Submitted by:

Name: Aditi Sudhir Igade

Batch: A3

Roll no.:221081

PRN No.:22220311

Assignment 4

Aim : Write a program to show the demonstration of Scheduling Algorithm : 1)FCFS 2)SJF 3)Round Robin TC=3ms

Theory :

Scheduling in operating system is the process of selecting a process from a ready queue. And allotting CPU to this process for execution. The operating system schedules the processes in such a way that the CPU doesn't sit idle and keeps processing some or the other process.

The criteria include the following:

1. **CPU utilization:** The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilization can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.
2. **Throughput:** A measure of the work done by the CPU is the number of processes being executed and completed per unit of time. This is called throughput. The throughput may vary depending on the length or duration of the processes.
3. **Turnaround time:** For a particular process, an important criterion is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O. The formula to calculate Turn Around Time = Compilation Time – Arrival Time.
4. **Waiting time:** A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue. The formula for calculating Waiting Time = Turnaround Time – Burst Time.
5. **Response time:** In an interactive system, turn-around time is not the best criterion. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criterion is the time taken from submission of the process of the request until the first response is produced. This measure is called response time. The

formula to calculate Response Time = CPU Allocation Time(when the CPU was allocated for the first) – Arrival Time

6. **Completion time:** The completion time is the time when the process stops executing, which means that the process has completed its burst time and is completely executed.
7. **Priority:** If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes.
8. **Predictability:** A given process always should run in about the same amount of time under a similar system load.

Scheduling Algorithms :

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

Code :

```
// Consider below processes with length of CPU burst time in ms.
//{, p2, p3, p4, p5}
#include <bits/stdc++.h>
using namespace std;
void FCFS(vector<int> burst)
{
   vector<int> TAT;
   vector<int> WT;
   int t = 0;
   cout << "TAT  WT\n";
   for (int i = 0; i < burst.size(); i++)
   {
      t += burst[i];
      TAT.push_back(t);
      WT.push_back(t - burst[i]);
      cout << t << "   " << t - burst[i] << "\n";
```

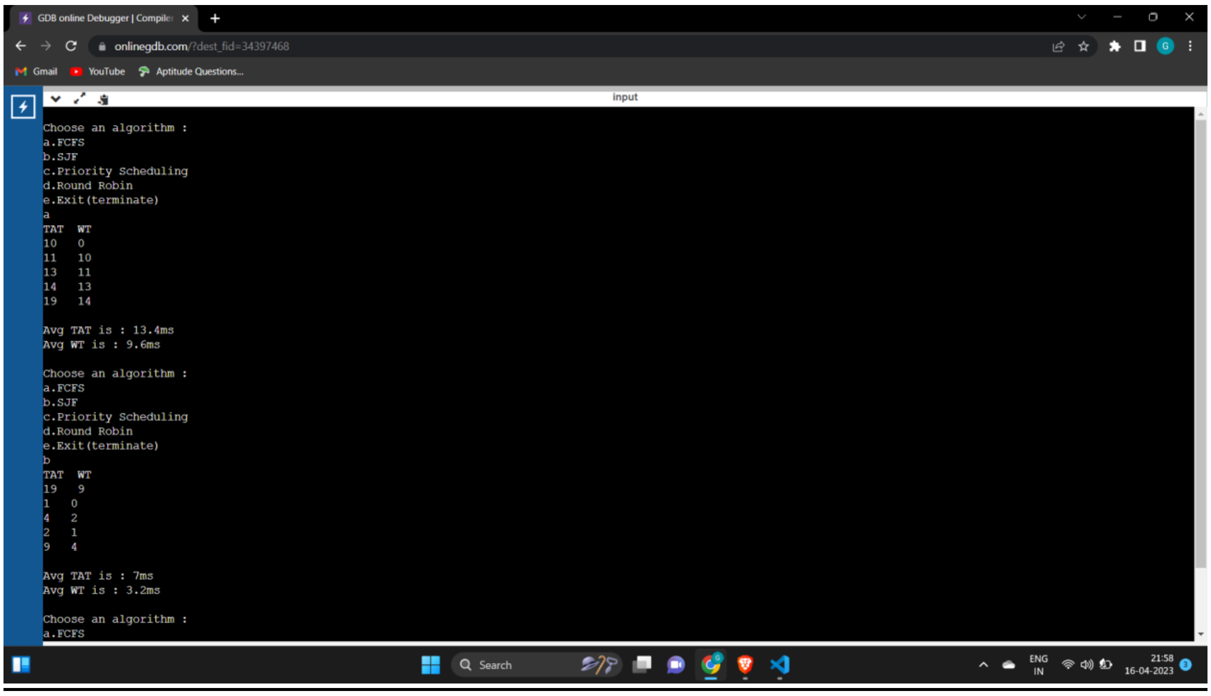
```
    }
    cout << "\nAvg TAT is : " << (float)accumulate(TAT.begin(), TAT.end(), 0) / burst.size() <<
"ms\n";
    cout << "Avg WT is : " << (float)accumulate(WT.begin(), WT.end(), 0) / burst.size() <<
"ms\n";
}
void SJF(vector<int> burst)
{
    vector<int> TAT(burst.size());
    vector<int> WT(burst.size());
    vector<pair<int, int>> vec;
    for (int i = 0; i < burst.size(); i++)
    {
        vec.push_back({burst[i], i});
    }
    sort(vec.begin(), vec.end());
    int t = 0;
    for (int i = 0; i < vec.size(); i++)
    {
        t += vec[i].first;
        TAT[vec[i].second] = t;
        WT[vec[i].second] = t - burst[vec[i].second];
    }
    cout << "TAT  WT\n";
    for (int i = 0; i < burst.size(); i++)
    {
        cout << TAT[i] << "   " << WT[i] << "\n";
    }
    cout << "\nAvg TAT is : " << (float)accumulate(TAT.begin(), TAT.end(), 0) / burst.size() <<
"ms\n";
```

```
    cout << "Avg WT is : " << (float)accumulate(WT.begin(), WT.end(), 0) / burst.size() <<
"ms\n";
}
void Priority(vector<int> burst, vector<int> priority)
{
    vector<int> TAT(burst.size());
    vector<int> WT(burst.size());
    vector<pair<int, pair<int, int>>> vec;
    for (int i = 0; i < burst.size(); i++)
    {
        vec.push_back({priority[i], {burst[i], i}});
    }
    sort(vec.begin(), vec.end());
    int t = 0;
    for (int i = 0; i < vec.size(); i++)
    {
        t += vec[i].second.first;
        TAT[vec[i].second.second] = t;
        WT[vec[i].second.second] = t - burst[vec[i].second.second];
    }
    cout << "TAT  WT\n";
    for (int i = 0; i < burst.size(); i++)
    {
        cout << TAT[i] << "   " << WT[i] << "\n";
    }
    cout << "\nAvg TAT is : " << (float)accumulate(TAT.begin(), TAT.end(), 0) / burst.size() <<
"ms\n";
    cout << "Avg WT is : " << (float)accumulate(WT.begin(), WT.end(), 0) / burst.size() <<
"ms\n";
}
```

```
void RoundR(vector<int> burst, int q)
{
    int currt = 0;
    vector<int> givenburst;
    givenburst = burst;
    vector<int> TAT(burst.size());
    vector<int> WT(burst.size());
    int processes = burst.size();
    int i = 0;
    while (processes)
    {
        if (burst[i] == 0)
        {
            i = (i + 1) % burst.size();
            continue;
        }
        if (burst[i] <= q)
        {
            currt += burst[i];
            burst[i] = 0;
            processes--;
            TAT[i] = currt;
            WT[i] = TAT[i] - givenburst[i];
        }
        else
        {
            currt += q;
            burst[i] = burst[i] - q;
        }
        i = (i + 1) % burst.size();
    }
    cout << "TAT  WT\n";
    for (int i = 0; i < burst.size(); i++)
    {
        cout << TAT[i] << "   " << WT[i] << "\n";
    }
    cout << "\nAvg TAT is : " << (float)accumulate(TAT.begin(), TAT.end(), 0) / burst.size() <<
"ms\n";
    cout << "Avg WT is : " << (float)accumulate(WT.begin(), WT.end(), 0) / burst.size() <<
"ms\n";
}
int main()
{
    vector<int> burst = {10, 1, 2, 1, 5};
    vector<int> priority = {5, 1, 3, 4, 2};
    int q = 3;
    while (true)
    {
        char choice;
        cout << "\nChoose an algorithm : \na.FCFS\nb.SJF\nc.Priority Scheduling\nd.Round
Robin\ne.Exit(terminate)\n";
        cin >> choice;
        switch (choice)
        {
        case 'a':
            FCFS(burst);
            break;
        case 'b':
            SJF(burst);
```

```
                break;
            case 'c':
                Priority(burst, priority);
                break;
            case 'd':
                RoundR(burst, q);
                break;
            case 'e':
                cout << "\nBYE !!";
                return 0;
            }
        }
        return 0;
}
```

Output :



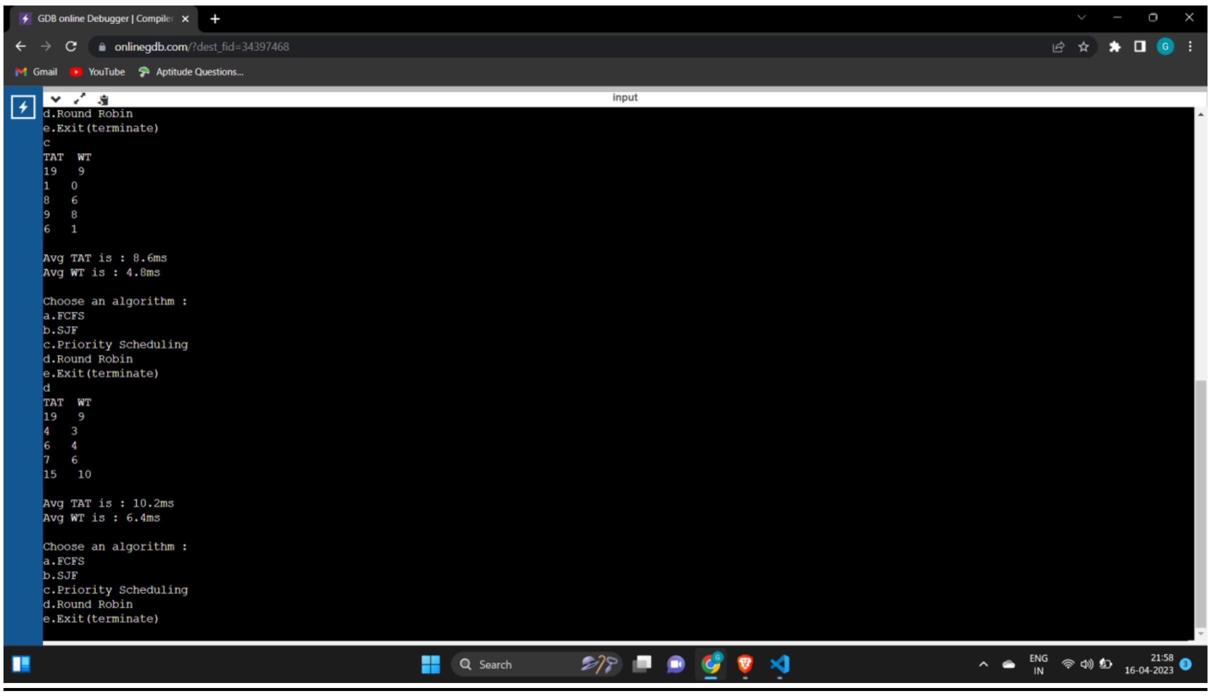
```
Choose an algorithm :
a.FCFS
b.SJF
c.Priority Scheduling
d.Round Robin
e.Exit(terminate)
a
TAT  WT
10   0
11   10
13   11
14   13
19   14

Avg TAT is : 13.4ms
Avg WT is : 9.6ms

Choose an algorithm :
a.FCFS
b.SJF
c.Priority Scheduling
d.Round Robin
e.Exit(terminate)
b
TAT  WT
19   9
1   0
4   2
2   1
9   4

Avg TAT is : 7ms
Avg WT is : 3.2ms

Choose an algorithm :
a.FCFS
```



GDB online Debugger | Compiler

onlinegdb.com/?dest_fid=34397468

Gmail YouTube Aptitude Questions...

input

```
d.Round Robin
e.Exit(terminate)
c
TAT  WT
19   9
1   0
8   6
9   8
6   1

Avg TAT is : 8.6ms
Avg WT is : 4.8ms

Choose an algorithm :
a.FCFS
b.SJF
c.Priority Scheduling
d.Round Robin
e.Exit(terminate)
d
TAT  WT
19   9
4   3
6   4
7   6
15   10

Avg TAT is : 10.2ms
Avg WT is : 6.4ms

Choose an algorithm :
a.FCFS
b.SJF
c.Priority Scheduling
d.Round Robin
e.Exit(terminate)
```

Search ENG IN 21:58 16-04-2023