**Practical NO:- 4**

Name:- Danane Nikita

Roll No:- 13140

import java.util.\*;

class Process {

int pid, arrivalTime, burstTime, priority, waitingTime, turnaroundTime, completionTime, remainingTime;

public Process(int pid, int arrivalTime, int burstTime, int priority) {

this.pid = pid;

this.arrivalTime = arrivalTime;

this.burstTime = burstTime;

this.priority = priority;

this.remainingTime = burstTime;

}

}

public class Main {

static void FCFS(List<Process> processes) {

processes.sort(Comparator.comparingInt(p -> p.arrivalTime));

int time = 0;

for (Process p : processes) {

if (time < p.arrivalTime) time = p.arrivalTime;

p.waitingTime = time - p.arrivalTime;

time += p.burstTime;

p.completionTime = time;

p.turnaroundTime = p.completionTime - p.arrivalTime;

}

printResult("FCFS", processes);

}

static void SJF(List<Process> processes) {

int n = processes.size();

int completed = 0, time = 0, minm = Integer.MAX\_VALUE;

Process shortest = null;

boolean check = false;

while (completed != n) {

for (Process p : processes) {

if (p.arrivalTime <= time && p.remainingTime < minm && p.remainingTime > 0) {

minm = p.remainingTime;

shortest = p;

check = true;

}

}

if (!check) {

time++;

continue;

}

shortest.remainingTime--;

minm = shortest.remainingTime;

if (minm == 0) minm = Integer.MAX\_VALUE;

if (shortest.remainingTime == 0) {

completed++;

check = false;

shortest.completionTime = time + 1;

shortest.waitingTime = shortest.completionTime - shortest.arrivalTime - shortest.burstTime;

if (shortest.waitingTime < 0) shortest.waitingTime = 0;

shortest.turnaroundTime = shortest.burstTime + shortest.waitingTime;

}

time++;

}

printResult("SJF (Preemptive)", processes);

}

static void PriorityScheduling(List<Process> processes) {

processes.sort(Comparator.comparingInt(p -> p.arrivalTime));

int time = 0, completed = 0;

boolean[] done = new boolean[processes.size()];

while (completed < processes.size()) {

int idx = -1, highestPriority = Integer.MAX\_VALUE;

for (int i = 0; i < processes.size(); i++) {

if (!done[i] && processes.get(i).arrivalTime <= time) {

if (processes.get(i).priority < highestPriority) {

highestPriority = processes.get(i).priority;

idx = i;

}

}

}

if (idx == -1) {

time++;

} else {

Process p = processes.get(idx);

p.waitingTime = time - p.arrivalTime

time += p.burstTime;

p.completionTime = time;

p.turnaroundTime = p.completionTime - p.arrivalTime;

done[idx] = true;

completed++;

}

}

printResult("Priority (Non-Preemptive)", processes);

}

static void RoundRobin(List<Process> processes, int quantum) {

Queue<Process> q = new LinkedList<>();

int time = 0, completed = 0;

processes.sort(Comparator.comparingInt(p -> p.arrivalTime));

q.add(processes.get(0));

int i = 1;

while (!q.isEmpty()) {

Process p = q.poll();

if (p.remainingTime > quantum) {

time += quantum;

p.remainingTime -= quantum;

} else {

time += p.remainingTime;

p.waitingTime = time - p.arrivalTime - p.burstTime;

p.remainingTime = 0;

p.completionTime = time;

p.turnaroundTime = p.burstTime + p.waitingTime;

completed++;

}

while (i < processes.size() && processes.get(i).arrivalTime <= time) {

q.add(processes.get(i));

i++;

}

if (p.remainingTime > 0) q.add(p);

}

printResult("Round Robin (q=" + quantum + ")", processes);

}

static void printResult(String algo, List<Process> processes) {

System.out.println("\n--- " + algo + " ---");

double avgWT = 0, avgTAT = 0;

System.out.printf("%-5s %-12s %-10s %-10s %-10s %-10s %-10s\n",

"PID", "Arrival", "Burst", "Priority", "Waiting", "Turnaround", "Completion");

for (Process p : processes) {

avgWT += p.waitingTime;

avgTAT += p.turnaroundTime;

System.out.printf("%-5d %-12d %-10d %-10d %-10d %-10d %-10d\n",

p.pid, p.arrivalTime, p.burstTime, p.priority,

p.waitingTime, p.turnaroundTime, p.completionTime);

}

System.out.printf("Average Waiting Time: %.2f\n", avgWT / processes.size());

System.out.printf("Average Turnaround Time: %.2f\n", avgTAT / processes.size());

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

List<Process> processes = new ArrayList<>();

System.out.print("Enter number of processes: ");

int n = sc.nextInt();

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

System.out.print("Enter Arrival Time, Burst Time, Priority for P" + (i + 1) + ": ");

int at = sc.nextInt(), bt = sc.nextInt(), pr = sc.nextInt();

processes.add(new Process(i + 1, at, bt, pr));

}

FCFS(cloneList(processes));

SJF(cloneList(processes));

PriorityScheduling(cloneList(processes));

System.out.print("Enter time quantum for Round Robin: ");

int q = sc.nextInt();

RoundRobin(cloneList(processes), q);

sc.close();

}

static List<Process> cloneList(List<Process> list) {

List<Process> copy = new ArrayList<>();

for (Process p : list) {

copy.add(new Process(p.pid, p.arrivalTime, p.burstTime, p.priority));

}

return copy;

}

}

**OUTPUT :-**

Enter number of processes: 4

Enter Arrival Time, Burst Time, Priority for P1: 0 2 4

Enter Arrival Time, Burst Time, Priority for P2: 3 4 5

Enter Arrival Time, Burst Time, Priority for P3: 5 4 8

Enter Arrival Time, Burst Time, Priority for P4: 2 4 9

--- FCFS ---

PID Arrival Burst Priority Waiting Turnaround Completion

1 0 2 4 0 2 2

4 2 4 9 0 4 6

2 3 4 5 3 7 10

3 5 4 8 5 9 14

Average Waiting Time: 2.00

Average Turnaround Time: 5.50

--- SJF (Preemptive) ---

PID Arrival Burst Priority Waiting Turnaround Completion

1 0 2 4 0 2 2

2 3 4 5 3 7 10

3 5 4 8 5 9 14

4 2 4 9 0 4 6

Average Waiting Time: 2.00

Average Turnaround Time: 5.50

--- Priority (Non-Preemptive) ---

PID Arrival Burst Priority Waiting Turnaround Completion

1 0 2 4 0 2 2

4 2 4 9 0 4 6

2 3 4 5 3 7 10

3 5 4 8 5 9 14

Average Waiting Time: 2.00

Average Turnaround Time: 5.50

Enter time quantum for Round Robin: 3

--- Round Robin (q=3) ---

PID Arrival Burst Priority Waiting Turnaround Completion

1 0 2 4 0 2 2

4 2 4 9 6 10 12

2 3 4 5 6 10 13

3 5 4 8 5 9 14

Average Waiting Time: 4.25

Average Turnaround Time: 7.75

NOTE :- Priority not considered while scheduling during FCFS SJF and ROUND ROBIN scheduling algorithms.