#include <iostream>

#include <vector>

#include <algorithm>

#include <queue>

using namespace std;

struct Process {

int id; // Process ID

int burstTime; // Burst Time

int arrivalTime; // Arrival Time

int priority; // Priority (lower value means higher priority)

int waitingTime; // Waiting Time

int turnaroundTime; // Turnaround Time

int remainingTime; // Remaining Time for SJF and Round Robin

};

// Function to get input for processes

void getInput(vector<Process>& processes) {

int n;

cout << "Enter the number of processes: ";

cin >> n;

processes.resize(n);

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

processes[i].id = i + 1;

cout << "\nEnter details for Process " << processes[i].id << ":\n";

cout << "Arrival Time: ";

cin >> processes[i].arrivalTime;

cout << "Burst Time: ";

cin >> processes[i].burstTime;

cout << "Priority: ";

cin >> processes[i].priority;

processes[i].remainingTime = processes[i].burstTime;

processes[i].waitingTime = 0;

processes[i].turnaroundTime = 0;

}

}

// First Come First Serve Scheduling

void FCFS(vector<Process> processes) {

cout << "\nFCFS Scheduling:\n";

int currentTime = 0;

for (auto &process : processes) {

if (currentTime < process.arrivalTime) {

currentTime = process.arrivalTime;

}

process.waitingTime = currentTime - process.arrivalTime;

process.turnaroundTime = process.waitingTime + process.burstTime;

currentTime += process.burstTime;

cout << "Process " << process.id

<< ": Waiting Time = " << process.waitingTime

<< ", Turnaround Time = " << process.turnaroundTime << endl;

}

}

// Shortest Job First (Preemptive) Scheduling

void SJF\_Preemptive(vector<Process> processes) {

cout << "\nSJF (Preemptive) Scheduling:\n";

int completed = 0, currentTime = 0, n = processes.size();

while (completed != n) {

int shortestJob = -1, minRemainingTime = INT\_MAX;

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

if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0 &&

processes[i].remainingTime < minRemainingTime) {

shortestJob = i;

minRemainingTime = processes[i].remainingTime;

}

}

if (shortestJob == -1) {

currentTime++;

continue;

}

processes[shortestJob].remainingTime--;

if (processes[shortestJob].remainingTime == 0) {

completed++;

int finishTime = currentTime + 1;

processes[shortestJob].turnaroundTime = finishTime - processes[shortestJob].arrivalTime;

processes[shortestJob].waitingTime = processes[shortestJob].turnaroundTime - processes[shortestJob].burstTime;

cout << "Process " << processes[shortestJob].id

<< ": Waiting Time = " << processes[shortestJob].waitingTime

<< ", Turnaround Time = " << processes[shortestJob].turnaroundTime << endl;

}

currentTime++;

}

}

// Priority Scheduling (Non-Preemptive)

void Priority\_NonPreemptive(vector<Process> processes) {

cout << "\nPriority (Non-Preemptive) Scheduling:\n";

int currentTime = 0;

sort(processes.begin(), processes.end(), [](Process a, Process b) {

return a.arrivalTime < b.arrivalTime ||

(a.arrivalTime == b.arrivalTime && a.priority < b.priority);

});

for (auto &process : processes) {

if (currentTime < process.arrivalTime) {

currentTime = process.arrivalTime;

}

process.waitingTime = currentTime - process.arrivalTime;

process.turnaroundTime = process.waitingTime + process.burstTime;

currentTime += process.burstTime;

cout << "Process " << process.id

<< ": Waiting Time = " << process.waitingTime

<< ", Turnaround Time = " << process.turnaroundTime << endl;

}

}

// Round Robin Scheduling

void RoundRobin(vector<Process> processes, int timeQuantum) {

cout << "\nRound Robin Scheduling:\n";

int currentTime = 0, n = processes.size(), completed = 0;

queue<int> readyQueue;

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

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

readyQueue.push(i);

}

}

while (completed != n) {

if (readyQueue.empty()) {

currentTime++;

continue;

}

int idx = readyQueue.front();

readyQueue.pop();

int timeSlice = min(timeQuantum, processes[idx].remainingTime);

processes[idx].remainingTime -= timeSlice;

currentTime += timeSlice;

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

completed++;

processes[idx].turnaroundTime = currentTime - processes[idx].arrivalTime;

processes[idx].waitingTime = processes[idx].turnaroundTime - processes[idx].burstTime;

cout << "Process " << processes[idx].id

<< ": Waiting Time = " << processes[idx].waitingTime

<< ", Turnaround Time = " << processes[idx].turnaroundTime << endl;

}

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

if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0 &&

find(readyQueue.begin(), readyQueue.end(), i) == readyQueue.end()) {

readyQueue.push(i);

}

}

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

readyQueue.push(idx);

}

}

}

int main() {

vector<Process> processes;

getInput(processes);

cout << "\nSelect CPU Scheduling Algorithm:\n";

cout << "1. FCFS\n2. SJF (Preemptive)\n3. Priority (Non-Preemptive)\n4. Round Robin\n";

int choice;

cin >> choice;

switch (choice) {

case 1:

FCFS(processes);

break;

case 2:

SJF\_Preemptive(processes);

break;

case 3:

Priority\_NonPreemptive(processes);

break;

case 4:

int timeQuantum;

cout << "Enter Time Quantum for Round Robin: ";

cin >> timeQuantum;

RoundRobin(processes, timeQuantum);

break;

default:

cout << "Invalid choice!";

}

return 0;

}