Lab-2

1. C++ program for Round Robin Scheduling

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <queue>
#include <cstring>
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;
};

bool compare1(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}

bool compare2(process p1, process p2)
{
    return p1.pid < p2.pid;
}

int main() {
```

```
int n;
int tq;
struct process p[100];
float avg_turnaround_time;
float avg_waiting_time;
int total_turnaround_time = 0;
int total_waiting_time = 0;
int total_idle_time = 0;
int burst_remaining[100];
int idx;

cout << setprecision(2) << fixed;

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

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;
    burst_remaining[i] = p[i].burst_time;
    p[i].pid = i+1;
    cout<<endl;
}

sort(p,p+n,compare1);
```

```
queue<int> q;
int current_time = 0;
q.push(0);
int completed = 0;
int mark[100];
memset(mark,0,sizeof(mark));
mark[0] = 1;

while(completed != n) {
    idx = q.front();
    q.pop();

    if(burst_remaining[idx] == p[idx].burst_time) {
        p[idx].start_time = max(current_time,p[idx].arrival_time);
        total_idle_time += p[idx].start_time - current_time;
        current_time = p[idx].start_time;
    }

    if(burst_remaining[idx]-tq > 0) {
        burst_remaining[idx] -= tq;
        current_time += tq;
    }
    else {
        current_time += burst_remaining[idx];
        burst_remaining[idx] = 0;
        completed++;

        p[idx].completion_time = current_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
```

```
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;

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

    for(int i = 1; i < n; i++) {
        if(burst_remaining[i] > 0 && p[i].arrival_time <= current_time && mark[i] == 0) {
            q.push(i);
            mark[i] = 1;
        }
    }
    if(burst_remaining[idx] > 0) {
        q.push(idx);
    }

    if(q.empty()) {
        for(int i = 1; i < n; i++) {
            if(burst_remaining[i] > 0) {
                q.push(i);
                mark[i] = 1;
                break;
            }
        }
    }


}

avg_turnaround_time = (float) total_turnaround_time / n;
```

```
    avg_waiting_time = (float) total_waiting_time / n;


    sort(p,p+n,compare2);


    cout<<endl;
    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\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].completion_tim
e<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;



}
```

Output:

```
Enter the number of processes: 5
Enter Time quantum: 2
Enter arrival time of process 1: 0
Enter burst time of process 1: 5

Enter arrival time of process 2: 1
Enter burst time of process 2: 3

Enter arrival time of process 3: 2
Enter burst time of process 3: 1

Enter arrival time of process 4: 3
Enter burst time of process 4: 2

Enter arrival time of process 5: 4
Enter burst time of process 5: 3


#P      AT      BT      CT      TAT     WT

1       0       5       13      13      8

2       1       3       12      11      8

3       2       1       5       3       2

4       3       2       9       6       4

5       4       3       14      10      7

Average Turnaround Time = 8.60
Average Waiting Time = 5.80
```

2. C++ program for Shortest Remaining Job First Scheduling:

```
#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 main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    int total_turnaround_time = 0;
    int total_waiting_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].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_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;
```

```
    cout<<endl<<endl;


    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\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].completion_tim
e<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;



}
```

Output:

```
Enter the number of processes: 6
Enter arrival time of process 1: 3
Enter burst time of process 1: 4

Enter arrival time of process 2: 4
Enter burst time of process 2: 2

Enter arrival time of process 3: 5
Enter burst time of process 3: 1

Enter arrival time of process 4: 2
Enter burst time of process 4: 6

Enter arrival time of process 5: 1
Enter burst time of process 5: 8

Enter arrival time of process 6: 2
Enter burst time of process 6: 4



#P      AT      BT      CT      TAT     WT

1       3       4       13      10      6

2       4       2       9       5       3

3       5       1       7       2       1

4       2       6       19      17      11

5       1       8       26      25      17

6       2       4       6       4       0

Average Turnaround Time = 10.50
Average Waiting Time = 6.33
```

Submitted by: Gelle Hruthesh Reddy,20BCB7031