Operating Systems

Experiment 4

Name: Kartik Jolapara **SAP ID: 60004200107**

Div.: B1 **Branch: Computer Engineering**

Aim -

CPU scheduling algorithms like FCFS, SJF, Round Robin etc.

Problem Statement -

1. Perform comparative assessment of various Scheduling Policies like FCFS, SJF (preemptive and non-preemptive), Priority (preemptive and non-preemptive) and Round Robin.
2. Take the input processes, their arrival time, burst time, priority, quantum from user.

Theory -

Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first.

Terms used

1. Submit time: The process at which the process is given to CPU
2. Burst time: The amount of time each process takes for execution
3. Response time: The difference between the time when the process starts execution and the submit time.
4. Turnaround time: The difference between the time when the process completes execution and the submit time.

First Come First Serve (FCFS)

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

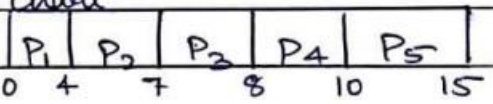
- Simple to understand
- Easy to implement

Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

Gantt Chart



Average Turnaround Time = 6.8

Average Waiting Time = 3.8

Code –

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    vector<pair<float, float>> vect;
    vector<float> TT;
    vector<float> WT;
    int burst = 0, Total = 0;
    int p;
    cout << "Enter no processess: ";
    cin >> p;
    float AT[p], BT[p];
    cout << "Now enter the times in order of->\nAT BT\n";
    for (int i = 0; i < p; i++)
    {
        cin >> AT[i];
        cin >> BT[i];
    }
    int n = sizeof(AT) / sizeof(AT[0]);
    for (int i = 0; i < n; i++)
    {
        vect.push_back(make_pair(AT[i], BT[i]));
    }
    sort(vect.begin(), vect.end());
    int cmpl_T = 0;
    Total = 0;
    for (int i = 0; i < n; i++)
    {
        int tt = 0;
        cmpl_T += vect[i].second;
        tt = cmpl_T - vect[i].first;
        TT.push_back(tt);
        Total += TT[i];
    }
    float Avg_TT = Total / (float)n;
    float total_wt = 0;
    for (int i = 0; i < n; i++)
    {
        int wt = 0;
        if (i == 0)
        {
            wt = TT[i] - vect[i].second;
            WT.push_back(wt);
        }
        else
        {
```

```
            wt = TT[i] - vect[i].second;
            WT.push_back(wt);
            total_wt += WT[i];
        }
    }
    float Avg_WT = total_wt / n;
    printf("Process\tWT \tTAT\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t%.2f\t%.2f\n", i + 1, WT[i], TT[i]);
    }
    printf("Average turn around time is : %.2f\n", Avg_TT);
    printf("Average waiting time is : %.2f\n", Avg_WT);
    return 0;
}
```

Output -

```
Enter no processess: 5
Now enter the times in order of->
AT BT
0 4
1 3
2 1
3 2
4 5
Process WT      TAT
1       0.00    4.00
2       3.00    6.00
3       5.00    6.00
4       5.00    7.00
5       6.00    11.00
Average turn around time is : 6.80
Average waiting time is : 3.80
```

Shortest Job First (SJF)

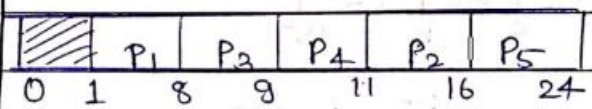
Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

Example.

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turnaround Time (TAT)	Waiting Time (WT)
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11

Gantt chart



Average Turnaround Time = 10.6

Average Waiting Time = 6.

Code –

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    vector<pair<float, float>> vect;
    vector<pair<float, float>> vect1;
    vector<pair<float, float>>::iterator it;
    vector<float> CT;
    vector<float> TT;
    vector<float> WT;
    int burst = 0, Total = 0;
    int p;
    cout << "Enter no processess: ";
    cin >> p;
    float AT[p], BT[p];
    cout << "AT BT\n";
    for (int i = 0; i < p; i++)
    {
        cin >> AT[i];
        cin >> BT[i];
    }
    int n = sizeof(AT) / sizeof(AT[0]);
    for (int i = 0; i < n; i++)
    {
        vect.push_back(make_pair(AT[i], BT[i]));
    }
    sort(vect.begin(), vect.end());
    int cmpl_T = vect[0].first + vect[0].second;
    vect1.push_back(make_pair(vect[0].first, vect[0].second));
    CT.push_back(cmpl_T);
    vect.erase(vect.begin());
    int min = 999;
    int index;
    int Total1 = 0;
    Total = 0;
    while (vect.size() > 0)
    {
        it = vect.begin();
        min = 999;
        for (int i = 0; i < vect.size(); i++)
        {
            if (vect[i].first <= cmpl_T && min > vect[i].second)
            {
                min = vect[i].second;
```

```
                index = i;
            }
        }
        if (min == 999)
        {
            for (int i = 0; i < vect.size(); i++)
            {
                if (min > vect[i].second)
                {
                    min = vect[i].second;
                    index = i;
                }
            }
            cmpl_T = vect[index].first + vect[index].second;
        }
        else
        {
            cmpl_T += vect[index].second;
        }
        CT.push_back(cmpl_T);
        int at = vect[index].first;
        int bt = vect[index].second;
        vect1.push_back(make_pair(at, bt));
        vect.erase(it + index);
    }
    for (int i = 0; i < n; i++)
    {
        int tt = CT[i] - vect1[i].first;
        int wt = tt - vect1[i].second;
        TT.push_back(tt);
        WT.push_back(wt);
        Total += TT[i];
        Total1 += WT[i];
    }
    float Avg_TT = Total / (float)n;
    float Avg_WT = Total1 / (float)n;
    printf("Process\t\tAT\tBT\tCT\tTT\tWT\n");
    for (int i = 0; i < vect1.size(); i++)
    {
        printf("%d\t\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", i + 1,
vect1[i].first, vect1[i].second,
               CT[i], TT[i], WT[i]);
    }
    printf("Average waiting time is : %.2f\n", Avg_WT);
    printf("Average turn around time is : %.2f\n", Avg_TT);
    return 0;
}
```

Output -

```
Enter no processess: 5
AT BT
1 7
2 5
3 1
4 2
5 8
Process         AT      BT      CT      TT      WT
1               1.00    7.00    8.00    7.00    0.00
2               3.00    1.00    9.00    6.00    5.00
3               4.00    2.00    11.00   7.00    5.00
4               2.00    5.00    16.00   14.00   9.00
5               5.00    8.00    24.00   19.00   11.00
Average waiting time is : 6.00
Average turn around time is : 10.60
```

Conclusion –

We have completed the execution of the scheduling algorithms FCFS and SJF(Non-preemptive)