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: Yash Chitmalwar

USN: CM24057

Semester / Year: 4th/2nd

Academic Session: 2025-26

Date of Performance: 10-02-26

Date of Submission: 17-02-26

- **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:

✗ More complex to implement.

✗ Involves **overhead** for suspending a running process and switching contexts.

✗ **May cause starvation** if low-priority processes are frequently preempted.

✗ Can create **concurrency issues**, especially when accessing shared resources.

---

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.

---

4. Frequently Asked Questions (FAQs)

a. How is priority determined in Preemptive scheduling?

Ans: Preemptive scheduling systems assign priority based on **task importance, deadlines, or urgency**. Higher-priority tasks execute before lower-priority ones.

b. What happens in non-preemptive scheduling if a process does not yield the CPU?

Ans: If a process does not voluntarily yield the CPU, it can lead to **starvation or deadlock**, where other tasks are unable to execute.

c. Which scheduling method is better for real-time systems?

Ans: Preemptive scheduling is better for **real-time systems**, as it allows high-priority tasks to execute immediately.

❖ CODE

```
#include <stdio.h>

#include <limits.h>

int main() {

    int n, time = 0, completed = 0;

    int contextSwitches = 0;

    int prev = -1;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    int at[n], bt[n], rt[n];

    int ct[n], wt[n];

    for (int i = 0; i < n; i++) {

        printf("Enter Arrival Time and Burst Time for P%d: ", i + 1);

        scanf("%d %d", &at[i], &bt[i]);

        rt[i] = bt[i];

    }

    printf("\n===== SRTF EXECUTION TRACE =====\n");

    while (completed < n) {

        int shortest = -1;

        int min_rt = INT_MAX;

        printf("\nTime = %d\n", time);

        printf("Arrived processes:\n");

        for (int i = 0; i < n; i++) {
```

```
        if (at[i] <= time && rt[i] > 0) {
            printf("  P%d (RT=%d)\n", i + 1, rt[i]);
        }
    }
    printf("Comparisons:\n");
    for (int i = 0; i < n; i++) {
        if (at[i] <= time && rt[i] > 0) {
            printf("  Checking P%d (RT=%d)\n", i + 1, rt[i]);
            if (rt[i] < min_rt) {
                min_rt = rt[i];
                shortest = i;
            }
        }
    }
    if (shortest == -1) {
        printf("  No process available → CPU Idle\n");
        time++;
        continue;
    }
    printf("Selected Process: P%d\n", shortest + 1);
    if (prev != -1 && prev != shortest) {
        contextSwitches++;
        printf("Context Switch: P%d → P%d\n",
               prev + 1, shortest + 1);
    }
```

```
    prev = shortest;
    // Execute for 1 unit
    rt[shortest]--;
    printf("Executing P%d for 1 unit → Remaining Time = %d\n",
          shortest + 1, rt[shortest]);
    time++;
    // Process completion
    if (rt[shortest] == 0) {
       completed++;
       ct[shortest] = time;
       wt[shortest] = ct[shortest] - at[shortest] - bt[shortest];
      printf("P%d COMPLETED at time %d\n", shortest + 1, time);
       printf("WT(P%d) = CT - AT - BT = %d - %d - %d = %d\n",
             shortest + 1,
             ct[shortest], at[shortest], bt[shortest], wt[shortest]);
    }
  }
  // Final summary
  printf("\n===== FINAL RESULTS =====\n");
  printf("Process\tAT\tBT\tCT\tWT\n");
  float avg_wt = 0;
  for (int i = 0; i < n; i++) {
    printf("P%d\t%d\t%d\t%d\t%d\n",
          i + 1, at[i], bt[i], ct[i], wt[i]);
    avg_wt += wt[i];
```

```
    }

    avg_wt /= n;

    printf("\nAverage Waiting Time = %.2f\n", avg_wt);

    printf("Total Context Switches = %d\n", contextSwitches);

    return 0;

}
```

❖ Output:

```
C:\Users\yashc\OneDrive\Des

Enter number of processes: 3
Enter Arrival Time and Burst Time for P1: 0 10
Enter Arrival Time and Burst Time for P2: 2 20
Enter Arrival Time and Burst Time for P3: 6 30

===== SRTF EXECUTION TRACE =====

Time = 0
Arrived processes:
  P1 (RT=10)
Comparisons:
  Checking P1 (RT=10)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 9

Time = 1
Arrived processes:
  P1 (RT=9)
Comparisons:
  Checking P1 (RT=9)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 8

Time = 2
Arrived processes:
  P1 (RT=8)
  P2 (RT=20)
Comparisons:
  Checking P1 (RT=8)
  Checking P2 (RT=20)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 7

Time = 3
Arrived processes:
  P1 (RT=7)
  P2 (RT=20)
Comparisons:
  Checking P1 (RT=7)
  Checking P2 (RT=20)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 6

Time = 4
Arrived processes:
  P1 (RT=6)
  P2 (RT=20)
Comparisons:
  Checking P1 (RT=6)
  Checking P2 (RT=20)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 5

Time = 5
Arrived processes:
  P1 (RT=5)
  P2 (RT=20)
Comparisons:
  Checking P1 (RT=5)
  Checking P2 (RT=20)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 4
```

```
C:\Users\yashc\OneDrive\Des

Time = 6
Arrived processes:
  P1 (RT=4)
  P2 (RT=20)
  P3 (RT=30)
Comparisons:
  Checking P1 (RT=4)
  Checking P2 (RT=20)
  Checking P3 (RT=30)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 3

Time = 7
Arrived processes:
  P1 (RT=3)
  P2 (RT=20)
  P3 (RT=30)
Comparisons:
  Checking P1 (RT=3)
  Checking P2 (RT=20)
  Checking P3 (RT=30)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 2

Time = 8
Arrived processes:
  P1 (RT=2)
  P2 (RT=20)
  P3 (RT=30)
Comparisons:
  Checking P1 (RT=2)
  Checking P2 (RT=20)
  Checking P3 (RT=30)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 1

Time = 9
Arrived processes:
  P1 (RT=1)
  P2 (RT=20)
  P3 (RT=30)
Comparisons:
  Checking P1 (RT=1)
  Checking P2 (RT=20)
  Checking P3 (RT=30)
Selected Process: P1
Executing P1 for 1 unit / Remaining Time = 0
P1 COMPLETED at time 10
WT(P1) = CT - AT - BT = 10 - 0 - 10 = 0

Time = 10
Arrived processes:
  P2 (RT=20)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=20)
  Checking P3 (RT=30)
Selected Process: P2
Context Switch: P1 ? P2
Executing P2 for 1 unit / Remaining Time = 19
```

C:\Users\yashc\OneDrive\Des

```
Time = 11
Arrived processes:
  P2 (RT=19)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=19)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 18

Time = 12
Arrived processes:
  P2 (RT=18)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=18)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 17

Time = 13
Arrived processes:
  P2 (RT=17)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=17)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 16

Time = 14
Arrived processes:
  P2 (RT=16)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=16)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 15

Time = 15
Arrived processes:
  P2 (RT=15)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=15)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 14

Time = 16
Arrived processes:
  P2 (RT=14)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=14)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 13
```

C:\Users\yashc\OneDrive\Des

```
Time = 17
Arrived processes:
  P2 (RT=13)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=13)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 12

Time = 18
Arrived processes:
  P2 (RT=12)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=12)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 11

Time = 19
Arrived processes:
  P2 (RT=11)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=11)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 10

Time = 20
Arrived processes:
  P2 (RT=10)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=10)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 9

Time = 21
Arrived processes:
  P2 (RT=9)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=9)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 8

Time = 22
Arrived processes:
  P2 (RT=8)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=8)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 7
```

C:\Users\yashc\OneDrive\Des

```
Time = 23
Arrived processes:
  P2 (RT=7)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=7)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 6

Time = 24
Arrived processes:
  P2 (RT=6)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=6)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 5

Time = 25
Arrived processes:
  P2 (RT=5)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=5)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 4

Time = 26
Arrived processes:
  P2 (RT=4)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=4)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 3

Time = 27
Arrived processes:
  P2 (RT=3)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=3)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 2

Time = 28
Arrived processes:
  P2 (RT=2)
  P3 (RT=30)
Comparisons:
  Checking P2 (RT=2)
  Checking P3 (RT=30)
Selected Process: P2
Executing P2 for 1 unit / Remaining Time = 1
```

C:\Users\yashc\OneDrive\Des

```
Arrived processes:
  P3 (RT=6)
Comparisons:
  Checking P3 (RT=6)
Selected Process: P3
Executing P3 for 1 unit / Remaining Time = 5

Time = 55
Arrived processes:
  P3 (RT=5)
Comparisons:
  Checking P3 (RT=5)
Selected Process: P3
Executing P3 for 1 unit / Remaining Time = 4

Time = 56
Arrived processes:
  P3 (RT=4)
Comparisons:
  Checking P3 (RT=4)
Selected Process: P3
Executing P3 for 1 unit / Remaining Time = 3

Time = 57
Arrived processes:
  P3 (RT=3)
Comparisons:
  Checking P3 (RT=3)
Selected Process: P3
Executing P3 for 1 unit / Remaining Time = 2

Time = 58
Arrived processes:
  P3 (RT=2)
Comparisons:
  Checking P3 (RT=2)
Selected Process: P3
Executing P3 for 1 unit / Remaining Time = 1

Time = 59
Arrived processes:
  P3 (RT=1)
Comparisons:
  Checking P3 (RT=1)
Selected Process: P3
Executing P3 for 1 unit / Remaining Time = 0
P3 COMPLETED at time 60
WT(P3) = CT - AT - BT = 60 - 6 - 30 = 24

===== FINAL RESULTS =====
Process AT      BT      CT      WT
P1      0       10      10      0
P2      2       20      30      8
P3      6       30      60      24

Average Waiting Time = 10.67
Total Context Switches = 2

--------------------------------
Process exited after 10.7 seconds with return value 0
Press any key to continue . . .
```

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.

❖ **Discussion Questions:**

1. **What is the key difference between preemptive and non-preemptive scheduling?**
2. **Why does preemptive scheduling require context switching?**
3. **Which CPU scheduling algorithm is most suitable for real-time systems and why**?
4. **Which CPU scheduling algorithm is most suitable for real-time systems and why?**
5. **Why is the Round Robin scheduling algorithm preferred in time-sharing systems?**

❖ **References:**

https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/

Date:17/02/2026

Signature
Course Coordinator
B.Tech CSE(AIML)
Sem: 4 / 2025-26