

## 2.5 Concept of Multithreading & Multithreading Models

Multithreading means a single process contains multiple threads that execute concurrently (or in parallel on multiple cores), sharing the same code, data, and resources. To connect user-level threads to kernel-level threads, the OS uses one of these models:

| Model        | How it works                                                                              | Effect                                                                                                                                                               |
|--------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Many-to-One  | Many user-level threads map to a single kernel thread.                                    | Thread management is fast (done in user space) but if one thread blocks, the whole process blocks; no true parallelism on multi-core CPUs.                           |
| One-to-One   | Each user-level thread maps to its own separate kernel thread.                            | More concurrency — if one thread blocks, others can still run, and multiple threads can run in parallel on multi-core CPUs. But creating many threads adds overhead. |
| Many-to-Many | Many user-level threads are multiplexed onto a smaller or equal number of kernel threads. | Best of both worlds — developer can create as many user threads as needed, while OS can schedule the kernel threads efficiently across multiple cores.               |

 **Diagram Suggestion (search on Google Images):** Draw 3 small diagrams side by side: (1) Many-to-One: 4 small circles (user threads) all pointing with arrows into 1 big circle (kernel thread). (2) One-to-One: 3 small circles each with its own separate arrow into 3 separate big circles. (3) Many-to-Many: 4 small circles pointing into 2 big circles (some sharing). Label each diagram clearly. Search Google Images: "many to one one to one many to many threading model diagram".

## 2.6 Process vs Thread — Key Differences

| Basis                | Process                                                    | Thread                                                                          |
|----------------------|------------------------------------------------------------|---------------------------------------------------------------------------------|
| Definition           | A program under execution                                  | A lightweight sub-part of a process                                             |
| Memory               | Has its own separate memory space                          | Shares memory space with other threads of the same process                      |
| Creation Time / Cost | Slower and costlier to create                              | Faster and cheaper to create                                                    |
| Communication        | Needs Inter-Process Communication (IPC) — slower           | Can directly share data with other threads — faster                             |
| Crash Effect         | One process crashing does not affect other processes       | If one thread crashes, it can crash the entire process (since memory is shared) |
| Context Switch Cost  | Higher (full memory map, registers, etc. must be switched) | Lower (only registers, stack, and program counter need to be switched)          |

 **VVI Hotpoint (Exam Important):** Very commonly asked short question: 'Why is thread called a lightweight process?' Answer: Because threads share the memory/resources of their parent process, so creating and switching threads needs much less overhead compared to creating/switching a full process.

## 3. Process Scheduling — Foundation & Concepts

### 3.1 Foundation — Why Scheduling is Needed

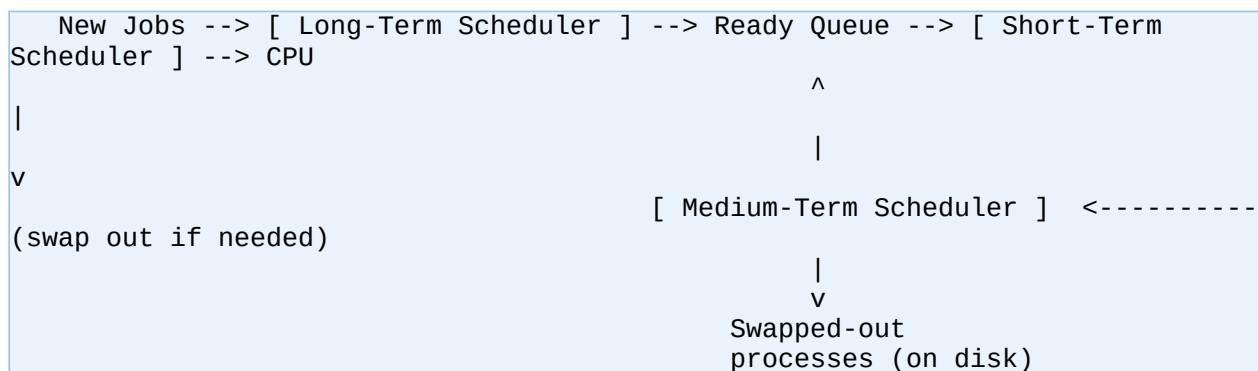
In a multiprogramming system, many processes are kept in memory at the same time, but a single CPU (or a limited number of CPUs) can run only one process at a time (per core). Process Scheduling is the activity of the OS that decides WHICH process gets the CPU and WHEN, so that CPU is used efficiently and no process waits forever.

### 3.2 Scheduling Objectives

- Maximize CPU utilization — keep the CPU as busy as possible, avoid idle time.
- Maximize throughput — complete as many processes as possible in a given time.
- Minimize turnaround time — reduce the total time taken to execute a process from submission to completion.
- Minimize waiting time — reduce the time a process spends waiting in the ready queue.
- Minimize response time — reduce the time from submitting a request to getting the first response (important for interactive systems).
- Ensure fairness — give every process a fair share of the CPU, avoid starvation.

### 3.3 Types of Schedulers

| Scheduler                            | Function                                                                                                                                          | Frequency                           | Purpose                                               |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------------------------------------------------|
| Long-Term Scheduler (Job Scheduler)  | Selects which processes/jobs should be brought from the disk (job pool) into main memory for execution (controls degree of multiprogramming).     | Runs less frequently (minutes)      | Controls how many processes are in the system at once |
| Short-Term Scheduler (CPU Scheduler) | Selects which process (already in Ready queue, in memory) will get the CPU next.                                                                  | Runs very frequently (milliseconds) | Must be extremely fast since it runs so often         |
| Medium-Term Scheduler                | Temporarily removes a process from memory (swaps it out to disk) to reduce the degree of multiprogramming, and can bring it back later (swap in). | Runs occasionally, as needed        | Helps manage memory when system is overloaded         |



### 3.4 Scheduling Criteria (How we measure a scheduling algorithm)

| Criteria              | Meaning / Formula                                                                                                                                                          | Goal                                                |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| CPU Utilization       | Percentage of time the CPU is actually busy doing work (not idle).                                                                                                         | Higher is better (close to 100%)                    |
| Throughput            | Number of processes completed per unit of time.                                                                                                                            | Higher is better (more work done)                   |
| Turnaround Time (TAT) | Total time taken from process submission (arrival) to process completion. Formula: $TAT = \text{Completion Time} - \text{Arrival Time}$                                    | Lower is better                                     |
| Waiting Time (WT)     | Total time a process spends waiting in the ready queue (not running, not doing I/O). Formula: $WT = \text{Turnaround Time} - \text{Burst Time}$                            | Lower is better                                     |
| Response Time (RT)    | Time from submission of a request until the first response is produced (first time it gets CPU). Formula: $RT = \text{Time of first CPU allocation} - \text{Arrival Time}$ | Lower is better (important for interactive systems) |

★ **VVI Hotpoint (Exam Important):** Formulas are EXTREMELY important for numerical questions — memorize these exactly: Turnaround Time (TAT) = Completion Time - Arrival Time Waiting Time (WT) = Turnaround Time - Burst Time (also equals: Waiting Time = Completion Time - Arrival Time - Burst Time) Response Time = Time of first CPU allocation - Arrival Time

### 3.5 Preemptive vs Non-Preemptive Scheduling

| Aspect                  | Non-Preemptive                                                                                                            | Preemptive                                                                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Meaning                 | Once CPU is given to a process, it keeps the CPU until it finishes or goes for I/O — CPU cannot be forcefully taken away. | CPU can be forcefully taken away from a running process (e.g., if a higher priority process arrives, or the time quantum expires), even if the process has not finished. |
| Overhead                | Low (no need to frequently save/restore process state)                                                                    | Higher (context switching happens more often)                                                                                                                            |
| Response for short jobs | A short job may have to wait a long time if a long job is running first                                                   | Short jobs get better response since long jobs can be interrupted                                                                                                        |
| Examples of algorithms  | FCFS, Non-preemptive SJF                                                                                                  | Round Robin, Preemptive SJF (SRTF), Preemptive Priority                                                                                                                  |

## 4. Scheduling Algorithms

Below, each algorithm is explained with its logic AND a fully solved numerical example, since numerical questions on scheduling are very common in exams. Practice re-drawing the Gantt chart by hand for each.

### 4.1 First Come First Serve (FCFS)

Definition: The process that arrives first in the ready queue is given the CPU first — exactly like a queue at a ticket counter. It is Non-Preemptive, so once a process starts running, it runs completely till it finishes.

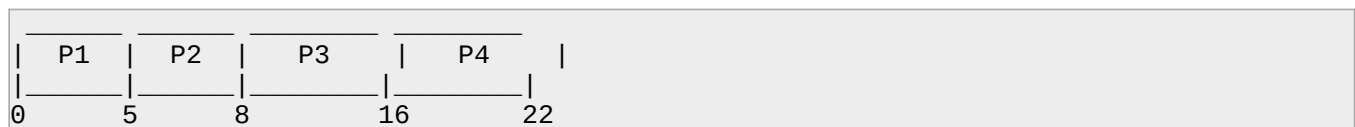
- **Advantage:** Very simple to understand and implement (just a FIFO queue).
- **Disadvantage:** Causes the 'Convoy Effect' — if a long process arrives first, all shorter processes behind it must wait a long time, increasing average waiting time badly.

#### Solved Numerical — FCFS

Given: 4 processes with the following Arrival Time (AT) and Burst Time (BT):

| Process | Arrival Time | Burst Time |
|---------|--------------|------------|
| P1      | 0            | 5          |
| P2      | 1            | 3          |
| P3      | 2            | 8          |
| P4      | 3            | 6          |

Step 1 — Gantt Chart (order of execution = order of arrival, since all arrive in order P1, P2, P3, P4):



Step 2 — Calculate Completion Time, Turnaround Time (TAT), and Waiting Time (WT):

| Process | AT | BT | Completion Time | Turnaround Time | Waiting Time  |
|---------|----|----|-----------------|-----------------|---------------|
| P1      | 0  | 5  | 5               | $5 - 0 = 5$     | $5 - 5 = 0$   |
| P2      | 1  | 3  | 8               | $8 - 1 = 7$     | $7 - 3 = 4$   |
| P3      | 2  | 8  | 16              | $16 - 2 = 14$   | $14 - 8 = 6$  |
| P4      | 3  | 6  | 22              | $22 - 3 = 19$   | $19 - 6 = 13$ |

Step 3 — Calculate Averages:

- Average Turnaround Time =  $(5 + 7 + 14 + 19) / 4 = 45 / 4 = 11.25$  units
- Average Waiting Time =  $(0 + 4 + 6 + 13) / 4 = 23 / 4 = 5.75$  units

🟢 **Sticky Note:** In FCFS, notice P3 has a huge Waiting Time (6) even though it arrived early, just because P1's burst time is not that long but the queue order made it wait. This is a mild version of Convoy Effect — imagine if P1's burst time was 50 instead of 5!

## 4.2 Shortest Job First (SJF)

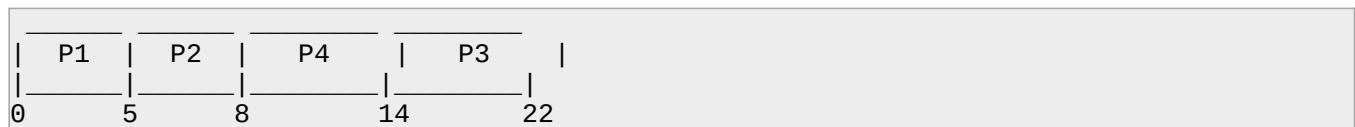
Definition: The process with the smallest burst time (CPU time needed) among the currently available (arrived) processes is executed first. SJF can be Non-Preemptive (once started, runs till completion) or Preemptive (called SRTF — Shortest Remaining Time First — can be interrupted if a new process arrives with even shorter remaining time).

- **Advantage:** Gives the minimum possible average waiting time among all scheduling algorithms (it is provably optimal for average waiting time).
- **Disadvantage:** Can cause 'Starvation' — a long process may never run if shorter processes keep arriving continuously; also, burst time must be known/predicted in advance, which is hard in real systems.

### Solved Numerical — SJF (Non-Preemptive)

Using the SAME processes as the FCFS example above (P1: AT0/BT5, P2: AT1/BT3, P3: AT2/BT8, P4: AT3/BT6):

- At  $t=0$ , only P1 has arrived  $\rightarrow$  run P1 (no choice yet).
- At  $t=5$ , P2(BT3), P3(BT8), P4(BT6) have all arrived  $\rightarrow$  shortest is P2(3)  $\rightarrow$  run P2.
- At  $t=8$ , remaining are P3(BT8), P4(BT6)  $\rightarrow$  shortest is P4(6)  $\rightarrow$  run P4.
- At  $t=14$ , only P3 remains  $\rightarrow$  run P3.



| Process | AT | BT | Completion Time | Turnaround Time | Waiting Time  |
|---------|----|----|-----------------|-----------------|---------------|
| P1      | 0  | 5  | 5               | $5 - 0 = 5$     | $5 - 5 = 0$   |
| P2      | 1  | 3  | 8               | $8 - 1 = 7$     | $7 - 3 = 4$   |
| P4      | 3  | 6  | 14              | $14 - 3 = 11$   | $11 - 6 = 5$  |
| P3      | 2  | 8  | 22              | $22 - 2 = 20$   | $20 - 8 = 12$ |

- Average Turnaround Time =  $(5 + 7 + 11 + 20) / 4 = 43 / 4 = 10.75$  units
- Average Waiting Time =  $(0 + 4 + 5 + 12) / 4 = 21 / 4 = 5.25$  units

 **Sticky Note:** Compare with FCFS: Average Waiting Time dropped from 5.75 to 5.25 just by reordering using SJF logic. This proves SJF reduces average waiting time — a very common exam comparison point.

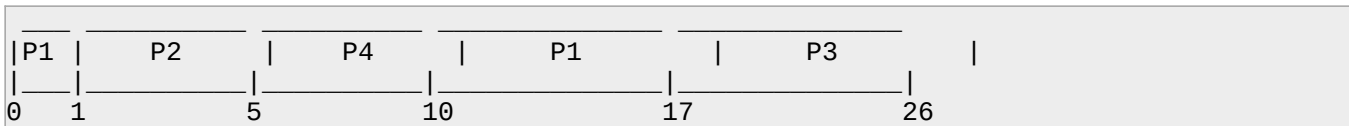
### Solved Numerical — SJF Preemptive (SRTF)

Given:

| Process | Arrival Time | Burst Time |
|---------|--------------|------------|
| P1      | 0            | 8          |
| P2      | 1            | 4          |
| P3      | 2            | 9          |
| P4      | 3            | 5          |

Step-by-step logic: at every new arrival, compare the remaining time of the running process with the new process's burst time, and run whichever is shortest.

- $t=0$ : Only P1 (rem 8)  $\rightarrow$  run P1.
- $t=1$ : P2 arrives (rem 4). P1's remaining = 7.  $P2(4) < P1(7) \rightarrow$  switch to P2.
- $t=2$ : P3 arrives (rem 9). P2's remaining = 3.  $P2(3)$  is still shortest  $\rightarrow$  continue P2.
- $t=3$ : P4 arrives (rem 5). P2's remaining = 2.  $P2(2)$  is still shortest  $\rightarrow$  continue P2.
- $t=4$ : P2 remaining = 1, still shortest  $\rightarrow$  continue P2.
- $t=5$ : P2 finishes (completion = 5). Compare remaining: P1(7), P3(9), P4(5)  $\rightarrow$  P4 is shortest  $\rightarrow$  run P4.
- $t=5$  to  $10$ : No new arrivals, P4 runs fully (5 units)  $\rightarrow$  P4 completes at  $t=10$ .
- $t=10$ : Compare remaining: P1(7), P3(9)  $\rightarrow$  P1 is shortest  $\rightarrow$  run P1 fully (7 units)  $\rightarrow$  P1 completes at  $t=17$ .
- $t=17$ : Only P3 left (rem 9)  $\rightarrow$  run P3  $\rightarrow$  completes at  $t=26$ .



| Process | AT | BT | Completion Time | Turnaround Time | Waiting Time  |
|---------|----|----|-----------------|-----------------|---------------|
| P1      | 0  | 8  | 17              | $17 - 0 = 17$   | $17 - 8 = 9$  |
| P2      | 1  | 4  | 5               | $5 - 1 = 4$     | $4 - 4 = 0$   |
| P3      | 2  | 9  | 26              | $26 - 2 = 24$   | $24 - 9 = 15$ |
| P4      | 3  | 5  | 10              | $10 - 3 = 7$    | $7 - 5 = 2$   |

- Average Turnaround Time =  $(17 + 4 + 24 + 7) / 4 = 52 / 4 = 13$  units
- Average Waiting Time =  $(9 + 0 + 15 + 2) / 4 = 26 / 4 = 6.5$  units

★ **VVI Hotpoint (Exam Important):** In SRTF, ALWAYS re-check remaining time at every new arrival — this is the step students forget. Also note P2 got Waiting Time = 0 because it was never interrupted once it started (it had the shortest job at that moment).

### 4.3 Round Robin (RR)

Definition: Each process is given a fixed small unit of CPU time called a Time Quantum (TQ). Processes are kept in a circular queue. When a process's time quantum expires and it has not finished, it is moved to the back of the queue, and the next process in line gets the CPU. This is a Preemptive algorithm, and it is the basis of all Time-Sharing systems.

- **Advantage:** Every process gets a fair, equal share of CPU time — no starvation, good for interactive/time-sharing systems, gives good response time.
- **Disadvantage:** If the time quantum is too large, RR behaves just like FCFS; if too small, there is too much context switching overhead, wasting CPU time.

#### Solved Numerical — Round Robin (Time Quantum = 4)

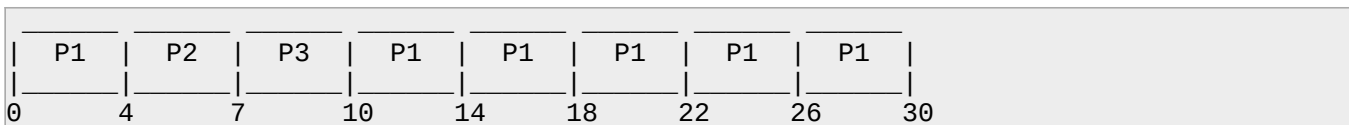
Given (all processes arrive at time 0):

| Process | Arrival Time | Burst Time |
|---------|--------------|------------|
| P1      | 0            | 24         |
| P2      | 0            | 3          |
| P3      | 0            | 3          |

| Process | Arrival Time | Burst Time |
|---------|--------------|------------|
|         |              |            |

Step-by-step simulation (Queue order: P1, P2, P3):

- t=0 to 4: P1 runs for 4 units (rem  $24-4=20$ ). New queue: P2, P3, P1.
- t=4 to 7: P2 runs — only needs 3 units, so it finishes completely at t=7 (P2 completion = 7).
- t=7 to 10: P3 runs — only needs 3 units, finishes completely at t=10 (P3 completion = 10).
- t=10 to 14: P1 runs again for 4 units (rem  $20-4=16$ ).
- t=14 to 18: P1 runs again for 4 units (rem  $16-4=12$ ).
- t=18 to 22: P1 runs again for 4 units (rem  $12-4=8$ ).
- t=22 to 26: P1 runs again for 4 units (rem  $8-4=4$ ).
- t=26 to 30: P1 runs its last 4 units and finishes (P1 completion = 30).



| Process | AT | BT | Completion Time | Turnaround Time | Waiting Time  |
|---------|----|----|-----------------|-----------------|---------------|
| P1      | 0  | 24 | 30              | $30 - 0 = 30$   | $30 - 24 = 6$ |
| P2      | 0  | 3  | 7               | $7 - 0 = 7$     | $7 - 3 = 4$   |
| P3      | 0  | 3  | 10              | $10 - 0 = 10$   | $10 - 3 = 7$  |

- Average Turnaround Time =  $(30 + 7 + 10) / 3 = 47 / 3 \approx 15.67$  units
- Average Waiting Time =  $(6 + 4 + 7) / 3 = 17 / 3 \approx 5.67$  units

**Sticky Note:** Notice how P1 (the long process) got interrupted multiple times, but P2 and P3 (short processes) got to finish quickly without waiting for P1 to fully complete — this fairness is the main selling point of Round Robin, unlike FCFS where a long first process blocks everyone.

#### 4.4 Comparison of FCFS, SJF, and RR

| Aspect               | FCFS                               | SJF / SRTF                                                  | Round Robin                        |
|----------------------|------------------------------------|-------------------------------------------------------------|------------------------------------|
| Preemptive?          | No                                 | Non-Preemptive version: No   Preemptive version (SRTF): Yes | Yes                                |
| Average Waiting Time | Highest (can suffer Convoy Effect) | Lowest (optimal)                                            | Medium (depends on time quantum)   |
| Starvation risk      | No                                 | Yes (long jobs may starve)                                  | No                                 |
| Best suited for      | Simple batch systems               | Systems where burst times are known/predictable             | Interactive / time-sharing systems |
| Fairness             | Low (order-dependent)              | Low (short jobs favored)                                    | High (equal turns for all)         |

**Numerical Practice Space:** Extra practice space — try solving a Priority Scheduling numerical here on your own (Given: processes with Arrival Time, Burst Time, and Priority number; lower number = higher priority). Draw

the Gantt chart, then calculate Completion Time, Turnaround Time, and Waiting Time for each process, just like the solved examples above.

## Quick Revision Summary

| Topic                 | One-Line Recap                                                                                                                                           |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Process               | A program in execution; an active entity that needs CPU, memory, and resources (unlike a program, which is passive).                                     |
| Process States        | New → Ready → Running → Waiting → Terminated — a process moves between these states during its life.                                                     |
| PCB                   | A data structure that stores all information about a process (PID, state, PC, registers, memory info, accounting, I/O info) so it can be paused/resumed. |
| Context Switching     | Saving the state of the current process and loading the state of the next process, so CPU can switch between processes; considered pure overhead.        |
| Thread                | The smallest unit of execution inside a process; called a 'lightweight process' because it shares memory with other threads of the same process.         |
| Types of Threads      | User-Level Threads (managed by library, fast but block entire process) and Kernel-Level Threads (managed by OS, slower but more reliable).               |
| Multithreading Models | Many-to-One, One-to-One, and Many-to-Many — different ways to map user threads to kernel threads.                                                        |
| Schedulers            | Long-Term (which jobs enter memory), Short-Term (which ready process gets CPU next), Medium-Term (swap processes in/out of memory).                      |
| Scheduling Criteria   | CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time — used to judge how good a scheduling algorithm is.                            |
| FCFS                  | First arrived, first served; simple but suffers from Convoy Effect (long process blocks short ones).                                                     |
| SJF / SRTF            | Shortest job/remaining time runs first; gives lowest average waiting time but can cause starvation of long jobs.                                         |
| Round Robin           | Each process gets a fixed time quantum in circular order; fair and good for time-sharing, but performance depends on quantum size.                       |

## 15 Exam-Style Questions — Unit 2

- Q1.** Define a Process. Differentiate between a Process and a Program.
- Q2.** Draw and explain the Process State transition diagram in detail.
- Q3.** What is a Process Control Block (PCB)? Explain its contents with a diagram.
- Q4.** What is Context Switching? Why is it considered an overhead for the system?
- Q5.** Define a Thread. Why is a thread called a 'lightweight process'?
- Q6.** Explain the benefits of using threads in an application.
- Q7.** Differentiate between User-Level Threads and Kernel-Level Threads.
- Q8.** Explain the Many-to-One, One-to-One, and Many-to-Many multithreading models with diagrams.
- Q9.** Differentiate between a Process and a Thread.
- Q10.** Explain the different types of schedulers (Long-Term, Short-Term, Medium-Term) with their functions.
- Q11.** Explain the various scheduling criteria used to evaluate a CPU scheduling algorithm, along with their formulas.
- Q12.** Differentiate between Preemptive and Non-Preemptive scheduling with examples.
- Q13.** What is FCFS scheduling? Explain the Convoy Effect problem with an example.
- Q14.** For the given set of processes with arrival time and burst time, draw the Gantt chart and calculate average Waiting Time and Turnaround Time using (a) FCFS (b) SJF Non-Preemptive (c) Round Robin (time quantum = 3).  
[Numerical — use your own process set for practice, following the solved examples in this unit]
- Q15.** Explain Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) scheduling with a suitable numerical example.