Experiment Name: CPU Scheduling Algorithm: Shortest Job First (Preemption)

AIM: To write a program to simulate the CPU scheduling algorithm Shortest Job First (SJF) with preemption.

DESCRIPTION:

In the SJF preemptive scheduling algorithm, the process with the shortest remaining burst time is selected to execute next. If a new process arrives with a burst time shorter than the remaining burst time of the current process, preemption occurs, and the new process starts execution. This process continues until all processes have completed. The goal is to calculate the average waiting time and average turnaround time.

ALGORITHM:

1. Start the process.
2. Accept the number of processes in the ready queue.
3. For each process in the ready queue, assign the process ID, CPU burst time, and arrival time.
4. Initialize current_time = 0. 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 remaining burst time.

    b) Execute that process for 1 time unit.

    c) If the process completes execution, record its waiting time and turnaround time.

    d) If a new process arrives with a shorter burst time than the remaining time of the current process, preempt the current process.

5. Calculate:

    a) Waiting time: Waiting Time = Turnaround Time - Burst Time

    b) Turnaround time**:** Turnaround Time = Completion Time - Arrival Time

6. After all processes are completed, calculate:

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

7. 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;
}

void SJFPreemptive(vector<int> arrivalTimes, vector<int> burstTimes)
{
    int n = arrivalTimes.size();
    vector<int> waitingTime(n), turnaroundTime(n), completionTime(n);

    vector<int> remainingTime = burstTimes;
    int complete = 0, currentTime = 0, minRemainingTime = INT_MAX;
    int shortestProcess = 0, finishTime;
    bool foundProcess = false;
    int TotalwatingTime=0;
    while (complete != n) {
        for (int i = 0; i < n; i++) {
            if (arrivalTimes[i] <= currentTime && remainingTime[i] <
minRemainingTime && remainingTime[i] > 0) {
                minRemainingTime = remainingTime[i];
                shortestProcess = i;
                foundProcess = true;
            }
```

```
        }
        if (!foundProcess) {
            currentTime++;
            continue;
        }

        remainingTime[shortestProcess]--;

        minRemainingTime = remainingTime[shortestProcess];
        if (minRemainingTime == 0) {
            minRemainingTime = INT_MAX;
        }

        if (remainingTime[shortestProcess] ==0 ) {
            complete++;
            foundProcess = false;
            finishTime = currentTime + 1;

            waitingTime[shortestProcess] = finishTime -
burstTimes[shortestProcess] - arrivalTimes[shortestProcess];
            if (waitingTime[shortestProcess] < 0)
waitingTime[shortestProcess] = 0;
            turnaroundTime[shortestProcess] = finishTime -
arrivalTimes[shortestProcess];
            completionTime[shortestProcess] = finishTime;
        }
        currentTime++;
    }

    cout << "\nProcess\tArrival Time\tBurst Time\tWaiting
Time\tTurnaround Time\tCompletion Time\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << i + 1 << "\t\t" << arrivalTimes[i] << "\t\t"
<< burstTimes[i] << "\t\t"
             << waitingTime[i] << "\t\t" << turnaroundTime[i] <<
"\t\t" << completionTime[i] << endl;
            TotalwatingTime+=waitingTime[i];
    }

    float avg= float(TotalwatingTime)/float(n);
```

```
    cout <<"Avarage waiting time : "<<avg;
}

int main() {
    string studentID = "2125051016";
    UniqueID(studentID);

    int a=0,b=0,c=0;
    vector<int> arrivalTimes ;
    vector<int> burstTimes ;
    cout<<"number of process : ";
    cin>>c;

    for(int i= 1; i<=c ;i++){
        cout<<"arrivaltime for process "<<i<< ": ";
        cin>>a;
        arrivalTimes.push_back(a);
        cout<<"brusttime for process " <<i<< ": ";
        cin>>b;
        burstTimes.push_back(b);
    }

    SJFPreemptive(arrivalTimes, burstTimes);

    return 0;
}
```

INPUT AND OUTPUT :

```
Unique ID based on Student ID (2125051016): 503
number of process : 3
arrivaltime for process 1: 4
brusttime for process 1: 2
arrivaltime for process 2: 1
brusttime for process 2: 3
arrivaltime for process 3: 2
brusttime for process 3: 1

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time Completion Time
P1              4               2               1               3               7
P2              1               3               1               4               5
P3              2               1               0               1               3
Avarage waiting time : 0.666667
Process returned 0 (0x0)   execution time : 8.705 s
Press any key to continue.
```

```
#!/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"
preemptive_sjf() {
    local n=$1
    local -a arrival=("${!2}")
    local -a burst=("${!3}")
    local -a remaining_burst=("${burst[@]}")
    local -a waiting_time
    local -a turnaround_time
    local time=0
    local completed=0
    local shortest=-1
    local min_remaining_time=9999
    local total_waiting_time=0
    local total_turnaround_time=0
    local end_time
    while [ $completed -ne $n ]; do
        for (( i = 0; i < n; i++ )); do
            if [[ ${arrival[$i]} -le $time && ${remaining_burst[$i]}
-lt $min_remaining_time && ${remaining_burst[$i]} -gt 0 ]]; then
                min_remaining_time=${remaining_burst[$i]}
                shortest=$i
            fi
        done
        if [ $shortest -eq -1 ]; then
            ((time++))
            continue
        fi
        ((remaining_burst[$shortest]--))
        ((time++))
        min_remaining_time=${remaining_burst[$shortest]}

        if [ ${remaining_burst[$shortest]} -eq 0 ]; then
            min_remaining_time=9999
            ((completed++))
            end_time=$time

            waiting_time[$shortest]=$((end_time -
${arrival[$shortest]} - ${burst[$shortest]}))
```

```
            if [ ${waiting_time[$shortest]} -lt 0 ]; then
                waiting_time[$shortest]=0
            fi

            turnaround_time[$shortest]=$((waiting_time[$shortest] +
${burst[$shortest]}))

            total_waiting_time=$((total_waiting_time +
waiting_time[$shortest]))
            total_turnaround_time=$((total_turnaround_time +
turnaround_time[$shortest]))
        fi
    done
    echo -e "\nProcess\tArrival\tBurst\tWaiting\tTurnaround"
    for (( i = 0; i < n; i++ )); do
        echo -e "P$((i +
1))\t${arrival[$i]}\t${burst[$i]}\t${waiting_time[$i]}\t${turnaround_
time[$i]}"
    done

    echo "Average Waiting Time: $(bc <<< "scale=2;
$total_waiting_time / $n")"
    echo "Average Turnaround Time: $(bc <<< "scale=2;
$total_turnaround_time / $n")"
}
echo "Enter the number of processes: "
read n
declare -a arrival
declare -a burst
for (( i = 0; i < n; i++ )); do
    echo "Enter arrival time for process $((i + 1)):"
    read arrival[$i]
    echo "Enter burst time for process $((i + 1)):"
    read burst[$i]
done
preemptive_sjf $n arrival[@] burst[@]
```

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:
3
Enter burst time for process 1:
4
Enter arrival time for process 2:
2
Enter burst time for process 2:
1

Process Arrival Burst   Waiting Turnaround
P1      3       4       0       4
P2      2       1       0       1
Average Waiting Time: 0
Average Turnaround Time: 2.50
```

DISCUSSION:

SJF is efficient in minimizing the waiting time of processes compared to other scheduling algorithms like First Come First Serve (FCFS), as shorter processes get a chance to complete sooner. However, it requires knowledge of the burst times of processes, which might not always be available in real-time scenarios.

SJF preemptive scheduling is widely used in batch systems where jobs arrive simultaneously, and it is easier to know their burst times. It is optimal in terms of average waiting time but suffers from the "starvation problem": longer processes might be delayed indefinitely if there is a continuous stream of shorter jobs.

In preemptive SJF:

- The CPU always switches to the process with the smallest remaining burst time.
- This ensures that shorter processes get prioritized even after a longer process has already started execution.

CONCLUSION:

The preemptive version of SJF, also called Shortest Remaining Time First (SRTF), allows processes to be preempted if a new process arrives with a shorter burst time. This significantly reduces the overall waiting time compared to non-preemptive SJF and other algorithms like First In First Out (FIFO) or First Come First Serve (FCFS).

SJF preemptive scheduling is efficient at reducing waiting time and turnaround time, especially when there are many short processes. However, this efficiency comes at the cost of potential starvation of longer processes, which may be continuously postponed by shorter processes.

Referance:

- https://drive.google.com/drive/u/2/folders/1iIHJa8MZ_R4uyr4QT4_Ddt5G6l0-bF5R