Heaven's Light is Our Guide

Rajshahi University of Engineering and Technology, Bangladesh

Department of Computer Science and Engineering

Course No: CSE 3202
Course Title: Sessional Based on CSE 3201

Submitted To:

Mohiuddin Ahmed
Lecturer
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology.

Submitted By:

Ashraf-Ul-Alam
Roll: 1803070,
Section: B,
Department of Computer Science and Engineering,
Rajshahi University of Engineering and Technology.

Date of Submission: 05 December 2022

Experiment No: 3

Expriment Name: CPU Scheduding.

Introduction:

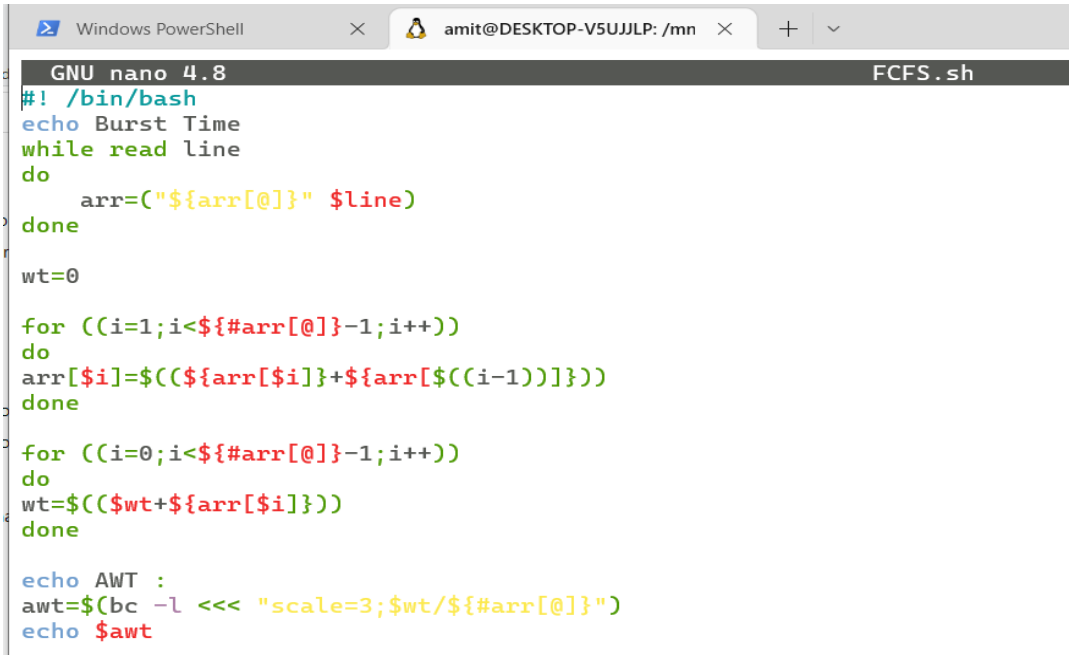
CPU Scheduling is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I / O etc, thus making full use of the CPU. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.

Here we implement two CPU scheduling algorithms:

1. First Come First Serve (FCFS): FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.
2. Shortest Job First (SJF): SJF is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed.

Command:

1. **FCFS**



Output:

```
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab3$ nano FCFS.sh
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab3$ ./FCFS.sh
Burst Time
24
3
3
AWT :
17.000
```

2. SJF

Windows PowerShell | amit@DESKTOP-V5UJJLP: /mn

GNU nano 4.8 SJF.sh

```
#! /bin/bash
echo Shortest Job First
while read line
do
    sjf=("${sjf[@]}" $line)
done

l=${#sjf[@]}

for ((i = 0; i<$l; i++))
do
    for((j = 0; j<$l-i-1; j++))
    do
        if [ ${sjf[j]} -gt ${sjf[$((j+1))]} ]
        then
            temp=${sjf[j]}
            sjf[$j]=${sjf[$((j+1))]}
            sjf[$((j+1))]=$temp
        fi
    done
done

for ((i=1;i<${#sjf[@]}-1;i++))
do
sjf[$i]=$((${sjf[$((i-1))]}+${sjf[$i]}))

done
wt=0
for ((i=0;i<${#sjf[@]}-1;i++))
do
wt=$(($wt+${sjf[$i]}))
done

awt=$(bc -l <<< "scale=3;$wt/${#sjf[@]}")
echo AWT  $awt
```

Output:

```
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab3$ nano SJF.sh
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab3$ ./SJF.sh
Shortest Job First
6
8
7
3
AWT 7.000
```

3. Comparison:

```
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab3$ ./SJF.sh
Shortest Job First
6
8
7
3
AWT 7.000
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab3$ ./FCFS.sh
Burst Time
6
8
7
3
AWT :
10.250
```

Discussion:

Both SJF and FCFS are non-preemptive in nature, but SJF gives better performance than FCFS and gives smaller average waiting time, because it executes the processes based upon the ascending order of their burst times. But implementation of SJF is complex than FCFS.