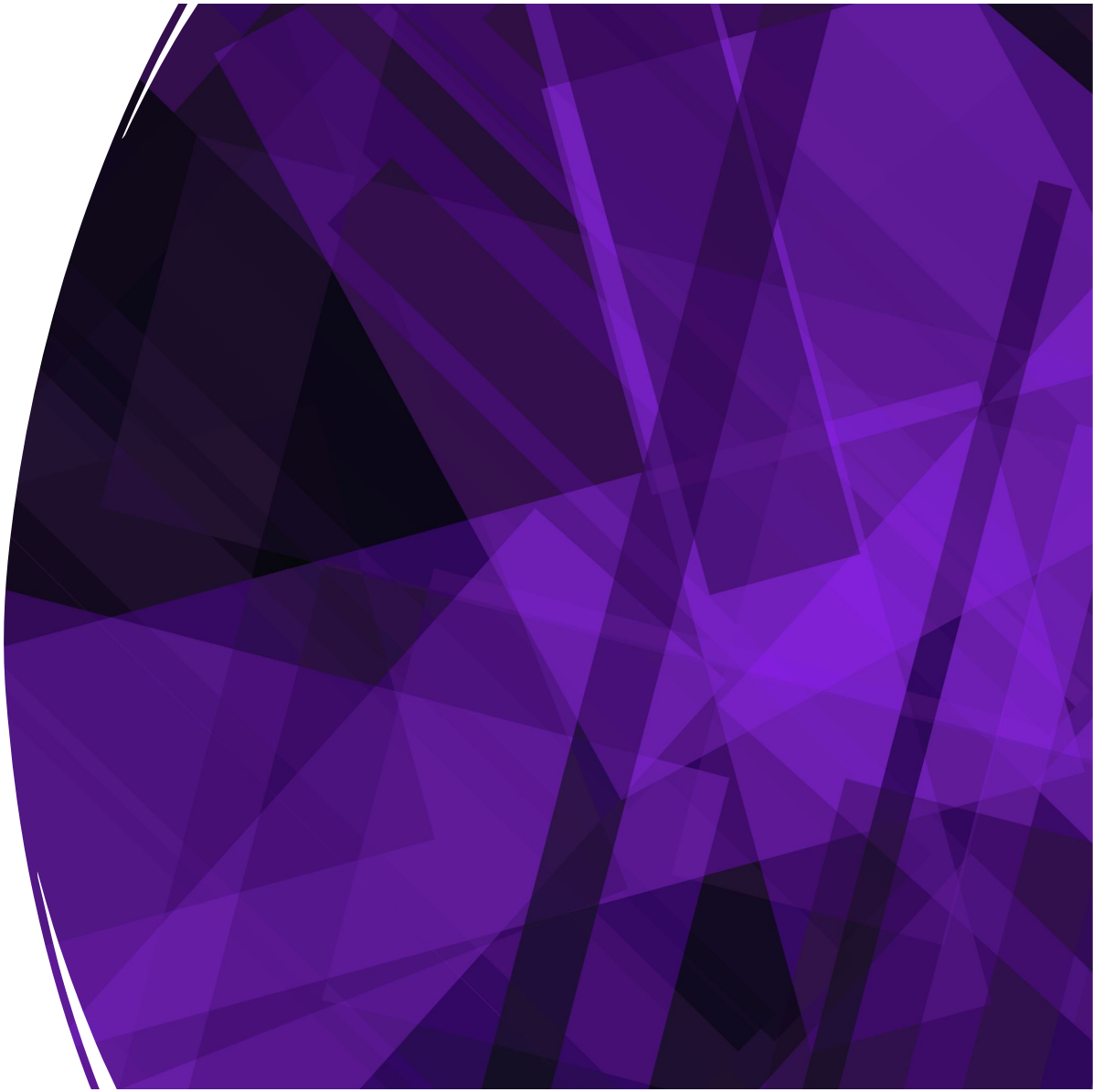
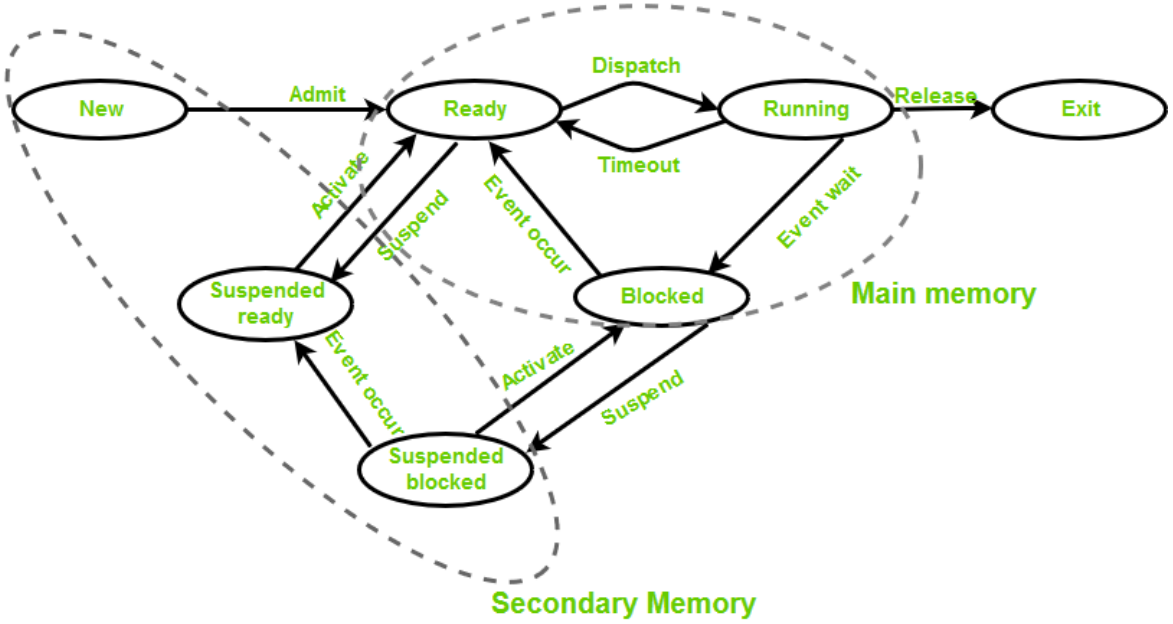
CPU Scheduling



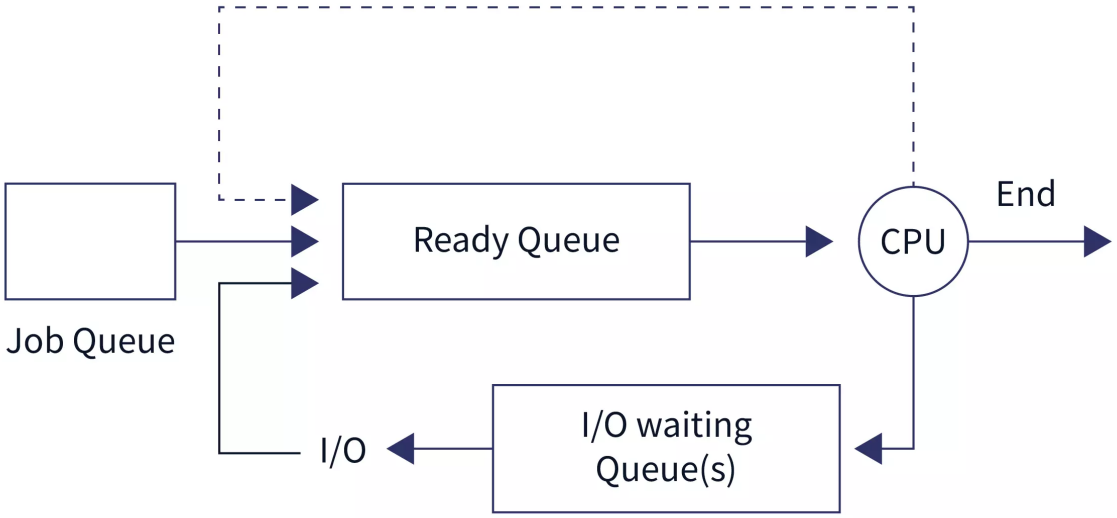
Process Execution Cycle

- Process execution consists of a cycle of CPU execution and I/O wait.



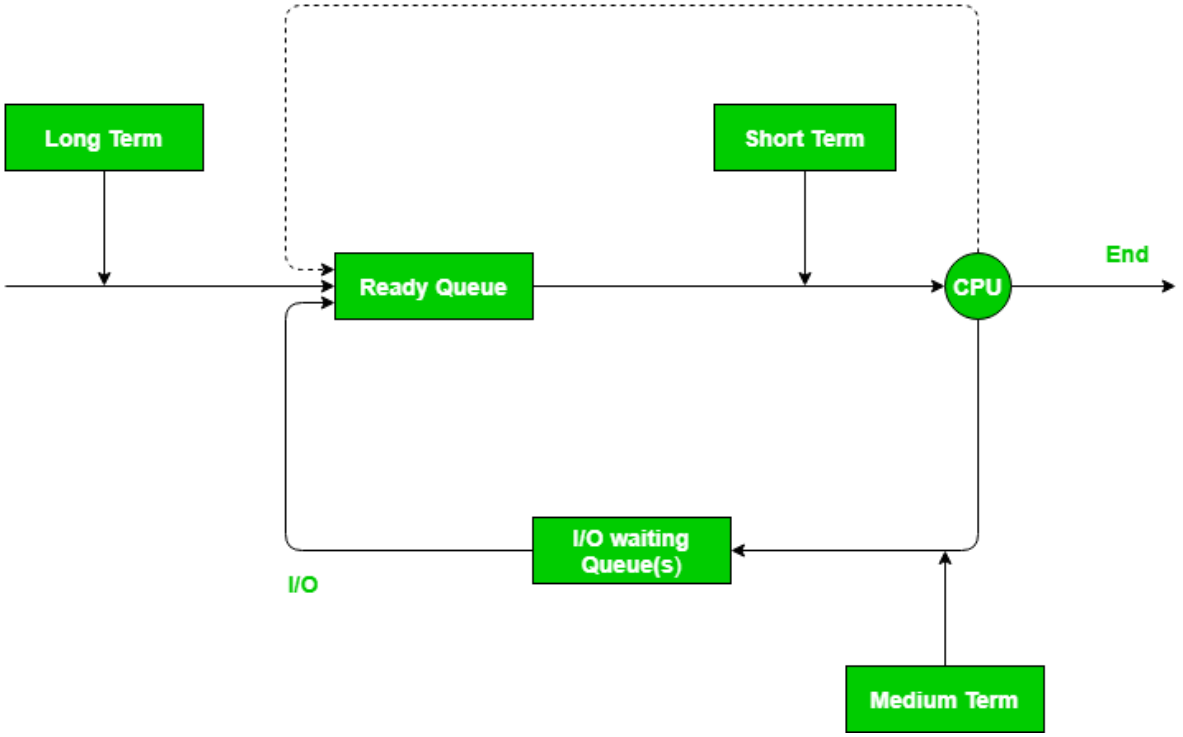
CPU Scheduling Decisions

- Scheduling decisions occur when a process:
  - Switches from running to waiting (nonpreemptive)
  - Switches from running to ready (preemptive)
  - Switches from waiting to ready (preemptive)
  - Terminates (nonpreemptive)



Dispatcher Module

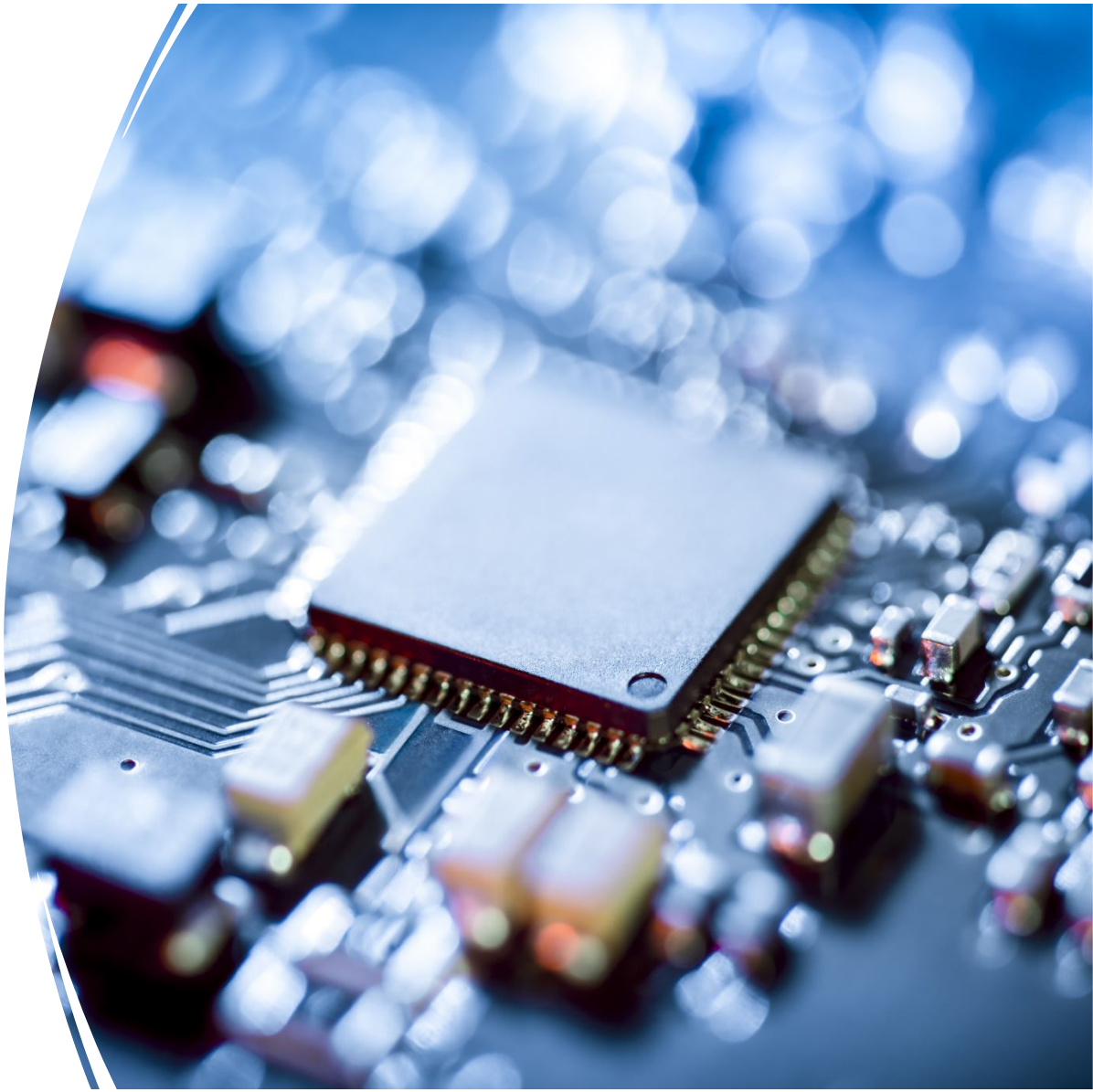
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler.
- **Dispatch latency**: the time it takes for the dispatcher to stop one process and start another.



CPU Scheduling Algorithms

Many CPU scheduling algorithms can be classified into two categories:

- Non-preemptive
- Preemptive.



Scheduling Algorithms (Continuation)

- Non-Preemptive Scheduling

Once a process starts its execution, it runs to completion without being interrupted by other processes.

Examples:

➢First-Come, First-Served (FCFS): Processes are executed in the order they arrive.

➢Shortest Job First (SJF): The process with the smallest burst time is executed next.

➢Priority Scheduling (Non-Preemptive): The highest priority process that is ready to run will execute until completion.

Scheduling Algorithms (Continuation)

- Preemptive Scheduling

A running process can be interrupted (preempted) if a new process arrives with a higher priority or shorter remaining time.

Examples:

➢Round Robin (RR): Each process is assigned a fixed time slice and can be preempted after its time slice expires.

➢Shortest Remaining Time First (SRTF): A process is preempted if a new process arrives with a shorter remaining burst time.

➢Priority Scheduling (Preemptive): A higher priority process can interrupt a currently running lower priority process.

Choosing between Preemptive and Non-Preemptive

The choice of whether to use a preemptive or non-preemptive algorithm often depends on the specific requirements of the system, such as response time for interactive processes or throughput for batch processes.

Real-World Systems

Most modern operating systems use preemptive scheduling to ensure responsiveness, particularly in multi-tasking environments where multiple processes need to share CPU time effectively.

Preemptive Scheduling Systems

- Windows OS: Uses preemptive scheduling to allow high-priority tasks to interrupt lower-priority ones for a responsive user experience.
- Linux OS: Employs a Completely Fair Scheduler (CFS) that allows tasks to be preempted, ensuring fairness and responsiveness.
- Mobile OS (Android/iOS): Both use preemptive scheduling to manage multiple applications, ensuring quick response to user interactions.

Non-Preemptive Scheduling Systems

- Embedded Systems: Simple microcontrollers often use non-preemptive scheduling to execute tasks sequentially without interruptions.
- Batch Processing Systems: Mainframe systems process jobs in the order they arrive, using non-preemptive scheduling to utilize resources efficiently.
- Hard Real-Time Systems: Certain industrial control systems use non-preemptive scheduling to ensure critical tasks are completed in a strict order without interruption.

Scheduling Algorithms (Continuation)

- Scheduling algorithms are chosen based on optimization criteria (e.g., throughput, turnaround time, etc.):
  - FCFS (First-Come, First-Served)
  - SJF (Shortest Job First)
  - Round Robin
  - Priority

First Come First Serve (FCFS)

- **Definition**: Allocates CPU to the process that requests it first, using a FIFO queue.
- **Characteristics:**
- Executes tasks on a First-come, First-serve basis
- Easy to implement
- High wait time; less efficient in performance

Example fcfs

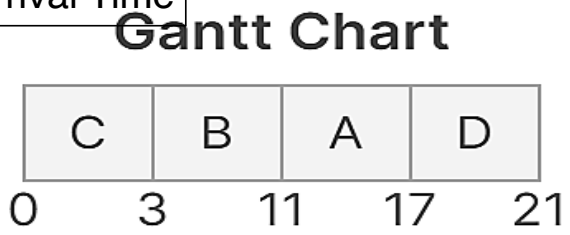
Let's consider an example where four processes with different burst times arrive at different times. Find the finish time, turnaround time, and waiting time for each process. Also, draw a Gantt chart.

	Process	Burst Time	Arrival time
A	P1	6	2
B	P2	8	1
C	P3	3	0
D	P4	4	4

Answer

Waiting Time = Turnaround Time – Burst Time

Turnaround Time = Finish time – Arrival Time



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
C	0	3	3	3 – 0 = 3	3 – 3 = 0
B	1	8	11	11 – 1 = 10	10 – 8 = 2
A	2	6	17	17 – 2 = 15	15 – 6 = 9
D	4	4	21	21 – 4 = 17	17 – 4 = 13

Explanation

- P3 arrives first at time 0 and starts executing immediately. It has a burst time of 3.
- P2 arrives at time 1 but must wait until P3 finishes. Since FCFS does not allow preemption, P2 will have to wait until P3 completes its execution.
- P1 arrives at time 2 and also has to wait for P3 and then P2.
- P4 arrives at time 4 and waits for P3,P2, and P1 to finish before it can start.

Shortest Job First (SJF)

- **Definition**: Selects the waiting process with the smallest execution time to execute next.
- **Characteristics:**
- Minimizes average waiting time among all scheduling algorithms
- Each task is associated with a unit of time to complete
- May cause starvation if shorter processes keep arriving (solvable with aging)

Example Non-Preemptive SJF

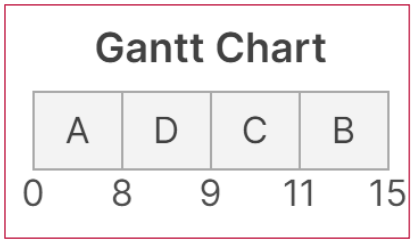
Let's consider an example where four processes with different burst times arrive at different times. Find the finish time, turnaround time, and waiting time for each process. Also, draw a Gantt chart.

	Process	Arrival Time	Burst Time
A	P1	0	8
B	P2	1	4
C	P3	2	2
D	P4	3	1

Waiting Time = Turnaround Time – Burst Time

Turnaround Time = Finish time – Arrival Time

Answer



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	8	8	8	0
B	1	4	15	14	10
C	2	2	11	9	7
D	3	1	9	6	5

Explanation

- **At time 0**: **P1** is the only process that has arrived, so it starts execution.
- **At time 8**: **P1** completes. The remaining processes (P2, P3, P4) have arrived.
- Among P2 (4), P3 (2), and P4 (1), **P4** has the shortest burst time. It starts execution.
- **At time 9**: **P4** completes. The remaining processes are P2 and P3.
- **P3** has the next shortest burst time (2), so it starts execution next.
- **At time 11**: **P3** completes. Finally, **P2** starts execution.
- **At time 15**: **P2** completes.

Preemptive Priority Scheduling

•**Definition**: Preemptive method based on process priority.
•**Mechanism**: Higher priority processes preempt lower priority ones. If priorities are equal, FCFS is used.
Characteristics:
•Schedules tasks based on priority.
•Higher priority process preempts lower priority one.
•Lower priority process is suspended until the higher priority task completes.
•Lower number = higher priority level.

who gives the priority in a priority scheduling algorithm?

Defined by the Problem or System:

- **Gi**ven in the Problem Statement: In many scheduling problems, such as the one we're discussing, the priority of each process is provided as part of the problem's input data. This means the priority levels are predefined and given to you to solve the scheduling problem.
- **Assigned by the Operating System**: In real-world scenarios, priorities can be assigned by the operating system based on various criteria. These criteria might include the importance of the task, user-defined settings, or system policies.

How Priorities Work?

- Priority Number: Each process is assigned a priority number. In most systems, a lower number indicates a higher priority.
- Scheduling Decision: The scheduler (a component of the operating system) uses these priority numbers to decide which process to run. The process with the highest priority (lowest number) is selected to run first.

Example Preemptive Priority

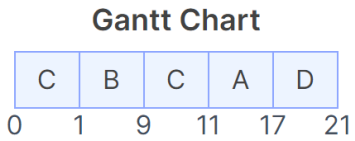
Let's consider an example where four processes with different burst times arrive at different times. Find the finish time, turnaround time, and waiting time for each process. Also, draw a Gantt chart.

	Process	Burst Time	Arrival time	Prio
A	P1	6	2	3
B	P2	8	1	1
C	P3	3	0	2
D	P4	4	4	4

Waiting Time = Turnaround Time – Burst Time

Turnaround Time = Finish time – Arrival Time

Answer



ID	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
C	0	3	11	11	8
B	1	8	9	8	0
A	2	6	17	15	9
D	4	4	21	17	13

Explanation

- **P3** starts at time 0 because it's the only process.
- **P2** arrives at time 1 and preempts P3 due to higher priority.
- **P2** continues running until it finishes at time 9.
- **P3** resumes from where it left off and runs from time 9 to 12.
- **P1** then starts at time 12 and runs until it finishes at time 18.
- **P4** finally runs from time 18 to 22.

Traditional UNIX Scheduling

- Provides a clear illustration of how process priorities can be adjusted based on CPU usage.
- Formula:

$$\text{Priority} = \left(\frac{\text{recent CPU usage}}{2}\right) + \text{base}$$

- The scheduler aims to prevent any single process from monopolizing the CPU, thereby ensuring fairer distribution of CPU time among all processes.
- Modern systems employ advanced algorithms for fair and efficient scheduling.

Round Robin Scheduling

- **Definition**: Each process is cyclically assigned a fixed time slot. Preemptive version of FCFS.
- **Focus**: Time Sharing technique.
- **Characteristics:**
- Simple, easy to use, and starvation-free.
- Widely used in CPU scheduling.
- Preemptive: Processes are given limited CPU time.

Example Round Robin

Let's consider an example where four processes with different burst times arrive at different times. Find the finish time, turnaround time, and waiting time for each process. Also, draw a Gantt chart. Time Quantum = 4

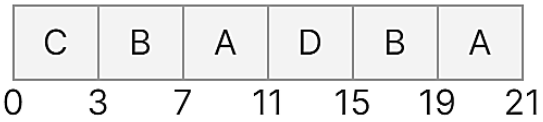
	Process	Burst Time	Arrival time
A	P1	6	2
B	P2	8	1
C	P3	3	0
D	P4	4	4

Waiting Time = Turnaround Time – Burst Time

Turnaround Time = Finish time – Arrival Time

Answer

Gantt Chart



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
C	0	3	3	3 – 0 = 3	3 – 3 = 0
B	1	8	19	19 – 1 = 18	18 – 8 = 10
A	2	6	21	21 – 2 = 19	19 – 6 = 13
D	4	4	15	15 – 4 = 11	11 – 4 = 7

Explanation

- **At time 0**: P3 starts executing.
- Remaining Burst Time for P3 = 3 (finishes in the first time slice).
- **C** runs from 0 to 3.
- **At time 3**: P2 arrives (burst time = 8). P3 finishes, and now it's P2's turn.
- **P2** runs from 3 to 7 (4 time units used).
- Remaining Burst Time for P2 = 4.
- **At time 7**: P1 arrives (burst time = 6). Now it's P1's turn.
- **P1** runs from 7 to 11 (4 time units used).
- Remaining Burst Time for P1 = 2.
- **At time 11**: P4 arrives (burst time = 4).
- **P4** runs from 11 to 15 (finishing now).
- Remaining Burst Time for P4 = 0.
- **At time 15**: P2 is next.
- **P2** runs from 15 to 19 (finishing now).
- Remaining Burst Time for P2 = 0.
- **At time 19**: P1 is next.
- **P1** runs from 19 to 21 (finishing now).
- Remaining Burst Time for P1 = 0.

Multilevel Queues

•**Structure**: Fixed multiple queues based on criteria (e.g., priority, process type).
•**Assignment**: Processes assigned to a queue remain there.

Scheduling:
•**Fixed Priority**: Higher priority queues are always chosen first.
•**Time Slicing**: Queues scheduled in a round-robin fashion.

Advantages: Simple and clear separation of processes.
Disadvantages: Inflexible, may cause starvation of lower-priority processes.

Multilevel Feedback Queues (MLFQ)

- **Structure**: Multiple dynamic queues with feedback.
- **Assignment**: Processes can move between queues based on behavior.

Scheduling:
- **Feedback**: Processes using too much CPU time move to lower-priority queues; long-waiting processes move to higher-priority queues.
- **Aging**: Prevents starvation by increasing priority of long-waiting processes.

Advantages: Flexible, dynamic, fair CPU time allocation.
Disadvantages: Complex to implement and manage.

Multiprocessor Scheduling

Overview:

- Manages processes across multiple CPUs/cores for optimal performance.

Types:

1.Asymmetric Multiprocessing (AMP):

1. **One master processor** handles all system tasks.
2. **Pros**: Simpler, reduced complexity.
3. **Cons**: Bottleneck risk, underutilization of other processors.

2.Symmetric Multiprocessing (SMP):

1. **All processors** share memory and I/O, each with its own scheduler.
2. **Pros**: Balanced load, better performance.
3. **Cons**: More complex, potential resource contention.

Little's Formula

- Helps determine average wait time per process in any scheduling algorithm:
  - $n=\lambda \times W$
  - n = avg queue length; W = avg waiting time in queue; λ = average arrival rate into queue.
- Essential for understanding and optimizing queuing systems across various applications.
- Applications: Call Centers, Computer Networks, and Manufacturing

Example

- **Scenario:** A bank teller system where customers arrive at an average rate of 10 per hour ($\lambda = 10$ customers/hour) and each customer spends an average of 6 minutes (0.1 hours) in the system ($W = 0.1$ hours).
- **Calculation**: $L = \lambda \cdot W = 10 \cdot 0.1 = 1$
- **Interpretation**: On average, there is 1 customer in the system.

Simulations

- **Definition**: Programmed models of a computer system with variable clocks, mimicking real-world processes.
- **Purpose**: Gather statistics indicating algorithm performance.
- **Advantages**: **High Accuracy**: More accurate than queuing models (e.g., Little's Law).
- **Disadvantages**: **High Cost & Risk**: Resource-intensive and complex to manage.

Comparison to Queuing Models

- **Little's Law**: Quick, approximate analysis; easier and cheaper, but less detailed. Queuing Models are suited for simpler, stable environments
- **Simulations**: Detailed insights and high accuracy; require significant resources. Simulations are ideal for complex, dynamic scenarios.

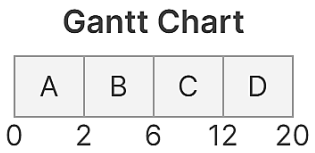
Activity

Let's consider an example where four processes with different burst times arrive at different times. Find the finish time, turnaround time, and waiting time for each process. Also, draw a Gantt chart and give me the average turnaround time and average waiting time. Please do it for FCFS and RR. Time Quantum = 3.

Process	Arrival Time	Burst Time
P1	0	2
P2	2	4
P3	4	6
P4	6	8

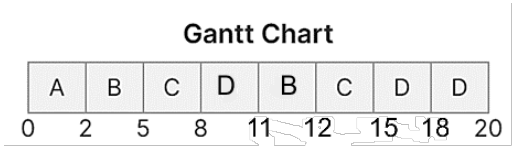
Answer

FCFS



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	2	2	2	0
B	2	4	6	4	0
C	4	6	12	8	2
D	6	8	20	14	6
Average				28 / 4 = 7	8 / 4 = 2

RR



Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	2	2	2	0
B	2	4	12	10	6
C	4	6	15	11	5
D	6	8	20	14	6
Average				9.25	4.25