Process Scheduling

Dr Tran Duc Minh and Dr Hung Tran

DATCOM Lab
Faculty of Data Science and Artificial Intelligence
College of Technology, National Economics University
Email: minhdt@neu.edu.vn, hung.tran@neu.edu.vn

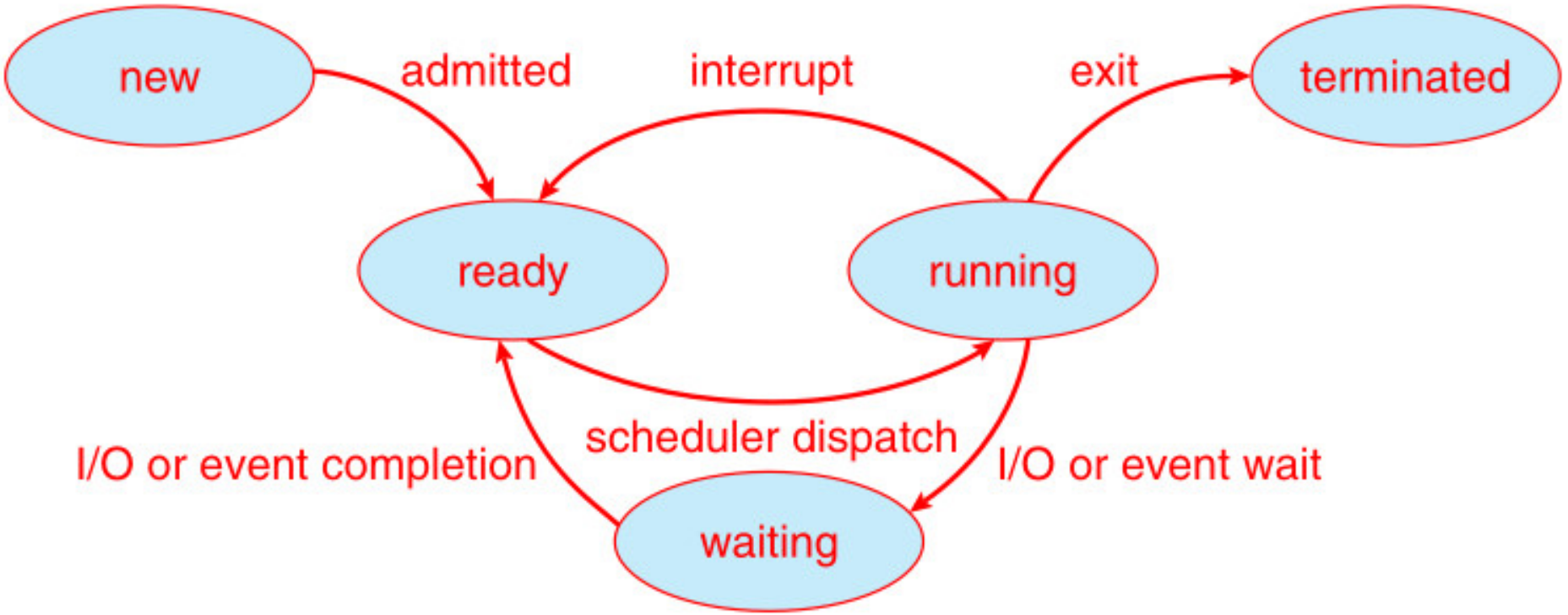
March 31, 2025

Outline

Process State

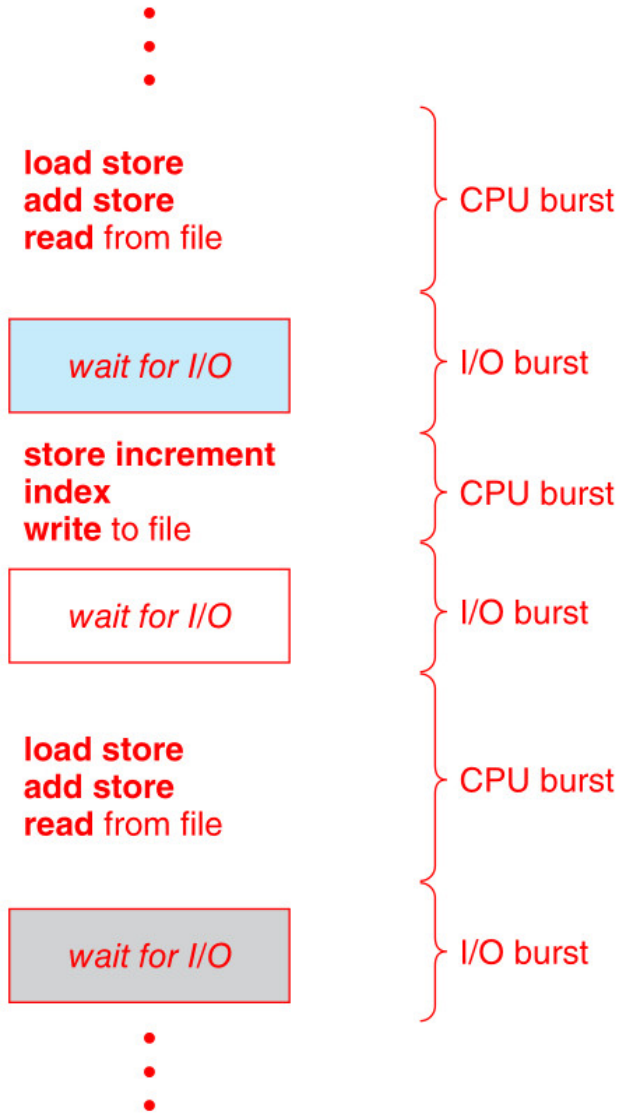
- As a process executes, it changes state
  - **New**: The process is being created.
  - **Running**: Instructions are being executed.
  - **Waiting**: The process is waiting for some event to occur.
  - **Ready**: The process is waiting to be assigned to a processor.
  - **Terminated**: The process has finished execution

Process State



CPU - I/O Burst Cycle

- Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate between these two states.
- Process execution begins with a **CPU burst**. That is followed by an **I/O burst**, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution



CPU Scheduler

- Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed.
  - The selection process is carried out by the **CPU scheduler**, which selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
  - A ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.
  - The records in the queues are generally process control blocks (PCBs) of the processes.

Preemptive and Nonpreemptive Scheduling

- **Preemptive Scheduling**
  - Preemptive Scheduling is a CPU scheduling method where the operating system can interrupt (preempt) a currently running process to allocate the CPU to another process with a higher priority or based on a specific scheduling criterion.
  - Preemption typically occurs when:
    - ★ A new process arrives with a higher priority.
    - ★ A fixed time interval (time quantum) expires, as in algorithms like Round Robin.
  - The running process can be interrupted at any time if another process meets the scheduling criteria (e.g., higher priority, shorter remaining CPU Burst).
  - Requires context switching: The operating system must save the state of the preempted process and restore the state of the new process.

Preemptive and Nonpreemptive Scheduling

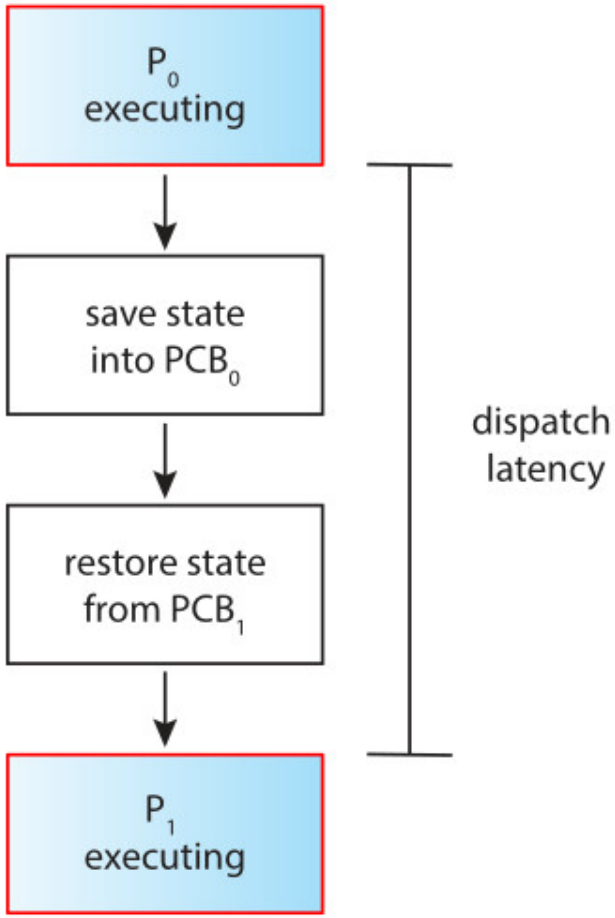
- **Non-preemptive Scheduling**
  - Non-preemptive Scheduling is a CPU scheduling method where, once a process is allocated the CPU, it runs to completion of its CPU Burst or voluntarily yields the CPU (e.g., when it enters an I/O wait state).
  - The operating system cannot intervene to interrupt the running process unless the process completes or yields the CPU.
  - The running process is not interrupted by other processes, even if a higher-priority process arrives.
  - Requires fewer context switches compared to preemptive scheduling, as switching only occurs when a process completes or yields.

Dispatcher

- The **dispatcher** is the module that gives control of the CPU's core to the process selected by the CPU scheduler. This function involves the following:
  - Switching context from one process to another.
  - Switching to user mode.
  - Jumping to the proper location in the user program to resume that program.
- How often do context switches occur?
  - The number of context switches can be obtained by using the **vmstat** command that is available on Linux systems.
  - For example: **vmstat 1 3**
    - ★ Show system-wide statistics, including the number of context switches per second.

Dispatcher

- **Dispatch latency**: Time it takes for the dispatcher to stop one process and start another running.



Scheduling Criteria

- **CPU utilization**: Keep the CPU as busy as possible.
  - Max CPU utilization
- **Throughput**: Number of processes that complete their execution per time unit.
  - Max throughput
- **Turnaround time**: the sum of the periods spent waiting in the ready queue, executing on the CPU, and doing I/O
  - Min turnaround time
- **Waiting time**: Amount of time a process has been waiting in the ready queue.
  - Min waiting time
- **Response time**: Amount of time it takes from when a request was submitted until the first response is produced.
  - Min response time

First-Come, First-Served Scheduling

- **FCFS (First-Come, First-Served)** is a CPU scheduling algorithm that operates on the principle of "first-come, first-served." This means that processes are executed in the order they arrive in the ready queue.
- FCFS is a non-preemptive algorithm.
- **Advantages**
  - Simple and easy to implement.
  - Fair in the sense that processes are handled in the order of arrival.
- **Disadvantages**
  - **Convoy Effect**: If a process with a long CPU Burst arrives first, subsequent processes must wait for a long time, leading to a high average waiting time.
  - Not optimal for systems requiring fast response times, as it lacks a priority mechanism.

First-Come, First-Served Scheduling

- How It Works
  1. Processes are placed in the ready queue in the order of their arrival.
  2. The first process in the queue is assigned the CPU and executes until its CPU Burst is complete.
  3. Once the first process finishes, the next process in the queue is executed, and this continues sequentially.

First-Come, First-Served Scheduling

- For example: Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

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:

The average waiting time is $(0 + 24 + 27) / 3 = 17$ milliseconds.

First-Come, First-Served Scheduling

- For example: Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

  - Suppose that the processes arrive in the order: P_2, P_3, P_1. The Gantt Chart for the schedule is:

The average waiting time is $(6 + 0 + 3) / 3 = 3$ milliseconds.

Shortest-Job-First Scheduling

- **SJF (Shortest-Job-First)** is a CPU scheduling algorithm that prioritizes the process with the shortest CPU Burst (shortest job) at the time the CPU becomes available. In other words, when the CPU is free, the operating system selects the process with the smallest CPU Burst in the ready queue for execution.
- SJF is a non-preemptive algorithm.
- If two processes have the same CPU Burst time, the algorithm uses FCFS to break the tie.
- **Advantages**
  - Optimizes average waiting time: Since shorter processes are executed first, other processes do not have to wait as long, resulting in a lower average waiting time.
  - Efficient in systems where the CPU Burst times of processes can be predicted in advance.

Shortest-Job-First Scheduling

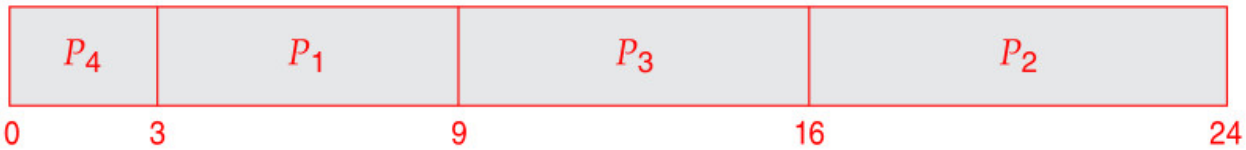
- **Disadvantages**
  - Requires prior knowledge of the CPU Burst times of processes, which is not always feasible in practice.
  - Can lead to starvation: If short processes keep arriving, processes with longer CPU Bursts may have to wait indefinitely.
  - Not suitable for systems requiring fast response times, as it does not prioritize based on arrival time.
- How It Works
  1. Processes are placed in the ready queue.
  2. When the CPU becomes available, the operating system selects the process with the shortest CPU Burst in the queue for execution.
  3. The process runs to completion, and then the operating system selects the next process with the shortest CPU Burst.

Shortest-Job-First Scheduling

- For example: Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

  - Using SJF scheduling, we would schedule these processes according to the following Gantt chart:



The average waiting time is $(3 + 16 + 9 + 0)/4 = 7$ milliseconds.

Shortest-Job-First Scheduling

- Although the SJF algorithm is optimal for minimizing the average waiting time, it faces a practical challenge in CPU scheduling because there is no way to know the exact length of the next CPU Burst of a process in advance.
- In practice, the CPU Burst time is only known after the process has completed its execution.
- **Predicting CPU Burst Time**
  - A **prediction method** is used to estimate the CPU Burst time based on the history of previous CPU Bursts.
  - The core idea is that the next CPU Burst of a process is often similar to its previous CPU Bursts. Therefore, we can predict the next CPU Burst by calculating an **exponential average** of the previous CPU Bursts.

Determining Length of Next CPU Burst

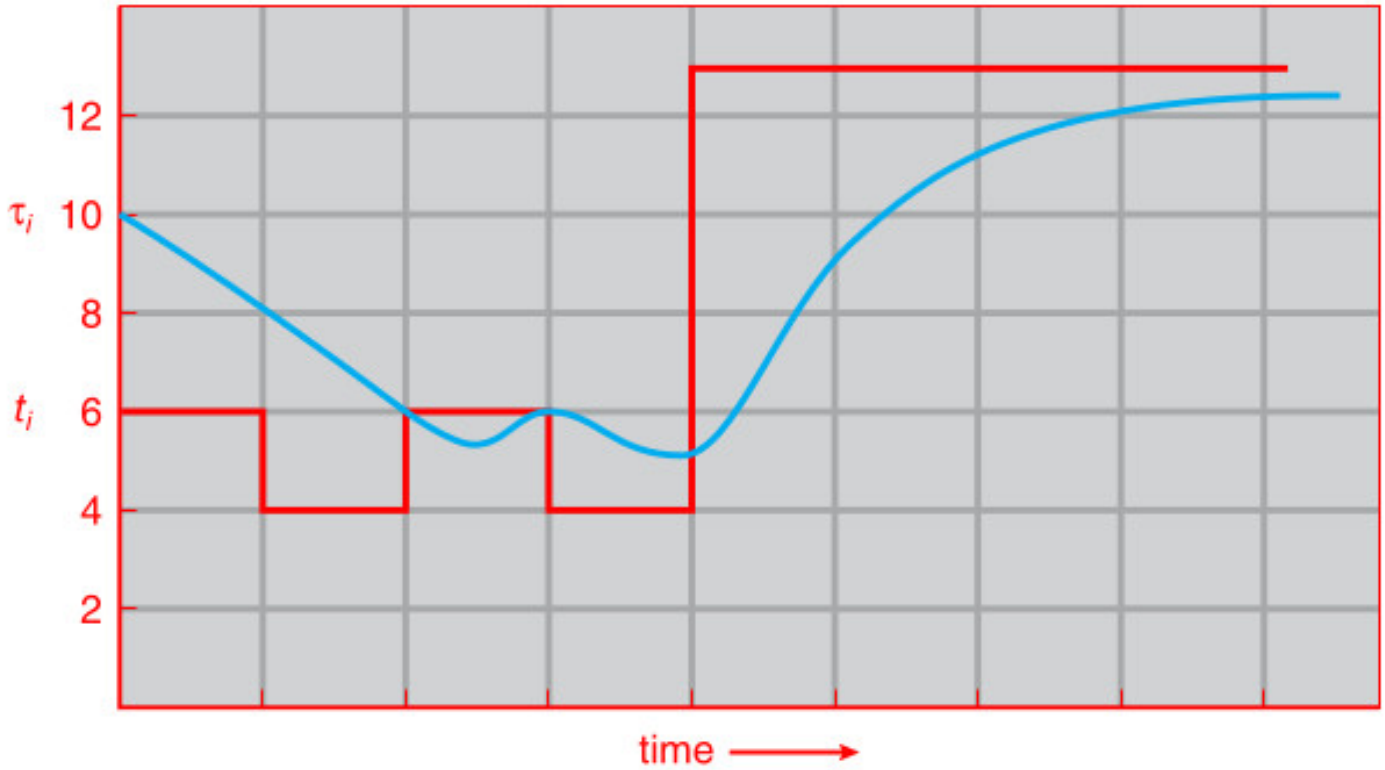
- Let t_n be the actual CPU Burst time of the n-th execution of a process.
- Let τ_{n+1} be the predicted CPU Burst time for the (n+1)-th execution.
- The exponential average formula is:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

  where
  - α is a weighting factor, with $0 \leq \alpha \leq 1$, which determines the influence of the most recent value versus the historical prediction.
  - τ_n: The predicted CPU Burst time for the current execution (calculated from the previous prediction).
- α controls the relative weight of recent and past history in our prediction; The initial τ_0 can be defined as a constant or as an overall system average.

Prediction of the Length of the Next CPU Burst

- With $\alpha = 1/2$ and $\tau_0 = 10$



CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Expanded Formula

- When we substitute τ_n in the formula $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$, we get:
$$\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1}$$
- Substituting τ_n into the formula for τ_{n+1}, we obtain:
$$\tau_{n+1} = \alpha t_n + (1-\alpha)(\alpha t_{n-1} + (1-\alpha)\tau_{n-1})$$
- Continuing to expand τ_{n-1}, τ_{n-2}, ..., down to τ_0, we arrive at the general formula:
$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^{n+1}\tau_0$$

This formula shows that τ_{n+1} is a linear combination of all actual CPU Burst times from t_0 to t_n, plus the initial value τ_0, with weights that decrease over time.

Expanded Formula: Example

- We will calculate τ_4 with $\alpha = 1/2$ and $\tau_0 = 10$

$$\tau_4 = \alpha t_3 + (1-\alpha)\alpha t_2 + (1-\alpha)^2 \alpha t_1 + (1-\alpha)^3 \alpha t_0 + (1-\alpha)^3 \tau_0$$

$$\tau_4 = (0.5x4) + (1-0.5)x0.5x6 + (1-0.5)^2 x0.5x4 + (1-0.5)^3 x0.5x6 + (1-0.5)^4 x10$$

$$\tau_4 = (0.5x4) + (0.25x6) + (0.125x4) + (0.0625x6) + (0.0625x10)$$

$$\tau_4 = 5$$

Result: The predicted $\tau_4 = 5$, which matches the result from the basic formula.

Shortest-Remaining-Time-First Scheduling

- **SRTF (Shortest-Remaining-Time-First)**: If a new process arrives in the ready queue and has a remaining time shorter than the remaining time of the currently running process, the current process is preempted, and the new process is executed.
- SRTF is a preemptive algorithm.
- **Advantages**
  - Reduces average waiting time: By always prioritizing the process with the shortest remaining time, SRTF typically achieves a lower average waiting time compared to non-preemptive SJF.
  - Theoretically optimal: SRTF is the most optimal algorithm for minimizing average waiting time if the CPU Burst times are known in advance.

Shortest-Remaining-Time-First Scheduling

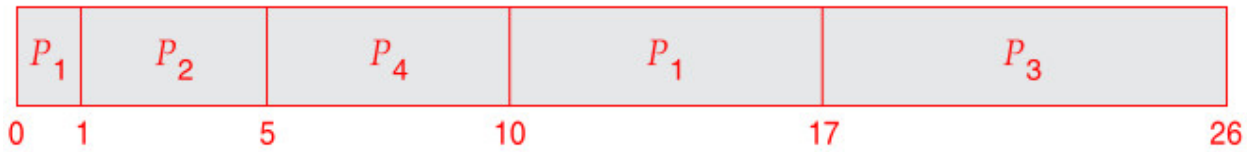
- **Disadvantages**
  - Requires prior knowledge of CPU Burst times, which is not always feasible.
  - Increases system overhead: Preemption requires saving and restoring the state of processes, incurring context switch overhead.
  - Can cause starvation: Processes with long CPU Bursts may be delayed indefinitely if short processes keep arriving.
- How It Works
  1. At each time step, the operating system checks all processes in the ready queue.
  2. The process with the shortest remaining time is selected for execution.
  3. If a new process arrives with a CPU Burst shorter than the remaining time of the currently running process, the current process is preempted, and the new process is executed.

Shortest-Remaining-Time-First Scheduling

- For example: Consider the following four processes, with the length of the CPU burst given in milliseconds:

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

  - The resulting preemptive SJF schedule is as depicted in the following Gantt chart:



The average waiting time for this example is [(10 - 1) + (1 - 1) + (17 - 2) + (5 - 3)]/4 = 26/4 = 6.5 milliseconds.
Nonpreemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

Round Robin Scheduling

- **Round Robin** (RR) scheduling algorithm designed to ensure fairness among processes. Each process is assigned the CPU for a fixed time interval, known as the time quantum.
- RR is a preemptive algorithm.
- Processes are placed in a circular queue. The operating system allocates the CPU to each process in the queue in a round-robin fashion, allowing each process to run for up to one time quantum.
- Two scenarios can occur:
  - If the CPU Burst of a process is less than the time quantum, the process completes and voluntarily releases the CPU.
  - If the CPU Burst is longer than the time quantum, the process is preempted after the time quantum expires and is placed at the end of the queue to wait for its next turn.

Round Robin Scheduling

- **Advantages**
  - Fairness: Each process gets a fair share of CPU time, preventing starvation.
  - Suitable for time-sharing systems, as it ensures quick response times for all processes.
  - Easy to implement and manage.
- **Disadvantages**
  - Context switch overhead: If the time quantum is too small, frequent context switches between processes can lead to significant overhead.
  - High average waiting time: If the time quantum is not chosen appropriately, the average waiting time may be higher compared to algorithms like SJF or SRTF.
  - Performance depends on the time quantum: If the time quantum is too large, RR behaves like FCFS; if too small, context switch overhead increases.

Round Robin Scheduling

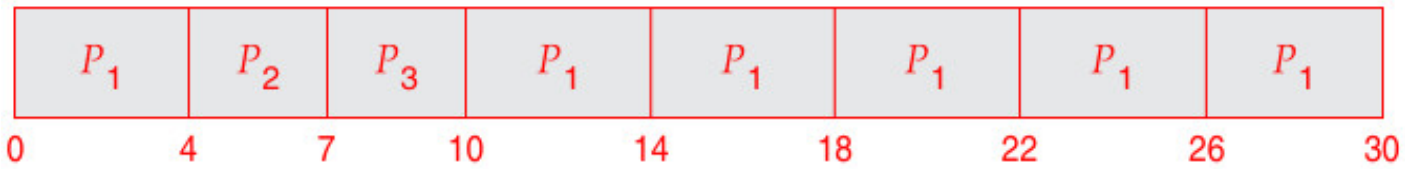
- How It Works
  1. Processes are placed in a circular queue.
  2. The first process in the queue is assigned the CPU and runs for up to one time quantum.
  3. If the process completes before the time quantum expires, it releases the CPU, and the next process in the queue is executed.
  4. If the process does not complete within the time quantum, it is preempted and placed at the end of the queue. The next process in the queue is then executed.
  5. This process repeats until all processes have completed.

Round Robin Scheduling

- For example: Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds. We use a time quantum of 4 milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

  - The resulting RR schedule is as follows:



The average waiting time is $17/3 = 5.66$ milliseconds.

Priority Scheduling

- **Priority Scheduling** is a CPU scheduling algorithm where each process is assigned a priority. The CPU is allocated to the process with the highest priority in the ready queue.
- If two processes have the same priority, they are scheduled in FCFS order, meaning the process that arrived first is executed first.
- **Advantages**
  - Simple and easy to implement.
  - Prioritizes important processes based on their priority.
- **Disadvantages**
  - **Starvation**: Low-priority processes may never get executed if high-priority processes keep arriving.
  - Not optimal for systems requiring fast response times, as it lacks a priority mechanism.

Priority Scheduling

- For example: consider the following set of processes, assumed to have arrived at time 0 in the order P_1, P_2, ..., P_5 , with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

  - The Gantt Chart for the schedule is:

The average waiting time is 8.2 milliseconds.

Priority Scheduling

- For example: consider the following set of processes, assumed to have arrived at time 0 in the order P_1, P_2, ..., P_5 , with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

  - The Gantt Chart for the schedule using a time quantum of 2 milliseconds:

The average waiting time is 14 milliseconds.

Questions