

Name : Sohag Ali

ID : 0242220005101426

Section : 63_B

Q1. FCFS with idle time and arrival time

```
#include <iostream>

#include <algorithm>

using namespace std;

struct Process {

    int id;

    int arrivalTime;

    int burstTime;

    int completionTime;

    int waitingTime;

    int turnAroundTime;

};

void findCompletionTime(Process processes[], int n) {

    sort(processes, processes + n, [](Process a, Process b) {

        return a.arrivalTime < b.arrivalTime;

    });

    int currentTime = 0;

    for (int i = 0; i < n; i++) {

        if (currentTime < processes[i].arrivalTime) {

            currentTime = processes[i].arrivalTime;

        }

    }

}
```

```

        processes[i].completionTime = currentTime + processes[i].burstTime;
        currentTime = processes[i].completionTime;

        processes[i].turnAroundTime = processes[i].completionTime - processes[i].arrivalTime;
        processes[i].waitingTime = processes[i].turnAroundTime - processes[i].burstTime;
    }
}

```

```

void displayResults(Process processes[], int n) {
    cout << "Process ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time\n";
    cout << "-----\n";
    for (int i = 0; i < n; i++) {
        cout << " P" << processes[i].id << "    |    "
            << processes[i].arrivalTime << "    |    "
            << processes[i].burstTime << "    |    "
            << processes[i].completionTime << "    |    "
            << processes[i].waitingTime << "    |    "
            << processes[i].turnAroundTime << "\n";
    }
}

```

```

int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;

```

```

Process processes[n];

for (int i = 0; i < n; i++) {

processes[i].id = i + 1;

cout << "Enter Arrival Time and Burst Time for Process P" << i + 1 << ": ";

cin >> processes[i].arrivalTime >> processes[i].burstTime;

}

findCompletionTime(processes, n);

displayResults(processes, n);

return 0;

}

```

```

D:\c++\sohag_FCFS.exe
Enter number of processes: 4
Enter Arrival Time and Burst Time for Process P1: 2 6
Enter Arrival Time and Burst Time for Process P2: 3 5
Enter Arrival Time and Burst Time for Process P3: 1 6
Enter Arrival Time and Burst Time for Process P4: 4 9
Process ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time
-----
P3 | 1 | 6 | 7 | 0 | 6
P1 | 2 | 6 | 13 | 5 | 11
P2 | 3 | 5 | 18 | 10 | 15
P4 | 4 | 9 | 27 | 14 | 23
Process returned 0 (0x0)   execution time : 12.659 s
Press any key to continue.

```

Q2. SJF with arrival time and handle idle issue

```

#include <iostream>

#include <climits>

#include <algorithm>

using namespace std;

struct Process {

```

```
int id;
int arrivalTime;
int burstTime;
int completionTime;
int waitingTime;
int turnAroundTime;
bool isCompleted;
};
```

```
bool compareArrival(Process a, Process b) {
    return a.arrivalTime < b.arrivalTime;
}
```

```
void findCompletionTime(Process processes[], int n) {
    int currentTime = 0;
    int completed = 0;

    while (completed < n) {
        int idx = -1;
        int minBurstTime = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime) {
                if (processes[i].burstTime < minBurstTime) {
                    minBurstTime = processes[i].burstTime;
                    idx = i;
                }
            }
        }
    }
}
```

```

    }
}
}

```

```

if (idx != -1) {
    processes[idx].isCompleted = true;
    processes[idx].completionTime = currentTime + processes[idx].burstTime;
    currentTime = processes[idx].completionTime;
    processes[idx].turnAroundTime = processes[idx].completionTime -
processes[idx].arrivalTime;
    processes[idx].waitingTime = processes[idx].turnAroundTime - processes[idx].burstTime;
    completed++;
} else {

    currentTime++;
}
}
}

```

```

void displayResults(Process processes[], int n) {
    cout << "P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround
Time\n";
    cout << "-----\n";
    for (int i = 0; i < n; i++) {
        cout << "  P" << processes[i].id << "    |    "
        << processes[i].arrivalTime << "    |    "

```

```

        << processes[i].burstTime << "    |    "
        << processes[i].completionTime << "    |    "
        << processes[i].waitingTime << "    |    "
    << processes[i].turnAroundTime << "\n";
}
}

int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;
    Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        processes[i].isCompleted = false;
        cout << "Enter Arrival Time and Burst Time for Process P" << i + 1 << ": ";
        cin >> processes[i].arrivalTime >> processes[i].burstTime;
    }
    sort(processes, processes + n, compareArrival);
    findCompletionTime(processes, n);
    displayResults(processes, n);
    return 0;
}

```

```
D:\c++\sohag_SJF.exe
Enter number of processes: 5
Enter Arrival Time and Burst Time for Process P1: 2 6
Enter Arrival Time and Burst Time for Process P2: 3 5
Enter Arrival Time and Burst Time for Process P3: 9 6
Enter Arrival Time and Burst Time for Process P4: 4 8
Enter Arrival Time and Burst Time for Process P5: 2 9
P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time
-----
P1 | 2 | 6 | 8 | 0 | 6
P5 | 2 | 9 | 36 | 25 | 34
P2 | 3 | 5 | 13 | 5 | 10
P4 | 4 | 8 | 27 | 15 | 23
P3 | 9 | 6 | 19 | 4 | 10

Process returned 0 (0x0)   execution time : 13.669 s
Press any key to continue.
```

Q3. Priority Scheduling

```
#include <iostream>

#include <climits>

#include <algorithm>

using namespace std;

struct Process {

int id;

int arrivalTime;

int burstTime;

int completionTime;

int waitingTime;

int turnAroundTime;

bool isCompleted;

int priority;

};

bool compareArrival(Process a, Process b) {

return a.arrivalTime < b.arrivalTime;

}
```

```

void findCompletionTime(Process processes[], int n) {
    int currentTime = 0;
    int completed = 0;

    while (completed < n) {
        int idx = -1;
        int highpriority = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime) {
                if (processes[i].priority < highpriority) {
                    highpriority = processes[i].priority;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            processes[idx].isCompleted = true;
            processes[idx].completionTime = currentTime + processes[idx].burstTime;
            currentTime = processes[idx].completionTime;
            processes[idx].turnAroundTime = processes[idx].completionTime -
            processes[idx].arrivalTime;
        }
    }
}

```



```

        processes[idx].waitingTime = processes[idx].turnAroundTime - processes[idx].burstTime;
        completed++;
    } else {

        currentTime++;
    }
}
}

```

```

void displayResults(Process processes[], int n) {
    cout << "P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround
Time\n";
    cout << "-----\n";
    for (int i = 0; i < n; i++) {
        cout << " P" << processes[i].id << "    |    "
            << processes[i].arrivalTime << "    |    "
            << processes[i].burstTime << "    |    "
            << processes[i].completionTime << "    |    "
            << processes[i].waitingTime << "    |    "
            << processes[i].turnAroundTime << "\n";
    }
}

```

```

int main() {
    int n;
    cout << "Enter number of processes: ";

```

```

cin >> n;

Process processes[n];

for (int i = 0; i < n; i++) {
    processes[i].id = i + 1;
    processes[i].isCompleted = false;

    cout << "Enter Arrival Time and Burst Time & priority for Process P" << i + 1 << ": ";
    cin >> processes[i].arrivalTime >> processes[i].burstTime >> processes[i].priority;
}

sort(processes, processes + n, compareArrival);

findCompletionTime(processes, n);

displayResults(processes, n);

return 0;
}

```

```

D:\c\sohag1.exe
Enter number of processes: 4
Enter Arrival Time and Burst Time & priority for Process P1: 1 4 7
Enter Arrival Time and Burst Time & priority for Process P2: 2 5 8
Enter Arrival Time and Burst Time & priority for Process P3: 3 6 9
Enter Arrival Time and Burst Time & priority for Process P4: 1 5 9
P ID | Arrival Time | Burst Time | Completion Time | Waiting Time | Turnaround Time
P1 | 1 | 4 | 5 | 0 | 4
P4 | 1 | 5 | 15 | 9 | 14
P2 | 2 | 5 | 10 | 3 | 8
P3 | 3 | 6 | 21 | 12 | 18
Process returned 0 (0x0) execution time : 16.000 s
Press any key to continue.

```

Q4. SRFT Algorithm With Gantt Chart :

```

#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

struct Process {

```

```

    int id, arrivalTime, burstTime, remainingTime, waitingTime, turnaroundTime,
    completionTime;
};

```

```

int main() {
    int n, currentTime = 0, completed = 0;
    cout << "Enter number of processes: ";
    cin >> n;
    vector<Process> p(n);

    cout << "Enter Process arrival time and burst time:" << endl;
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        cin >> p[i].arrivalTime >> p[i].burstTime;
        p[i].remainingTime = p[i].burstTime;
    }

    vector<pair<int, int>> ganttChart;
    while (completed < n) {
        int idx = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0 && (idx == -1
|| p[i].remainingTime < p[idx].remainingTime)) {
                idx = i;
            }
        }

        if (idx != -1) {
            p[idx].remainingTime--;
            ganttChart.push_back({currentTime, p[idx].id});
            currentTime++;

            if (p[idx].remainingTime == 0) {
                p[idx].completionTime = currentTime;
                p[idx].turnaroundTime = currentTime - p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnaroundTime - p[idx].burstTime;
                completed++;
            }
        } else {
            currentTime++;
        }
    }
}

```

```

    }

    double totalWT = 0, totalTAT = 0;
    cout << "\nProcess Details:" << endl;
    for (auto &proc : p) {
        totalWT += proc.waitingTime;
        totalTAT += proc.turnaroundTime;
        cout << "P" << proc.id << " CT: " << proc.completionTime << " WT: " <<
proc.waitingTime << " TAT: " << proc.turnaroundTime << endl;
    }

    cout << "\nAvg WT: " << totalWT / n << " Avg TAT: " << totalTAT / n << endl;

    cout << "\nGantt Chart:\n";
    for (int i = 0; i < ganttChart.size(); i++) {
        cout << "| P" << ganttChart[i].second << " ";
    }
    cout << "|" << endl;

    for (int i = 0; i < ganttChart.size(); i++) {
        cout << ganttChart[i].first << " ";
    }
    cout << currentTime << endl;

    return 0;
}

```

```

D:\ishag.exe
Enter number of processes: 5
Enter Process arrival time and burst time:
4 6
1 3
7 9

5 9
7 3

Process Details:
P1 CT: 10 WT: 0 TAT: 6
P2 CT: 4 WT: 0 TAT: 3
P3 CT: 22 WT: 6 TAT: 15
P4 CT: 31 WT: 17 TAT: 26
P5 CT: 13 WT: 3 TAT: 6

Avg WT: 5.2 Avg TAT: 11.2

Gantt Chart:
| P2 | P2 | P2 | P1 | P1 | P1 | P1 | P1 | P1 | P5 | P5 | P5 | P3 | P3 | P3 | P3 | P3 | P3 | P3 | P3 | P4 | P4 | P4 | P4 | P4 | P4 | P4 | P4 |
| P4 |
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31

Process returned 0 (0x0) execution time : 24.004 s
Press any key to continue.

```

Q5. Preemptive Algorithm

```

#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

struct Process {
    int id, arrivalTime, burstTime, remainingTime, waitingTime, turnaroundTime,
    completionTime,
    priority;
};

int main() {
    int n, currentTime = 0, completed = 0;
    cout << "Enter number of processes: ";
    cin >> n;
    vector<Process> p(n);

    cout << "Enter Process arrival time, burst time, and priority:" << endl;
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        cin >> p[i].arrivalTime >> p[i].burstTime >> p[i].priority;
        p[i].remainingTime = p[i].burstTime;
    }

    vector<pair<int, int>> ganttChart;
    while (completed < n) {
        int idx = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0 && (idx == -1
|| p[i].priority <
p[idx].priority)) {
                idx = i;
            }
        }

        if (idx != -1) {
            p[idx].remainingTime--;
            ganttChart.push_back({currentTime, p[idx].id});
            currentTime++;

            if (p[idx].remainingTime == 0) {

```

```

        p[idx].completionTime = currentTime;
        p[idx].turnaroundTime = currentTime - p[idx].arrivalTime;
        p[idx].waitingTime = p[idx].turnaroundTime - p[idx].burstTime;
        completed++;
    }
} else {
    currentTime++;
}
}

double totalWT = 0, totalTAT = 0;
cout << "\nProcess Details:" << endl;
for (auto &proc : p) {
    totalWT += proc.waitingTime;
    totalTAT += proc.turnaroundTime;
    cout << "P" << proc.id << " CT: " << proc.completionTime << " WT: " <<
proc.waitingTime << " TAT: "
<< proc.turnaroundTime << " Priority: " << proc.priority << endl;
}

cout << "\nAvg WT: " << totalWT / n << " Avg TAT: " << totalTAT / n << endl;

cout << "\nGantt Chart:\n";
for (int i = 0; i < ganttChart.size(); i++) {
    cout << "| P" << ganttChart[i].second << " ";
}
cout << "|" << endl;

for (int i = 0; i < ganttChart.size(); i++) {
    cout << ganttChart[i].first << " ";
}
cout << currentTime << endl;

return 0;
}

```

```
D:\c\sohag2.exe
Enter number of processes: 3
Enter Process arrival time, burst time, and priority:
2 5 8
1 4 7
3 6 9

Process Details:
P1 CT: 10 WT: 3 TAT: 8 Priority: 8
P2 CT: 5 WT: 0 TAT: 4 Priority: 7
P3 CT: 16 WT: 7 TAT: 13 Priority: 9

Avg WT: 3.33333 Avg TAT: 8.33333

Gantt Chart:
| P2 | P2 | P2 | P2 | P1 | P1 | P1 | P1 | P1 | P3 | P3 | P3 | P3 | P3 | P3 |
1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16

Process returned 0 (0x0)  execution time : 11.182 s
Press any key to continue.
```

Q6. Round Robin Algorithm

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

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

    int t = 0;
    gantSize = 0;

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

```

        gantt[ganttSize] = processes[i];
        timeMarks[ganttSize++] = t;

        if (rem_bt[i] > quantum)
        {
            t += quantum;
            rem_bt[i] -= quantum;
        }
        else
        {
            t += rem_bt[i];
            wt[i] = t - bt[i];
            rem_bt[i] = 0;
        }
    }
}
if (done)
{
    timeMarks[ganttSize] = t;
    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;
    int gantt[100], ganttSize = 0, timeMarks[100];

    findWaitingTime(processes, n, bt, wt, quantum, gantt, ganttSize, timeMarks);
    findTurnAroundTime(processes, n, bt, wt, tat);

    cout << "PN\t BT \tWT \tTAT\n";
    for (int i = 0; i < n; i++)
    {

```



```

        total_wt += wt[i];
        total_tat += tat[i];
        cout << " " << processes[i] << "\t " << bt[i] << "\t " << wt[i] << "\t " << tat[i] <<
endl;
    }

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

    cout << "\nGantt Chart:\n|";
    for (int i = 0; i < ganttSize; i++)
        cout << " P" << gantt[i] << " |";
    cout << "\n";

    for (int i = 0; i <= ganttSize; i++)
        cout << timeMarks[i] << "\t";
    cout << endl;
}

int main()
{
    int processes[] = {4, 5, 6};
    int n = sizeof processes / sizeof processes[0];
    int burst_time[] = {9, 6, 7};
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}

```

```

D:\c\sohag3.exe
PN      BT      WT      TAT
4        9       13       22
5        6       10       16
6        7       14       21
Average waiting time = 12.3333
Average turn around time = 19.6667
Gantt Chart:
| P4 | P5 | P6 | P4 | P5 | P6 | P4 | P5 | P6 | P4 | P6 | P4 |
0      2      4      6      8     10     12     14     16     18     20     21     22

Process returned 0 (0x0)   execution time : 0.177 s
Press any key to continue.

```