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 : Sanket Piprewar
USN : CD24043
Semester / Year : 4th / 2nd
Date of Perfomance :
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:

✗ 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.

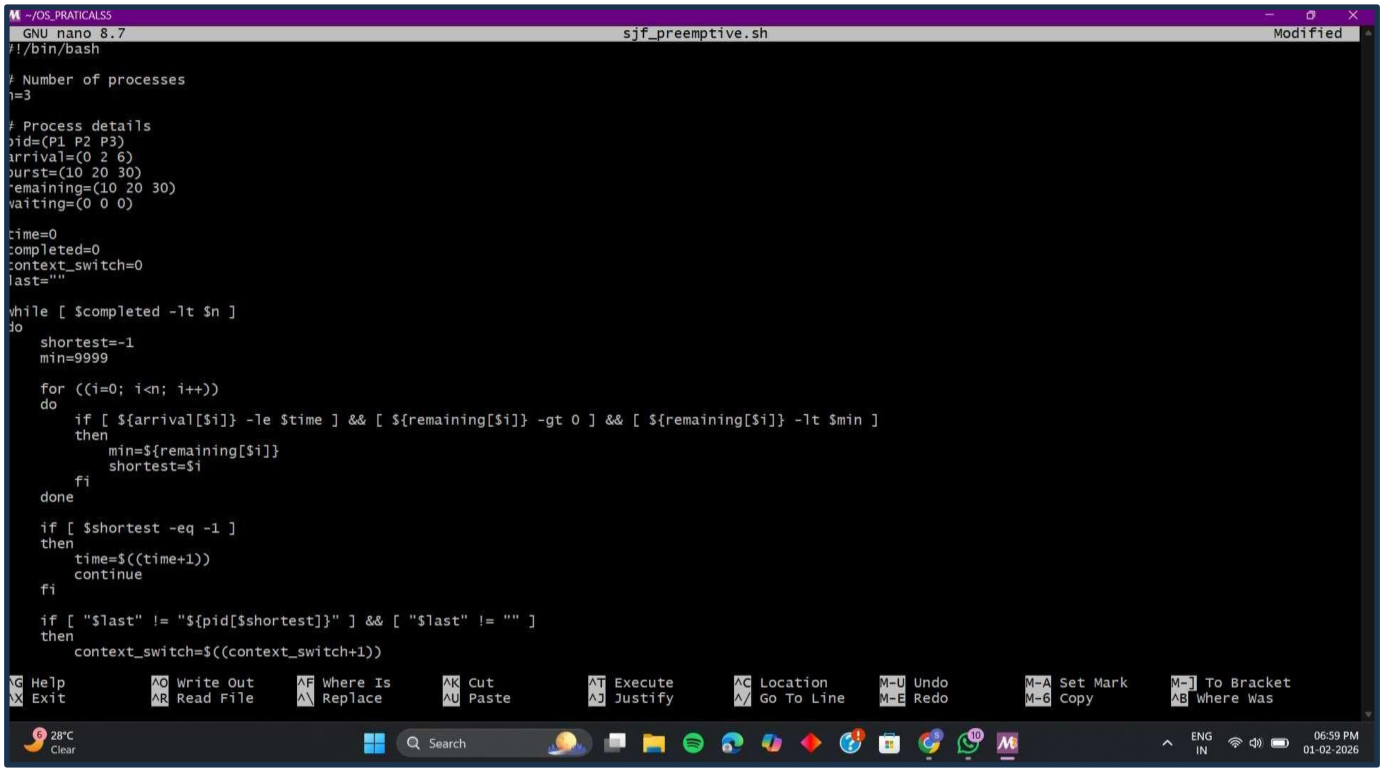
b. What happens in non-preemptive scheduling if a process does not yield the CPU?

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

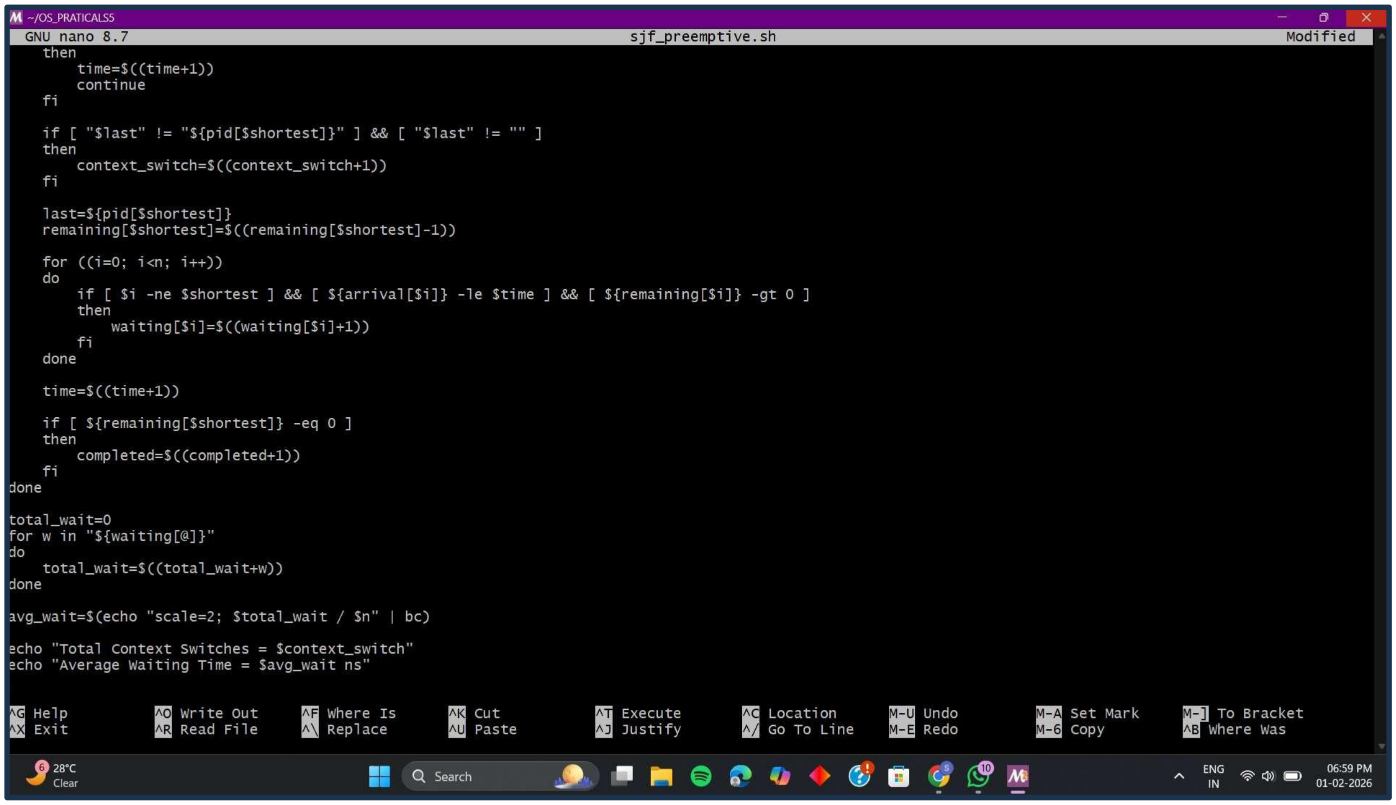
c. Which scheduling method is better for real-time systems?

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

❖ CODE:

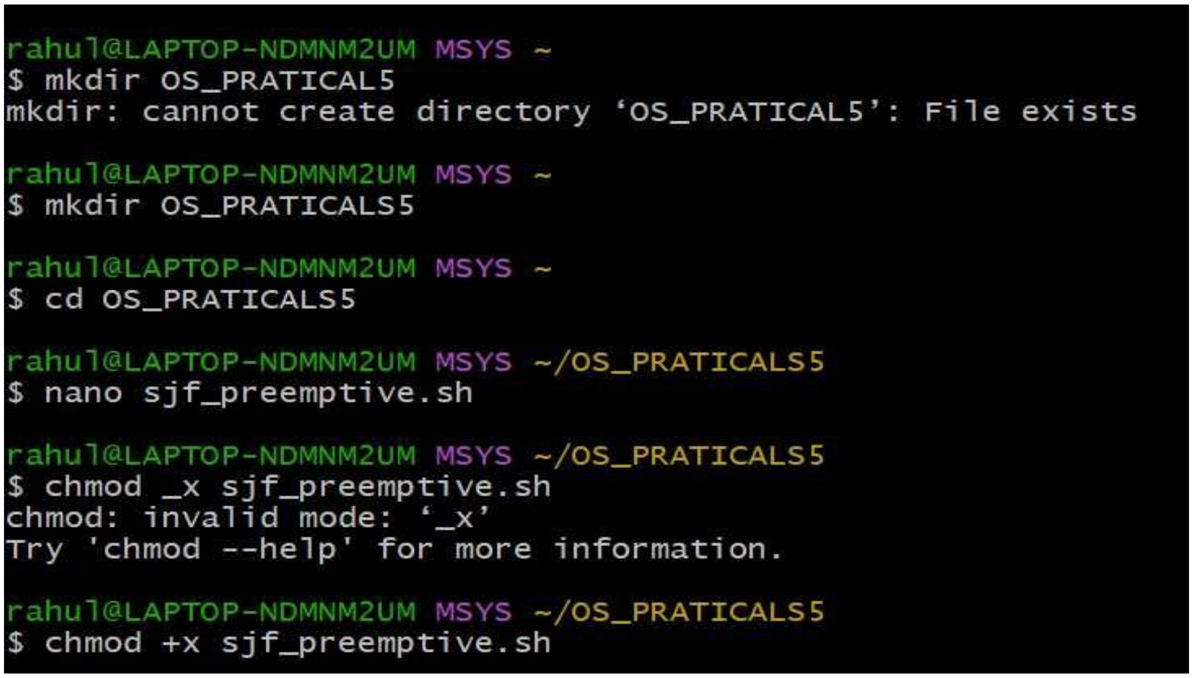


```
  GNU nano 8.7                                sjf_preemptive.sh                                Modified
#!/bin/bash

# Number of processes
n=3

# Process details
pid=(P1 P2 P3)
arrival=(0 2 6)
burst=(10 20 30)
remaining=(10 20 30)
waiting=(0 0 0)

time=0
completed=0
context_switch=0
last=""

while [ $completed -lt $n ]
do
    shortest=-1
    min=9999

    for ((i=0; i<n; i++))
    do
        if [ ${arrival[$i]} -le $time ] && [ ${remaining[$i]} -gt 0 ] && [ ${remaining[$i]} -lt $min ]
        then
            min=${remaining[$i]}
            shortest=$i
        fi
    done

    if [ $shortest -eq -1 ]
    then
        time=$((time+1))
        continue
    fi

    if [ "$last" != "${pid[$shortest]}" ] && [ "$last" != "" ]
    then
        context_switch=$((context_switch+1))

^G Help      ^O Write Out  ^F Where Is  ^K Cut     ^T Execute  ^C Location    M-U Undo   M-A Set Mark  M-] To Bracket
^X Exit      ^R Read File  ^\ Replace   ^U Paste   ^J Justify  ^/ Go To Line  M-E Redo   M-6 Copy      ^B Where Was
```

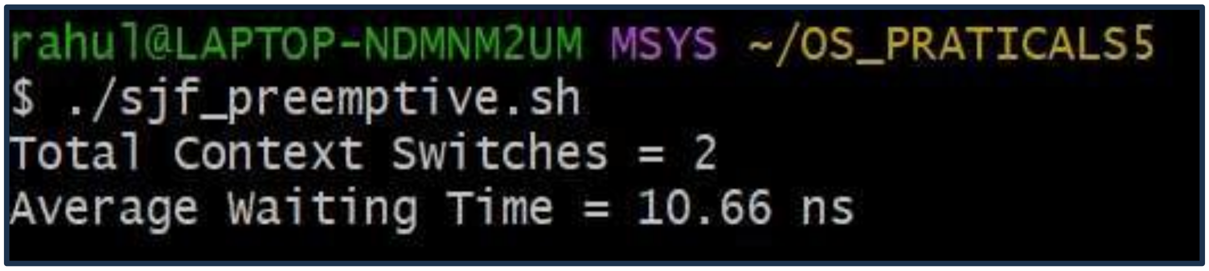


```
  GNU nano 8.7                                sjf_preemptive.sh                                Modified
    then
        time=$((time+1))
        continue
    fi

    if [ "$last" != "${pid[$shortest]}" ] && [ "$last" != "" ]
    then
        context_switch=$((context_switch+1))
    fi

    last=${pid[$shortest]}
    remaining[$shortest]=$((remaining[$shortest]-1))

    for ((i=0; i<n; i++))
    do
        if [ $i -ne $shortest ] && [ ${arrival[$i]} -le $time ] && [ ${remaining[$i]} -gt 0 ]
        then
            waiting[$i]=$((waiting[$i]+1))
        fi
    done

    time=$((time+1))

    if [ ${remaining[$shortest]} -eq 0 ]
    then
        completed=$((completed+1))
    fi
done

total_wait=0
for w in "${waiting[@]}"
do
    total_wait=$((total_wait+w))
done

avg_wait=$(echo "scale=2; $total_wait / $n" | bc)

echo "Total Context Switches = $context_switch"
echo "Average Waiting Time = $avg_wait ns"

^G Help      ^O Write Out  ^F Where Is  ^K Cut     ^T Execute  ^C Location    M-U Undo   M-A Set Mark  M-] To Bracket
^X Exit      ^R Read File  ^\ Replace   ^U Paste   ^J Justify  ^/ Go To Line  M-E Redo   M-6 Copy      ^B Where Was
```

❖ **Output:**



```
rahul@LAPTOP-NDMNM2UM MSYS ~
$ mkdir OS_PRATICAL5
mkdir: cannot create directory ‘OS_PRATICAL5’: File exists

rahul@LAPTOP-NDMNM2UM MSYS ~
$ mkdir OS_PRATICALS5

rahul@LAPTOP-NDMNM2UM MSYS ~
$ cd OS_PRATICALS5

rahul@LAPTOP-NDMNM2UM MSYS ~/OS_PRATICALS5
$ nano sjf_preemptive.sh

rahul@LAPTOP-NDMNM2UM MSYS ~/OS_PRATICALS5
$ chmod _x sjf_preemptive.sh
chmod: invalid mode: ‘_x’
Try 'chmod --help' for more information.

rahul@LAPTOP-NDMNM2UM MSYS ~/OS_PRATICALS5
$ chmod +x sjf_preemptive.sh
```



```
rahul@LAPTOP-NDMNM2UM MSYS ~/OS_PRATICALS5
$ ./sjf_preemptive.sh
Total Context Switches = 2
Average Waiting Time = 10.66 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.

❖ **Discussion Questions:**

1. **What is the key difference between preemptive and non-preemptive scheduling?**
   Ans: Preemptive scheduling allows the CPU to be taken away from a process before it finishes execution, whereas non-preemptive scheduling ensures a process runs until completion or voluntary release of the CPU.
2. **Why does preemptive scheduling require context switching?**
   Ans: Preemptive scheduling interrupts running processes, requiring the

system to save the current state (context) and load the next process, leading to context switching overhead.

1. **Which CPU scheduling algorithm is most suitable for real-time systems and why?**
   Ans: Preemptive priority scheduling is best for real-time systems as it ensures high-priority tasks get immediate CPU access, reducing response time for critical tasks.
2. **What is starvation in CPU scheduling, and how can it be prevented?**
   Ans: Starvation occurs when low-priority processes ~~wait indefinitely due to~~ frequent arrival of high-priority tasks. It can be prevented using **aging**, which gradually increases a process's priority over time.
3. **Why is the Round Robin scheduling algorithm preferred in time-sharing systems?**
   Ans: Round Robin ensures **fair CPU allocation** by assigning time slices (quantum) to each process, preventing monopolization and providing a balanced response time in multi-user environments.

❖ **References:**

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

Date:____/____/2026

: 4 / 2025-26