ASSIGNMENT 2

PROGRAM:

#include <iostream>

#include <algorithm>

#include <queue>

#include <vector>

#include <climits>

using namespace std;

struct Process {

int id, at, bt, ct, tat, wt, priority, remainingBt;

};

struct Execution {

int processId, startTime, duration;

};

vector<Execution> ganttChart;

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

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

processes[i].tat = processes[i].ct - processes[i].at;

processes[i].wt = processes[i].tat - processes[i].bt;

}

}

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

cout << "\nID AT BT CT TAT WT\n";

float avgTat= 0, avgWt= 0;

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

cout << processes[i].id << " " << processes[i].at << " " << processes[i].bt << " " << processes[i].ct << " " << processes[i].tat << " " << processes[i].wt << "\n";

avgTat += processes[i].tat;

avgWt += processes[i].wt;

}

avgTat /= n;

avgWt /= n;

cout << endl << "Average TAT: "<< avgTat << endl;

cout << "Average WT: "<< avgWt << endl << endl;

}

void printGanttChart() {

cout << "\nGantt Chart:\n";

for (const auto &exec : ganttChart) {

cout << "Process " << exec.processId << " executed from " << exec.startTime << " to " << (exec.startTime + exec.duration) << "\n";

}

}

// First Come First Serve (FCFS)

void fcfs() {

int n;

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

cout << "Enter Arrival Time & Burst Time:\n";

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

processes[i].id = i + 1;

cout << "Process " << processes[i].id << ": " << endl;

cout << "Arrival Time: ";

cin >> processes[i].at;

cout << "Burst Time: ";

cin >> processes[i].bt;

}

ganttChart.clear();

sort(processes, processes + n, [](Process a, Process b) { return a.at < b.at; });

int currentTime = 0;

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

if (currentTime < processes[i].at) currentTime = processes[i].at;

ganttChart.push\_back({processes[i].id, currentTime, processes[i].bt});

currentTime += processes[i].bt;

processes[i].ct = currentTime;

}

calculateTimes(processes, n);

printProcesses(processes, n);

printGanttChart();

}

// Shortest Job First (Non-Preemptive)

void sjfNonPreemptive() {

int n;

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

cout << "Enter Arrival Time & Burst Time:\n";

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

processes[i].id = i + 1;

cout << "Process " << i + 1 << ": " << endl;

cout << "Arrival Time: ";

cin >> processes[i].at;

cout << "Burst Time: ";

cin >> processes[i].bt;

}

ganttChart.clear();

bool completed[n] = {false};

int completedCount = 0, currentTime = 0;

while (completedCount < n) {

int minIndex = -1, minBt = INT\_MAX;

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

if (!completed[i] && processes[i].at <= currentTime && processes[i].bt < minBt) {

minBt = processes[i].bt;

minIndex = i;

}

}

if (minIndex == -1) {

currentTime++;

continue;

}

ganttChart.push\_back({processes[minIndex].id, currentTime, processes[minIndex].bt});

currentTime += processes[minIndex].bt;

processes[minIndex].ct = currentTime;

completed[minIndex] = true;

completedCount++;

}

calculateTimes(processes, n);

printProcesses(processes, n);

printGanttChart();

}

// Shortest Job First (Preemptive)

void sjfPreemptive() {

int n;

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

cout << "Enter Arrival Time & Burst Time:\n";

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

processes[i].id = i + 1;

cout << "Process " << i + 1 << ": " << endl;

cout << "Arrival Time: ";

cin >> processes[i].at;

cout << "Burst Time: ";

cin >> processes[i].bt;

processes[i].remainingBt = processes[i].bt;

}

ganttChart.clear();

int completed = 0, currentTime = 0, lastProcess = -1;

while (completed < n) {

int minIndex = -1, minBt = INT\_MAX;

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

if (processes[i].at <= currentTime && processes[i].remainingBt > 0 && processes[i].remainingBt < minBt) {

minBt = processes[i].remainingBt;

minIndex = i;

}

}

if (minIndex == -1) {

currentTime++;

continue;

}

if (lastProcess != processes[minIndex].id) {

ganttChart.push\_back({processes[minIndex].id, currentTime, 1});

} else {

ganttChart.back().duration++;

}

lastProcess = processes[minIndex].id;

processes[minIndex].remainingBt--;

currentTime++;

if (processes[minIndex].remainingBt == 0) {

processes[minIndex].ct = currentTime;

completed++;

}

}

calculateTimes(processes, n);

printProcesses(processes, n);

printGanttChart();

}

// Round Robin

void roundRobin() {

int n, tq;

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

cout << "Enter Arrival Time & Burst Time for each process:\n";

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

processes[i].id = i + 1;

cout << "Process " << i + 1 << ": " << endl << "Arrival Time: ";

cin >> processes[i].at;

cout << "Burst Time: ";

cin >> processes[i].bt;

processes[i].remainingBt = processes[i].bt;

}

cout << "Enter Time Quantum: ";

cin >> tq;

sort(processes, processes + n, [](Process a, Process b) { return a.at < b.at; });

ganttChart.clear();

queue<int> q;

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

q.push(i);

}

int currentTime = 0;

while (!q.empty()) {

int idx = q.front();

q.pop();

if (currentTime < processes[idx].at)

currentTime = processes[idx].at;

int execTime = min(tq, processes[idx].remainingBt);

ganttChart.push\_back({processes[idx].id, currentTime, execTime});

currentTime += execTime;

processes[idx].remainingBt -= execTime;

if (processes[idx].remainingBt > 0) {

q.push(idx);

} else {

processes[idx].ct = currentTime;

}

}

calculateTimes(processes, n);

printProcesses(processes, n);

printGanttChart();

}

void priorityNonPreemptive() {

int n;

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

cout << "Enter Arrival Time, Burst Time & Priority:\n";

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

processes[i].id = i + 1;

cout << "Process " << i + 1 << ": " << endl << "Arrival Time: ";

cin >> processes[i].at;

cout << "Burst Time: ";

cin >> processes[i].bt;

cout << "Priority: ";

cin >> processes[i].priority;

}

ganttChart.clear();

sort(processes, processes + n, [](Process a, Process b) { return a.at < b.at; });

int currentTime = 0, completed = 0;

bool done[n] = {false};

while (completed < n) {

int minIndex = -1, minPriority = INT\_MAX;

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

if (!done[i] && processes[i].at <= currentTime && processes[i].priority < minPriority) {

minPriority = processes[i].priority;

minIndex = i;

}

}

if (minIndex == -1) {

currentTime++;

continue;

}

ganttChart.push\_back({processes[minIndex].id, currentTime, processes[minIndex].bt});

currentTime += processes[minIndex].bt;

processes[minIndex].ct = currentTime;

done[minIndex] = true;

completed++;

}

calculateTimes(processes, n);

printProcesses(processes, n);

printGanttChart();

}

// Priority Scheduling (Preemptive)

void priorityPreemptive() {

int n;

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

cout << "Enter Arrival Time, Burst Time & Priority:\n";

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

processes[i].id = i + 1;

cout << "Process " << i + 1 <<": " << endl << "Arrival Time: ";

cin >> processes[i].at;

cout << "Burst Time: ";

cin >> processes[i].bt;

cout << "Priority: ";

cin >> processes[i].priority;

processes[i].remainingBt = processes[i].bt;

}

ganttChart.clear();

int completed = 0, currentTime = 0, lastProcess = -1;

while (completed < n) {

int minIndex = -1, minPriority = INT\_MAX;

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

if (processes[i].at <= currentTime && processes[i].remainingBt > 0 && processes[i].priority < minPriority) {

minPriority = processes[i].priority;

minIndex = i;

}

}

if (minIndex == -1) {

currentTime++;

continue;

}

if (lastProcess != processes[minIndex].id) {

ganttChart.push\_back({processes[minIndex].id, currentTime, 1});

} else {

ganttChart.back().duration++;

}

lastProcess = processes[minIndex].id;

processes[minIndex].remainingBt--;

currentTime++;

if (processes[minIndex].remainingBt == 0) {

processes[minIndex].ct = currentTime;

completed++;

}

}

calculateTimes(processes, n);

printProcesses(processes, n);

printGanttChart();

}

int main() {

int choice;

do {

cout << endl << "\n1.FCFS \n2.SJF(NP) \n3.SJF(P) \n4.Priority(NP) \n5.Priority(P) \n6.Round Robin \n7.Exit\nEnter your choice: ";

cin >> choice;

switch (choice) {

case 1:

fcfs();

break;

case 2:

sjfNonPreemptive();

break;

case 3:

sjfPreemptive();

break;

case 4:

priorityNonPreemptive();

break;

case 5:

priorityPreemptive();

break;

case 6:

roundRobin();

break;

case 7:

cout << endl << "Exiting . . ." << endl;

break;

default:

cout << "Invalid Option! Try Again!" << endl << endl;

}

} while (choice != 7);

}

OUTPUT:

PS C:\Users\User\Desktop\OSL> g++ osl2.cpp

PS C:\Users\User\Desktop\OSL> .\a

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 1

Enter number of processes: 6

Enter Arrival Time & Burst Time:

Process 1:

Arrival Time: 0

Burst Time: 9

Process 2:

Arrival Time: 1

Burst Time: 3

Process 3:

Arrival Time: 2

Burst Time: 2

Process 4:

Arrival Time: 1

Burst Time: 4

Process 5:

Arrival Time: 4

Burst Time: 3

Process 6:

Arrival Time: 3

Burst Time: 2

ID AT BT CT TAT WT

1 0 9 9 9 0

2 1 3 12 11 8

4 1 4 16 15 11

3 2 2 18 16 14

6 3 2 20 17 15

5 4 3 23 19 16

Average TAT: 14.5

Average WT: 10.6667

Gantt Chart:

Process 1 executed from 0 to 9

Process 2 executed from 9 to 12

Process 4 executed from 12 to 16

Process 3 executed from 16 to 18

Process 6 executed from 18 to 20

Process 5 executed from 20 to 23

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 2

Enter number of processes: 5

Enter Arrival Time & Burst Time:

Process 1:

Arrival Time: 2

Burst Time: 6

Process 2:

Arrival Time: 2

Burst Time: 4

Process 3:

Arrival Time: 2

Burst Time: 2

Process 4:

Arrival Time: 2

Burst Time: 6

Process 5:

Arrival Time: 2

Burst Time: 7

ID AT BT CT TAT WT

1 2 6 14 12 6

2 2 4 8 6 2

3 2 2 4 2 0

4 2 6 20 18 12

5 2 7 27 25 18

Average TAT: 12.6

Average WT: 7.6

Gantt Chart:

Process 3 executed from 2 to 4

Process 2 executed from 4 to 8

Process 1 executed from 8 to 14

Process 4 executed from 14 to 20

Process 5 executed from 20 to 27

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 3

Enter number of processes: 4

Enter Arrival Time & Burst Time:

Process 1:

Arrival Time: 3

Burst Time: 9

Process 2:

Arrival Time: 4

Burst Time: 5

Process 3:

Arrival Time: 0

Burst Time: 7

Process 4:

Arrival Time: 2

Burst Time: 2

ID AT BT CT TAT WT

1 3 9 23 20 11

2 4 5 9 5 0

3 0 7 14 14 7

4 2 2 4 2 0

Average TAT: 10.25

Average WT: 4.5

Gantt Chart:

Process 3 executed from 0 to 2

Process 4 executed from 2 to 4

Process 2 executed from 4 to 9

Process 3 executed from 9 to 14

Process 1 executed from 14 to 23

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 4

Enter number of processes: 5

Enter Arrival Time, Burst Time & Priority:

Process 1:

Arrival Time: 0

Burst Time: 11

Priority: 2

Process 2:

Arrival Time: 5

Burst Time: 28

Priority: 0

Process 3:

Arrival Time: 12

Burst Time: 2

Priority: 3

Process 4:

Arrival Time: 2

Burst Time: 10

Priority: 1

Process 5:

Arrival Time: 9

Burst Time: 16

Priority: 4

ID AT BT CT TAT WT

1 0 11 11 11 0

4 2 10 49 47 37

2 5 28 39 34 6

5 9 16 67 58 42

3 12 2 51 39 37

Average TAT: 37.8

Average WT: 24.4

Gantt Chart:

Process 1 executed from 0 to 11

Process 2 executed from 11 to 39

Process 4 executed from 39 to 49

Process 3 executed from 49 to 51

Process 5 executed from 51 to 67

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 5

Enter number of processes: 5

Enter Arrival Time, Burst Time & Priority:

Process 1:

Arrival Time: 0

Burst Time: 3

Priority: 3

Process 2:

Arrival Time: 1

Burst Time: 4

Priority: 2

Process 3:

Arrival Time: 2

Burst Time: 6

Priority: 4

Process 4:

Arrival Time: 3

Burst Time: 4

Priority: 6

Process 5:

Arrival Time: 5

Burst Time: 2

Priority: 10

ID AT BT CT TAT WT

1 0 3 7 7 4

2 1 4 5 4 0

3 2 6 13 11 5

4 3 4 17 14 10

5 5 2 19 14 12

Average TAT: 10

Average WT: 6.2

Gantt Chart:

Process 1 executed from 0 to 1

Process 2 executed from 1 to 5

Process 1 executed from 5 to 7

Process 3 executed from 7 to 13

Process 4 executed from 13 to 17

Process 5 executed from 17 to 19

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 6

Enter number of processes: 6

Enter Arrival Time & Burst Time for each process:

Process 1:

Arrival Time: 0

Burst Time: 7

Process 2:

Arrival Time: 1

Burst Time: 4

Process 3:

Arrival Time: 2

Burst Time: 15

Process 4:

Arrival Time: 3

Burst Time: 11

Process 5:

Arrival Time: 4

Burst Time: 20

Process 6:

Arrival Time: 4

Burst Time: 9

Enter Time Quantum: 5

ID AT BT CT TAT WT

1 0 7 31 31 24

2 1 4 9 8 4

3 2 15 55 53 38

4 3 11 56 53 42

5 4 20 66 62 42

6 4 9 50 46 37

Average TAT: 42.1667

Average WT: 31.1667

Gantt Chart:

Process 1 executed from 0 to 5

Process 2 executed from 5 to 9

Process 3 executed from 9 to 14

Process 4 executed from 14 to 19

Process 5 executed from 19 to 24

Process 6 executed from 24 to 29

Process 1 executed from 29 to 31

Process 3 executed from 31 to 36

Process 4 executed from 36 to 41

Process 5 executed from 41 to 46

Process 6 executed from 46 to 50

Process 3 executed from 50 to 55

Process 4 executed from 55 to 56

Process 5 executed from 56 to 61

Process 5 executed from 61 to 66

1.FCFS

2.SJF(NP)

3.SJF(P)

4.Priority(NP)

5.Priority(P)

6.Round Robin

7.Exit

Enter your choice: 7

Exiting . . .

PS C:\Users\User\Desktop\OSL>