CPU Scheduling

CPU scheduling involves selecting a process from the ready queue and allocating the CPU to that process. The goal is to optimize CPU utilization, maximize throughput, minimize turnaround time, and ensure fairness among processes.

1. First-Come, First-Served (FCFS)

- **Non-preemptive** scheduling algorithm where processes are executed in the order they arrive in the ready queue.
- Once a process starts running, it continues until it terminates or blocks.

Implementaion: The First-Come, First-Served (FCFS) algorithm schedules processes in the order they arrive. It's non-preemptive, meaning each process runs to completion before the next starts. C++ code sorts processes by arrival time and calculates waiting and turnaround times based on when each process starts and finishes execution.

Advantages:

- Simple and easy to understand.
- No starvation (every process gets CPU eventually).

Disadvantages:

- Poor in terms of average waiting time, especially if long processes arrive first (convoy effect).
- Not suitable for time-sharing systems or interactive applications.

2. Shortest Job First (SJF)

- **Non-preemptive:** Selects the process with the smallest execution time next.

  Implementation: The Shortest Job First (SJF) algorithm selects processes with the shortest burst time next. C++ code sorts processes by arrival and burst times, then iteratively picks the shortest available process. It's non-preemptive, so once a process starts, it runs to completion, with waiting and turnaround times calculated accordingly.

- **Preemptive:** Switches to a shorter job that arrives if it has a smaller remaining time than the currently running process.

  Implementation: The Shortest Job First (Preemptive) algorithm, also known as Shortest Remaining Time First (SRTF), selects the process with the shortest remaining burst time at every time unit. C++ code updates current time, remaining burst times, and checks for process completion, ensuring accurate waiting and turnaround time calculations.

Advantages:

- Minimizes average waiting time (optimal for minimizing average turnaround time).
- Efficient for batch systems where job lengths are known in advance.

Disadvantages:

- Difficult to predict job lengths accurately in real-time systems.
- Preemptive SJF can lead to starvation of longer jobs if shorter jobs keep arriving.

3. Round Robin (RR)

- Each process is assigned a fixed time slice (time quantum), and CPU switches between processes in the ready queue after the time slice expires.
- If a process doesn't finish within its time slice, it's put at the end of the ready queue.

Implementation: The Round Robin algorithm assigns a fixed time quantum for process execution, cycling through processes in a circular queue. C++ code iterates over processes, executing each for the time quantum or until completion. Remaining times are updated, and waiting and turnaround times are calculated once all processes finish.

Advantages:

- Fair sharing of CPU among processes.
- Suitable for time-sharing systems, ensures responsiveness.

Disadvantages:

- High context-switching overhead (especially with smaller time quanta).
- Performance depends heavily on the choice of time quantum.

4. Priority Scheduling

- Each process is assigned a priority, and the CPU is allocated to the highest priority process.
- **Preemptive:** If a higher priority process arrives, it preempts the currently running process.

  It's Implementaion: The Priority (Preemptive) algorithm schedules processes based on priority, preempting the current process if a higher priority one arrives. C++ code updates current time, remaining burst times, and checks for process completion, recalculating priorities dynamically. Waiting and turnaround times are computed after each process completes execution.

- **Non-preemptive:** The current process continues until it finishes or blocks.

Implementation: The Priority (Non-Preemptive) algorithm schedules processes based on priority, with lower numbers indicating higher priority. C++ code sorts processes by arrival time and priority, then selects the highest priority process available. Once a process starts, it runs to completion, calculating waiting and turnaround times accordingly.

Advantages:

- Allows for prioritization of important processes.
- Can be tailored to meet specific system requirements.

Disadvantages:

- **Preemptive:** Starvation of lower priority processes (solution: aging).
- **Non-preemptive:** May not utilize CPU efficiently if a high priority process runs for a long time.

Comparison Table

Algorithm	Preemption	Advantages	Disadvantages
FCFS	Non-preemptive	Simple	Poor average waiting time (convoy effect)
SJF (Non-preemptive)	Non-preemptive	Minimizes waiting time	Requires knowledge of job lengths
SJF (Preemptive)	Preemptive	Minimizes waiting time	Starvation of longer jobs
Round Robin	Preemptive	Fairness, suitable for time-sharing	High overhead, depends on time quantum
Priority (Preemptive)	Preemptive	Prioritization of important processes	Starvation of lower priority processes
Priority (Non-preemptive)	Non-preemptive	Allows important jobs to complete	Inefficient CPU usage for long-running jobs

RESOURCES :

1.https://www.youtube.com/playlist?list=PLBlnK6fEyqRitWSE_AyyySWfhRgyA-rHk

2.https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/

3.Chapter-4 of Linux Kernel Development by Robert Love (For implementation Details)

4.https://www.youtube.com/watch?v=I2UBjN5ER4s