Experiment Name: CPU Scheduling Algorithm: Shortest Job First (Non-Preemptive)

AIM: To write a program to simulate the CPU scheduling algorithm Shortest Job First (SJF) without preemption.

DESCRIPTION:

In the non-preemptive SJF scheduling algorithm, the process with the shortest burst time is selected to execute next, but once it starts, it cannot be preempted. This means that the currently running process will execute until completion, even if a new process arrives with a shorter burst time during its execution. After a process finishes, the next shortest job is selected from the ready queue. The goal is to calculate the average waiting time and average turnaround time for all processes.

ALGORITHM:

1. Start the process.
2. Input: Accept the number of processes in the ready queue.
3. For each process, input the process ID, CPU burst time, and arrival time.
4. Initialize the current time as current_time = 0.
5. While there are processes remaining to be executed: a) Among the processes that have arrived (arrival time ≤ current_time), find the one with the shortest burst time that hasn't been executed yet.
   b) Execute the selected process until it completes. c) Record the completion time, waiting time, and turnaround time for the completed process. d) Update the current time to reflect the completion of the executed process.
6. Calculate: a) Waiting time:
   - Waiting Time = Turnaround Time - Burst Time
     b) Turnaround time:
   - Turnaround Time = Completion Time - Arrival Time
7. After all processes are completed, calculate the average: a) Average waiting time:
   - Average Waiting Time = Total Waiting Time / Number of Processes
     b) Average turnaround time:
   - Average Turnaround Time = Total Turnaround Time / Number of Processes

8. Stop the process.

SOURCE CODE:

```
#include <bits/stdc++.h>
using namespace std;

void UniqueID(string studentID) {
    int uniqueCode = 0;
    for (char c : studentID) {
        uniqueCode += (int)c;
    }
    cout << "Unique ID based on Student ID (" << studentID << "): "
<< uniqueCode << endl;
}

struct Process {
    int pid;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int waitingTime;
    int turnaroundTime;
};

bool compare(Process a, Process b) {
    if (a.arrivalTime == b.arrivalTime)
        return a.burstTime < b.burstTime;
    return a.arrivalTime < b.arrivalTime;
}

void SJFNonPreemptive(vector<Process> &processes) {
    int n = processes.size();
    int currentTime = 0;
    vector<bool> isCompleted(n, false);

    sort(processes.begin(), processes.end(), compare);

    for (int i = 0; i < n; i++) {
        int shortestProcess = -1;
        int minBurstTime = INT_MAX;
```

```
        for (int j = 0; j < n; j++) {
            if (processes[j].arrivalTime <= currentTime &&
!isCompleted[j] && processes[j].burstTime < minBurstTime) {
                minBurstTime = processes[j].burstTime;
                shortestProcess = j;
            }
        }

        if (shortestProcess != -1) {
            currentTime += processes[shortestProcess].burstTime;
            processes[shortestProcess].completionTime = currentTime;
            processes[shortestProcess].turnaroundTime =
processes[shortestProcess].completionTime -
processes[shortestProcess].arrivalTime;
            processes[shortestProcess].waitingTime =
processes[shortestProcess].turnaroundTime -
processes[shortestProcess].burstTime;
            isCompleted[shortestProcess] = true;
        } else {
            currentTime++;
        }
    }
    cout << "\nProcess\tArrival Time\tBurst Time\tWaiting
Time\tTurnaround Time\tCompletion Time\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << processes[i].pid << "\t\t" <<
processes[i].arrivalTime << "\t\t"
             << processes[i].burstTime << "\t\t" <<
processes[i].waitingTime << "\t\t"
             << processes[i].turnaroundTime << "\t\t" <<
processes[i].completionTime << endl;
    }
    double totalWaitingTime = 0, totalTurnaroundTime = 0;
    for (int i = 0; i < n; i++) {
        totalWaitingTime += processes[i].waitingTime;
        totalTurnaroundTime += processes[i].turnaroundTime;
    }
    cout << "\nAverage Waiting Time: " << totalWaitingTime / n <<
endl;
```

```
    cout << "Average Turnaround Time: " << totalTurnaroundTime / n <<
endl;
}

int main() {
    string studentID = "2125051016";
    UniqueID(studentID);
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;

    vector<Process> processes(n);
    for (int i = 0; i < n; i++) {
        cout << "Enter Arrival Time and Burst Time for Process " << i
+ 1 << ": ";
        processes[i].pid = i + 1;
        cin >> processes[i].arrivalTime >> processes[i].burstTime;
    }
    SJFNonPreemptive(processes);

    return 0;
}
```

INPUT AND OUTPUT :

```
Unique ID based on Student ID (2125051016): 503
Enter the number of processes: 2
Enter Arrival Time and Burst Time for Process 1: 2 3
Enter Arrival Time and Burst Time for Process 2: 1 4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time Completion Time
P2              1               4               0               4               5
P1              2               3               0               0               0

Average Waiting Time: 0
Average Turnaround Time: 2

Process returned 0 (0x0)   execution time : 11.573 s
Press any key to continue.
```

SOURCE CODE:

```
#!/bin/bash
studentID="2125051016"

uniqueCode=0

for (( i=0; i<${#studentID}; i++ )); do
        uniqueCode=$(( uniqueCode + $(printf "%d"
"'${studentID:$i:1}") ))
    done
    echo "Unique ID based on Student ID (${studentID}): $uniqueCode"
nonPreemptiveSJF() {
    local numProcesses=$1
    local burstTimes=("${!2}")
    local arrivalTimes=("${!3}")
    local waitingTimes=()
    local turnaroundTimes=()
    local totalWaitingTime=0
    local totalTurnaroundTime=0
    for (( i=0; i<numProcesses; i++ )); do
        waitingTimes[i]=0
        turnaroundTimes[i]=0
    done

    local currentTime=0
    local completedProcesses=0

    while [ $completedProcesses -lt $numProcesses ]; do
        local minIndex=-1
        local minBurstTime=9999

        for (( i=0; i<numProcesses; i++ )); do
            if [ ${arrivalTimes[i]} -le $currentTime ] && [
$completedProcesses -lt $numProcesses ]; then
                if [ ${burstTimes[i]} -lt $minBurstTime ]; then
                    minBurstTime=${burstTimes[i]}
                    minIndex=$i
                fi
            fi
        done
        if [ $minIndex -eq -1 ]; then
            ((currentTime++))
            continue
        fi
```

```
        waitingTimes[minIndex]=$((currentTime -
arrivalTimes[minIndex]))
        turnaroundTimes[minIndex]=$((waitingTimes[minIndex] +
burstTimes[minIndex]))
        currentTime=$((currentTime + burstTimes[minIndex]))
        ((completedProcesses++))
    done

    echo -e "\nProcess\tArrival Time\tBurst Time\tWaiting
Time\tTurnaround Time"
    for (( i=0; i<numProcesses; i++ )); do
        echo -e
"P$((i+1))\t${arrivalTimes[i]}\t\t${burstTimes[i]}\t\t${waitingTimes[
i]}\t\t${turnaroundTimes[i]}"
        totalWaitingTime=$((totalWaitingTime + waitingTimes[i]))
        totalTurnaroundTime=$((totalTurnaroundTime +
turnaroundTimes[i]))
    done

    local averageWaitingTime=$(echo "scale=2; $totalWaitingTime /
$numProcesses" | bc)
    local averageTurnaroundTime=$(echo "scale=2; $totalTurnaroundTime
/ $numProcesses" | bc)

    echo -e "\nAverage Waiting Time: $averageWaitingTime"
    echo -e "Average Turnaround Time: $averageTurnaroundTime"
}
read -p "Enter the number of processes: " numProcesses
declare -a burstTimes
declare -a arrivalTimes
for (( i=0; i<numProcesses; i++ )); do
    read -p "Enter Arrival Time for Process $((i+1)): "
arrivalTimes[i]
    read -p "Enter Burst Time for Process $((i+1)): " burstTimes[i]
done
nonPreemptiveSJF $numProcesses burstTimes[@] arrivalTimes[@]
```

INPUT AND OUTPUT:

```
sajib@sajib-ubuntu:~/os$ bash nonpreamtive.bash
Unique ID based on Student ID (2125051016): 503
Enter the number of processes: 2
Enter Arrival Time for Process 1: 2
Enter Burst Time for Process 1: 3
Enter Arrival Time for Process 2: 1
Enter Burst Time for Process 2: 4

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P1      2               3               3               6
P2      1               4               0               4

Average Waiting Time: 1.50
Average Turnaround Time: 5.00
```

DISCUSSION: By running this algorithm in both bash and c++ code we can conclude that ans is correct. in non-preemptive SJF, once a process starts executing, it runs to completion without interruption. This leads to efficient utilization of the CPU, as shorter processes finish quickly, minimizing the overall waiting time.

However, SJF requires knowledge of burst times, which may not always be known in real-time. This algorithm is optimal in minimizing average waiting time compared to other methods like First Come First Serve (FCFS), but it can lead to the "starvation problem," where longer processes may be delayed indefinitely due to a continuous influx of shorter jobs.

CONCLUSION:

The non-preemptive version ensures that processes complete without interruption, it can lead to inefficiencies if longer jobs are starved.

Overall, SJF is a valuable scheduling strategy, especially in environments where job burst times are predictable, as it significantly reduces average waiting and turnaround times compared to other scheduling algorithms like FIFO and FCFS. However, careful consideration must be given to its potential drawbacks, such as process starvation.