OS ASSIGNMENT 6

NAME:SOURABH PATEL

ADMISSION NO:U19CS082

Q.Write a program for the simulation of

1. Shortest Job First (SJF)

2. Shortest Remaining Time First (SRTF) CPU scheduler.

3. Round Robin Scheduling.

SHORTEST JOB FIRST CODE:

```
#include <iostream>
using namespace std;
int mat[10][6];

void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
    for (int i = 0; i < num; i++) {
        for (int j = 0; j < num - i - 1; j++) {
            if (mat[j][1] > mat[j + 1][1]) {
                for (int k = 0; k < 5; k++) {
                    swap(mat[j][k], mat[j + 1][k]);
                }
            }
        }
    }
}

void completionTime(int num, int mat[][6])
{
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];

    for (int i = 1; i < num; i++) {
```

```
        temp = mat[i - 1][3];
        int low = mat[i][2];
        for (int j = i; j < num; j++) {
            if (temp >= mat[j][1] && low >= mat[j][2]) {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for (int k = 0; k < 6; k++) {
            swap(mat[val][k], mat[i][k]);
        }
    }
}

int main()
{
    int num, temp;

    cout << "Enter number of Process: ";
    cin >> num;

    cout << "...Enter the process ID...\n";
    for (int i = 0; i < num; i++) {
        cout << "...Process " << i + 1 << "...\n";
        cout << "Enter Process Id: ";
        cin >> mat[i][0];
        cout << "Enter Arrival Time: ";
        cin >> mat[i][1];
        cout << "Enter Burst Time: ";
        cin >> mat[i][2];
    }

    cout << "Before Arrange...\n";
    cout << "Process ID\tArrival Time\tBurst Time\n";
    for (int i = 0; i < num; i++) {
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
             << mat[i][2] << "\n";
    }

    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout << "Final Result...\n";
    cout << "Process ID\tArrival Time\tBurst Time\tWaiting "
            "Time\tTurnaround Time\n";
    for (int i = 0; i < num; i++) {
```

```
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
             << mat[i][2] << "\t\t" << mat[i][4] << "\t\t"
             << mat[i][5] << "\n";
    }
}
```

```
C:\Users\Sourabh Patel\Desktop\assignment\82\OS\assig6\a6.1.exe
Enter number of Process: 4
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 2
Enter Burst Time: 3
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 0
Enter Burst Time: 4
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 4
Enter Burst Time: 2
...Process 4...
Enter Process Id: 4
Enter Arrival Time: 5
Enter Burst Time: 4
Before Arrange...
Process ID      Arrival Time    Burst Time
1               2               3
2               0               4
3               4               2
4               5               4
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
2               0               4               0               4
3               4               2               0               2
1               2               3               4               7
4               5               4               4               8

--------------------------------
Process exited after 47.31 seconds with return value 0
Press any key to continue . . .
```

SJF is non-preemptive and in non-preemptive scheduling the processes finish their execution then only their context is switched to other processes.

SHORTEST REMAINING TIME FIRST CODE:

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;
```

```
struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int is_completed[100];
    memset(is_completed,0,sizeof(is_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";
    cin>>n;

    for(int i = 0; i < n; i++) {
        cout<<"Enter arrival time of process "<<i+1<<": ";
        cin>>p[i].arrival_time;
        cout<<"Enter burst time of process "<<i+1<<": ";
        cin>>p[i].burst_time;
        p[i].pid = i+1;
        burst_remaining[i] = p[i].burst_time;
        cout<<endl;
    }

    int current_time = 0;
    int completed = 0;
    int prev = 0;
```

```
    while(completed != n) {
        int idx = -1;
        int mn = 10000000;
        for(int i = 0; i < n; i++) {
            if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
                if(burst_remaining[i] < mn) {
                    mn = burst_remaining[i];
                    idx = i;
                }
                if(burst_remaining[i] == mn) {
                    if(p[i].arrival_time < p[idx].arrival_time) {
                        mn = burst_remaining[i];
                        idx = i;
                    }
                }
            }
        }

        if(idx != -1) {
            if(burst_remaining[idx] == p[idx].burst_time) {
                p[idx].start_time = current_time;
                total_idle_time += p[idx].start_time - prev;
            }
            burst_remaining[idx] -= 1;
            current_time++;
            prev = current_time;

            if(burst_remaining[idx] == 0) {
                p[idx].completion_time = current_time;
                p[idx].turnaround_time = p[idx].completion_time - p[idx].arriv
al_time;
                p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_ti
me;
                p[idx].response_time = p[idx].start_time - p[idx].arrival_time
;

                total_turnaround_time += p[idx].turnaround_time;
                total_waiting_time += p[idx].waiting_time;
                total_response_time += p[idx].response_time;

                is_completed[idx] = 1;
                completed++;
            }
        }
        else {
             current_time++;
        }
    }
```

```
    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_c
ompletion_time )*100;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n
"<<endl;

    for(int i = 0; i < n; i++) {
        cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<
p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<
<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;


}
```

```
C:\Users\Sourabh Patel\Desktop\assignment\82\OS\assig6\a6.2.exe
Enter the number of processes: 4
Enter arrival time of process 1: 1
Enter burst time of process 1: 6

Enter arrival time of process 2: 1
Enter burst time of process 2: 8

Enter arrival time of process 3: 2
Enter burst time of process 3: 7

Enter arrival time of process 4: 3
Enter burst time of process 4: 3



#P      AT      BT      ST      CT      TAT     WT      RT

1       1       6       1       10      9       3       0

2       1       8       17      25      24      16      16

3       2       7       10      17      15      8       8

4       3       3       3       6       3       0       0

Average Turnaround Time = 12.75
Average Waiting Time = 6.75
Average Response Time = 6.00
CPU Utilization = 96.00%
Throughput = 0.17 process/unit time

--------------------------------
Process exited after 34.75 seconds with return value 0
Press any key to continue . . .
```

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier.

ROUND ROBIN SCHEDULING CODE:

```cpp
#include<iostream>
using namespace std;

void findWaitingTime(int processes[], int n,
             int bt[], int wt[], int quantum)
{
    int rem_bt[n];
    for (int i = 0 ; i < n ; i++)
        rem_bt[i] =  bt[i];
```

```
    int t = 0;

    while (1)
    {
        bool done = true;
        for (int i = 0 ; i < n; i++)
        {
            if (rem_bt[i] > 0)
            {
                done = false;

                if (rem_bt[i] > quantum)
                {
                    t += quantum;
                    rem_bt[i] -= quantum;
                }
                else
                {
                    t = t + rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
        if (done == true)
          break;
    }
}

void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
{
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

void findavgTime(int processes[], int n, int bt[],
                                      int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, wt, quantum);

    findTurnAroundTime(processes, n, bt, wt, tat);

    cout << "Processes "<< " Burst time "
         << " Waiting time " << " Turn around time\n";
```

```
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
             << wt[i] <<"\t\t " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

int main()
{
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    int burst_time[] = {10, 5, 8};

    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```

```
C:\Users\Sourabh Patel\Desktop\assignment\82\OS\assig6\a6.3.exe
Processes  Burst time  Waiting time  Turn around time
 1              10         13              23
 2              5          10              15
 3              8          13              21
Average waiting time = 12
Average turn around time = 19.6667
--------------------------------
Process exited after 0.1154 seconds with return value 0
Press any key to continue . . .
```