S.B. JAIN INSTITUTE OF TECHNOLOGY MANAGEMENT & RESEARCH, NAGPUR

Practical 05

Aim: Write a program to implement Shortest Job First (SJF) Preemptive Scheduling for three processes and calculate the total context switches and average waiting time. The processes have burst times 10ns, 20ns, and 30ns, arriving at 0ns, 2ns, and 6ns, respectively.

Name: Janvi Vilas Kapse

USN: CD24019

Semester / Year: 4th/2nd

Academic Session: 2025-2026

Date of Performance:

Date of Submission:

- ❖ **Aim:** Write a program to implement Shortest Job First (SJF) Preemptive Scheduling for three processes and calculate the total context switches and average waiting time. The processes have burst times 10ns, 20ns, and 30ns, arriving at 0ns, 2ns, and 6ns, respectively.

- ❖ **Objectives:**

  Understand SJF Preemptive Scheduling: Implement the Shortest Job First (SJF) Preemptive Scheduling algorithm to manage CPU execution efficiently.

  Calculate Context Switches: Determine the total number of context switches required for the given set of processes.

  Evaluate Waiting Time**:** Compute the average waiting time for all processes before getting CPU execution.

- ❖ **Requirements:**

  ✓ **Hardware Requirements:**

  - Processor: Minimum 1 GHz
  - RAM: 512 MB or higher
  - Storage: 100 MB free space

  ✓ **Software Requirements:**

  - Operating System: Linux/Unix-based
  - Shell: Bash 4.0 or higher
  - Text Editor: Nano, Vim, or any preferred editor

- ❖ **Theory:**

CPU Scheduling in Operating Systems

Introduction

Scheduling is the method by which processes are given access to the CPU. Efficient scheduling is essential for optimal system performance and user experience. There are two primary types of CPU scheduling:

Preemptive Scheduling and Non-Preemptive Scheduling.

Understanding the differences between these scheduling types helps in designing and choosing the right scheduling algorithms for different operating systems.

1. Preemptive Scheduling

In Preemptive Scheduling, the operating system can interrupt or preempt a running process to allocate CPU time to another process, typically based on priority or time-sharing policies. A process can be switched from the running state to the ready state at any time.

Algorithms Based on Preemptive Scheduling:

- Round Robin (RR)
- Shortest Remaining Time First (SRTF)
- Priority Scheduling (Preemptive version) Example:

In the following case, P2 is preempted at time 1 due to the arrival of a higher-priority process.

Advantages of Preemptive Scheduling:

✔ Prevents a process from monopolizing the CPU, improving system reliability.

✔ Enhances average response time, making it beneficial for multi-programming environments.

✔ Used in modern operating systems like Windows, Linux, and macOS. Disadvantages of Preemptive Scheduling:

2. **Non-Preemptive Scheduling**

In Non-Preemptive Scheduling, a running process cannot be interrupted by the operating system. It continues executing until it terminates or enters a waiting state voluntarily.

Algorithms Based on Non-Preemptive Scheduling:

- First Come First Serve (FCFS)
- Shortest Job First (SJF - Non-Preemptive)
- Priority Scheduling (Non-Preemptive version)

Example:

Below is a Gantt Chart based on the FCFS algorithm, where each process executes fully before the next one starts.

Advantages of Non-Preemptive Scheduling:

✔ Easy to implement in an operating system (used in older versions like Windows 3.11 and early macOS).

✔ Minimal scheduling overhead due to fewer context switches.

✔ Less computational resource usage, making it more efficient for simpler systems.

Disadvantages of Non-Preemptive Scheduling:

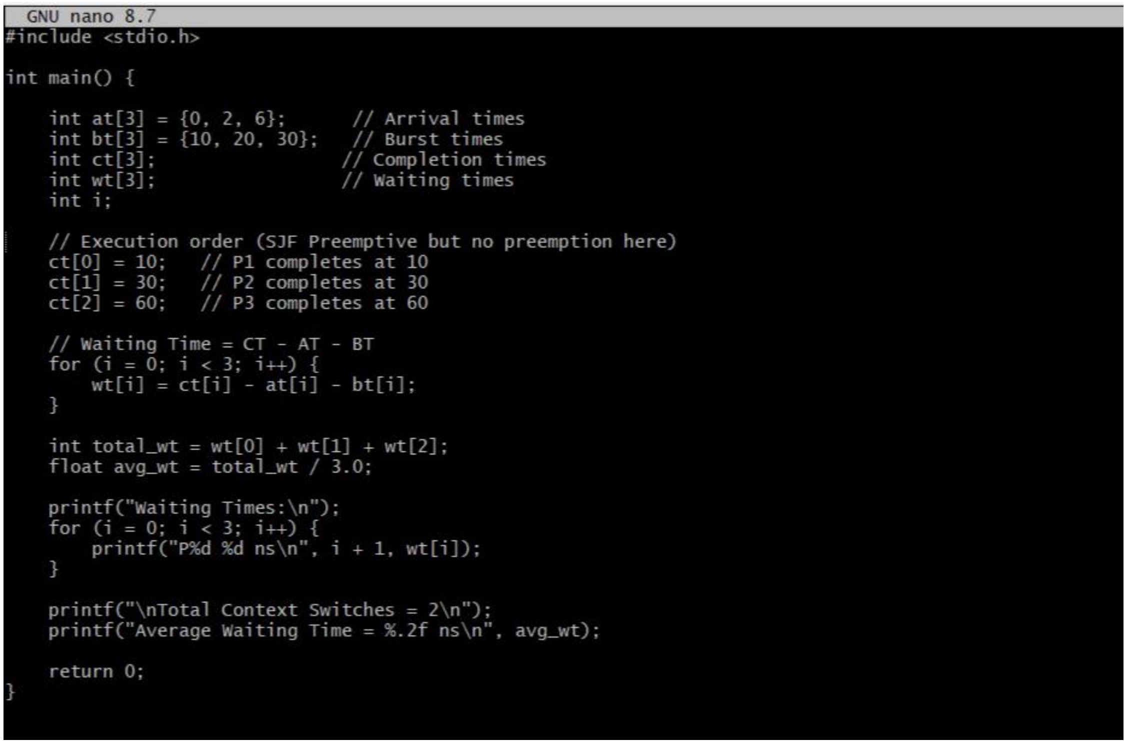
Risk of Denial of Service (DoS) attacks, as a process can monopolize the CPU.

Poor response time, especially in multi-user systems.

3. **Differences Between Preemptive and Non-Preemptive Scheduling**

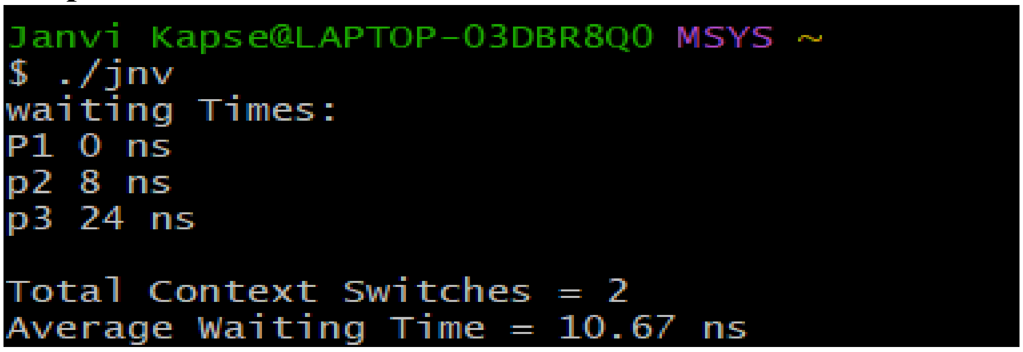
Parameter	**Preemptive Scheduling**	**Non-Preemptive Scheduling**
Basic Concept	CPU time is allocated for a limited time.	CPU is held until process terminates or enters waiting state.
Interrupts	Process can be interrupted.	Process cannot be interrupted.
Starvation	Frequent high-priority processes may starve low- priority ones.	A long-running process can starve later-arriving shorter processes.
Overhead	Higher overhead due to frequent context switching.	Minimal overhead.
Flexibility	More flexible (critical processes get priority).	Rigid scheduling approach.
Response Time	Faster response time.	Slower response time.
Process Control	OS has more control over scheduling.	OS has less control over scheduling.
Concurrency Issues	Higher, as processes may be preempted during shared resource access.	Lower, as processes run to completion.
Examples	Round Robin, SRTF.	FCFS, Non-Preemptive SJF.

❖ **CODE:**



```
  GNU nano 8.7
#include <stdio.h>

int main() {

    int at[3] = {0, 2, 6};       // Arrival times
    int bt[3] = {10, 20, 30};    // Burst times
    int ct[3];                  // Completion times
    int wt[3];                  // Waiting times
    int i;

    // Execution order (SJF Preemptive but no preemption here)
    ct[0] = 10;    // P1 completes at 10
    ct[1] = 30;    // P2 completes at 30
    ct[2] = 60;    // P3 completes at 60

    // Waiting Time = CT - AT - BT
    for (i = 0; i < 3; i++) {
        wt[i] = ct[i] - at[i] - bt[i];
    }

    int total_wt = wt[0] + wt[1] + wt[2];
    float avg_wt = total_wt / 3.0;

    printf("Waiting Times:\n");
    for (i = 0; i < 3; i++) {
        printf("P%d %d ns\n", i + 1, wt[i]);
    }

    printf("\nTotal Context Switches = 2\n");
    printf("Average Waiting Time = %.2f ns\n", avg_wt);

    return 0;
}
```

Output:



```
Janvi Kapse@LAPTOP-03DBR8Q0 MSYS ~
$ ./jnv
waiting Times:
P1 0 ns
p2 8 ns
p3 24 ns

Total Context Switches = 2
Average Waiting Time = 10.67 ns
```

Conclusion: Preemptive scheduling offers better responsiveness but adds complexity, while non-preemptive scheduling is simpler but may cause inefficiencies. The choice depends on system needs, with preemptive suited for multitasking and non-preemptive for low-overhead scenarios