

laboratory practice I

Practical Examination

SPPU AY 2021-22

Semester :- 1



NAME :- OJUS P. JAISWAL

YEAR & DIV :- TE A

ROLL NO. :- TACO19108

SEAT NO. :- S191094290

PRN NO. :- 72036776L

**Assignment No. B7**

**Problem Statement :-** Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).

**Solution :-**

Program :

a) FCFS =>

/\* FCFS \*/

package B2;

import java.io.\*;

import java.util.Scanner;

public class FCFS

{

public static void main(String args[])

{

int i,no\_p,burst\_time[],TT[],WT[];

float avg\_wait=0,avg\_TT=0;

burst\_time=new int[50];

TT=new int[50];

WT=new int[50];

WT[0]=0;

Scanner s=new Scanner(System.in);

System.out.println("Enter the number of process: ");

no\_p=s.nextInt();

System.out.println("\nEnter Burst Time for processes:");

for(i=0;i<no\_p;i++)

{

System.out.print("\tP"+(i+1)+": ");

burst\_time[i]=s.nextInt();

}

for(i=1;i<no\_p;i++)

{

WT[i]=WT[i-1]+burst\_time[i-1];

avg\_wait+=WT[i];

}

avg\_wait/=no\_p;

for(i=0;i<no\_p;i++)

{

TT[i]=WT[i]+burst\_time[i];

avg\_TT+=TT[i];

}

avg\_TT/=no\_p;

System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\tProcesses:");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(" Process\tBurst Time\tWaiting Time\tTurn Around Time");

for(i=0;i<no\_p;i++)

{

System.out.println("\tP"+(i+1)+"\t "+burst\_time[i]+"\t\t "+WT[i]+"\t\t "+TT[i]);

}

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

System.out.println("\nAverage waiting time : "+avg\_wait);

System.out.println("\nAverage Turn Around time : "+avg\_TT+"\n");

}

}

b) SJF (Preemptive) =>

/\* SJF (Preemptive) \*/

package B2;

import java.util.\*;

public class SJF {

public static void main (String args[])

{

Scanner sc=new Scanner(System.in);

System.out.println ("enter no of process:");

int n= sc.nextInt();

int pid[] = new int[n]; // it takes pid of process

int at[] = new int[n]; // at means arrival time

int bt[] = new int[n]; // bt means burst time

int ct[] = new int[n]; // ct means complete time

int ta[] = new int[n];// ta means turn around time

int wt[] = new int[n]; // wt means waiting time

int f[] = new int[n]; // f means it is flag it checks process is completed or not

int k[]= new int[n]; // it is also stores brust time

int i, st=0, tot=0;

float avgwt=0, avgta=0;

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

{

pid[i]= i+1;

System.out.println ("enter process " +(i+1)+ " arrival time:");

at[i]= sc.nextInt();

System.out.println("enter process " +(i+1)+ " burst time:");

bt[i]= sc.nextInt();

k[i]= bt[i];

f[i]= 0;

}

while(true){

int min=99,c=n;

if (tot==n)

break;

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

{

if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))

{

min=bt[i];

c=i;

}

}

if (c==n)

st++;

else

{

bt[c]--;

st++;

if (bt[c]==0)

{

ct[c]= st;

f[c]=1;

tot++;

}

}

}

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

{

ta[i] = ct[i] - at[i];

wt[i] = ta[i] - k[i];

avgwt+= wt[i];

avgta+= ta[i];

}

System.out.println("pid arrival burst complete turn waiting");

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

{

System.out.println(pid[i] +"\t"+ at[i]+"\t"+ k[i] +"\t"+ ct[i] +"\t"+ ta[i] +"\t"+ wt[i]);

}

System.out.println("\naverage tat is "+ (float)(avgta/n));

System.out.println("average wt is "+ (float)(avgwt/n));

sc.close();

}

}

c) Priority (Non-Preemptive) =>

/\* Priority (Non-Preemptive) \*/

package B2;

import java.util.Scanner;

public class Priority

{

int burstTime[];

int priority[];

int arrivalTime[];

String[] processId;

int numberOfProcess;

void getProcessData(Scanner input)

{

System.out.print("Enter the number of Process for Scheduling : ");

int inputNumberOfProcess = input.nextInt();

numberOfProcess = inputNumberOfProcess;

burstTime = new int[numberOfProcess];

priority = new int[numberOfProcess];

arrivalTime = new int[numberOfProcess];

processId = new String[numberOfProcess];

String st = "P";

for (int i = 0; i < numberOfProcess; i++)

{

processId[i] = st.concat(Integer.toString(i));

System.out.print("Enter the burst time for Process - " + (i) + " : ");

burstTime[i] = input.nextInt();

System.out.print("Enter the arrival time for Process - " + (i) + " : ");

arrivalTime[i] = input.nextInt();

System.out.print("Enter the priority for Process - " + (i) + " : ");

priority[i] = input.nextInt();

}

}

void sortAccordingArrivalTimeAndPriority(int[] at, int[] bt, int[] prt, String[] pid)

{

int temp;

String stemp;

for (int i = 0; i < numberOfProcess; i++)

{

for (int j = 0; j < numberOfProcess - i - 1; j++)

{

if (at[j] > at[j + 1])

{

//swapping arrival time

temp = at[j];

at[j] = at[j + 1];

at[j + 1] = temp;

//swapping burst time

temp = bt[j];

bt[j] = bt[j + 1];

bt[j + 1] = temp;

//swapping priority

temp = prt[j];

prt[j] = prt[j + 1];

prt[j + 1] = temp;

//swapping process identity

stemp = pid[j];

pid[j] = pid[j + 1];

pid[j + 1] = stemp;

}

//sorting according to priority when arrival timings are same

if (at[j] == at[j + 1])

{

if (prt[j] > prt[j + 1])

{

//swapping arrival time

temp = at[j];

at[j] = at[j + 1];

at[j + 1] = temp;

//swapping burst time

temp = bt[j];

bt[j] = bt[j + 1];

bt[j + 1] = temp;

//swapping priority

temp = prt[j];

prt[j] = prt[j + 1];

prt[j + 1] = temp;

//swapping process identity

stemp = pid[j];

pid[j] = pid[j + 1];

pid[j + 1] = stemp;

}

}

}

}

}

void priorityNonPreemptiveAlgorithm()

{

int finishTime[] = new int[numberOfProcess];

int bt[] = burstTime.clone();

int at[] = arrivalTime.clone();

int prt[] = priority.clone();

String pid[] = processId.clone();

int waitingTime[] = new int[numberOfProcess];

int turnAroundTime[] = new int[numberOfProcess];

sortAccordingArrivalTimeAndPriority(at, bt, prt, pid);

//calculating waiting & turn-around time for each process

finishTime[0] = at[0] + bt[0];

turnAroundTime[0] = finishTime[0] - at[0];

waitingTime[0] = turnAroundTime[0] - bt[0];

for (int i = 1; i < numberOfProcess; i++)

{

finishTime[i] = bt[i] + finishTime[i - 1];

turnAroundTime[i] = finishTime[i] - at[i];

waitingTime[i] = turnAroundTime[i] - bt[i];

}

float sum = 0;

for (int n : waitingTime)

{

sum += n;

}

float averageWaitingTime = sum / numberOfProcess;

sum = 0;

for (int n : turnAroundTime)

{

sum += n;

}

float averageTurnAroundTime = sum / numberOfProcess;

//print on console the order of processes along with their finish time & turn around time

System.out.println("Priority Scheduling Algorithm : ");

System.out.format("%20s%20s%20s%20s%20s%20s%20s\n", "ProcessId", "BurstTime", "ArrivalTime", "Priority", "FinishTime", "WaitingTime", "TurnAroundTime");

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

System.out.format("%20s%20d%20d%20d%20d%20d%20d\n", pid[i], bt[i], at[i], prt[i], finishTime[i], waitingTime[i], turnAroundTime[i]);

}

System.out.format("%100s%20f%20f\n", "Average", averageWaitingTime, averageTurnAroundTime);

}

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

Priority obj = new Priority();

obj.getProcessData(input);

obj.priorityNonPreemptiveAlgorithm();

}

}

d) Round Robin (Preemptive) =>

/\* Round Robin (Preemptive) \*/

package B2;

import java.util.\*;

public class RoundRobin{

private static Scanner inp = new Scanner(System.in);

//Driver Code

public static void main(String[] args){

int n,tq, timer = 0, maxProccessIndex = 0;

float avgWait = 0, avgTT = 0;

System.out.print("\nEnter the time quanta : ");

tq = inp.nextInt();

System.out.print("\nEnter the number of processess : ");

n = inp.nextInt();

int arrival[] = new int[n];

int burst[] = new int[n];

int wait[] = new int[n];

int turn[] = new int[n];

int queue[] = new int[n];

int temp\_burst[] = new int[n];

boolean complete[] = new boolean[n];

System.out.print("\nEnter the arrival time of the processess : ");

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

arrival[i] = inp.nextInt();

System.out.print("\nEnter the burst time of the processess : ");

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

burst[i] = inp.nextInt();

temp\_burst[i] = burst[i];

}

for(int i = 0; i < n; i++){ //Initializing the queue and complete array

complete[i] = false;

queue[i] = 0;

}

while(timer < arrival[0]) //Incrementing Timer until the first process arrives

timer++;

queue[0] = 1;

while(true){

boolean flag = true;

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

if(temp\_burst[i] != 0){

flag = false;

break;

}

}

if(flag)

break;

for(int i = 0; (i < n) && (queue[i] != 0); i++){

int ctr = 0;

while((ctr < tq) && (temp\_burst[queue[0]-1] > 0)){

temp\_burst[queue[0]-1] -= 1;

timer += 1;

ctr++;

//Updating the ready queue until all the processes arrive

checkNewArrival(timer, arrival, n, maxProccessIndex, queue);

}

if((temp\_burst[queue[0]-1] == 0) && (complete[queue[0]-1] == false)){

turn[queue[0]-1] = timer; //turn currently stores exit times

complete[queue[0]-1] = true;

}

//checks whether or not CPU is idle

boolean idle = true;

if(queue[n-1] == 0){

for(int k = 0; k < n && queue[k] != 0; k++){

if(complete[queue[k]-1] == false){

idle = false;

}

}

}

else

idle = false;

if(idle){

timer++;

checkNewArrival(timer, arrival, n, maxProccessIndex, queue);

}

//Maintaining the entries of processes after each preemption in the ready Queue

queueMaintainence(queue,n);

}

}

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

turn[i] = turn[i] - arrival[i];

wait[i] = turn[i] - burst[i];

}

System.out.print("\nProgram No.\tArrival Time\tBurst Time\tWait Time\tTurnAround Time"

+ "\n");

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

System.out.print(i+1+"\t\t"+arrival[i]+"\t\t"+burst[i]

+"\t\t"+wait[i]+"\t\t"+turn[i]+ "\n");

}

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

avgWait += wait[i];

avgTT += turn[i];

}

System.out.print("\nAverage wait time : "+(avgWait/n)

+"\nAverage Turn Around Time : "+(avgTT/n));

}

public static void queueUpdation(int queue[],int timer,int arrival[],int n, int maxProccessIndex){

int zeroIndex = -1;

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

if(queue[i] == 0){

zeroIndex = i;

break;

}

}

if(zeroIndex == -1)

return;

queue[zeroIndex] = maxProccessIndex + 1;

}

public static void checkNewArrival(int timer, int arrival[], int n, int maxProccessIndex,int queue[]){

if(timer <= arrival[n-1]){

boolean newArrival = false;

for(int j = (maxProccessIndex+1); j < n; j++){

if(arrival[j] <= timer){

if(maxProccessIndex < j){

maxProccessIndex = j;

newArrival = true;

}

}

}

if(newArrival) //adds the index of the arriving process(if any)

queueUpdation(queue,timer,arrival,n, maxProccessIndex);

}

}

public static void queueMaintainence(int queue[], int n){

for(int i = 0; (i < n-1) && (queue[i+1] != 0) ; i++){

int temp = queue[i];

queue[i] = queue[i+1];

queue[i+1] = temp;

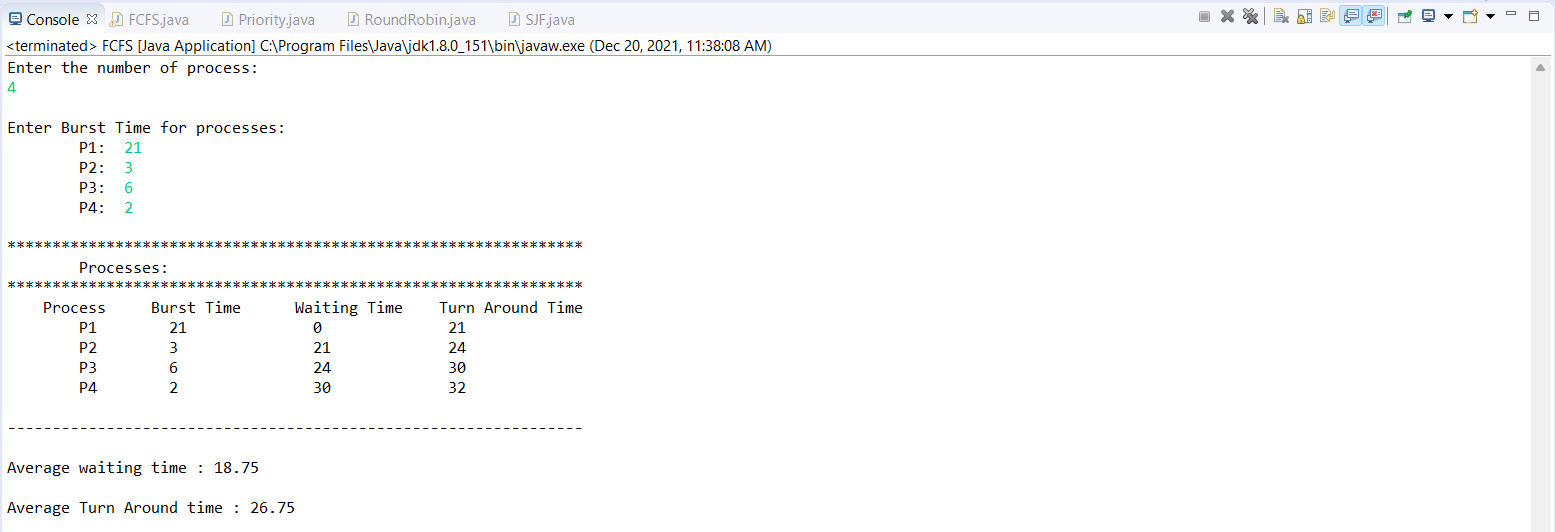
}

}

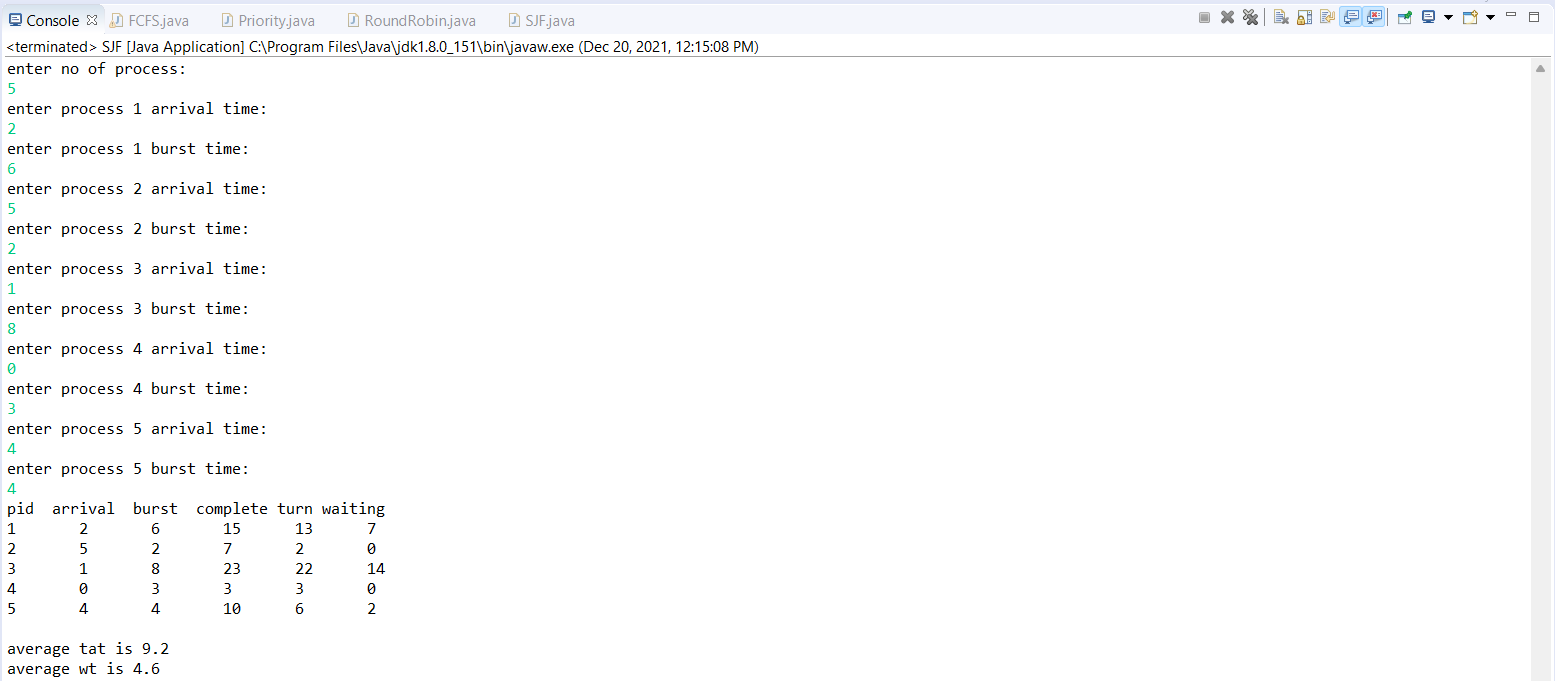
}

Output :

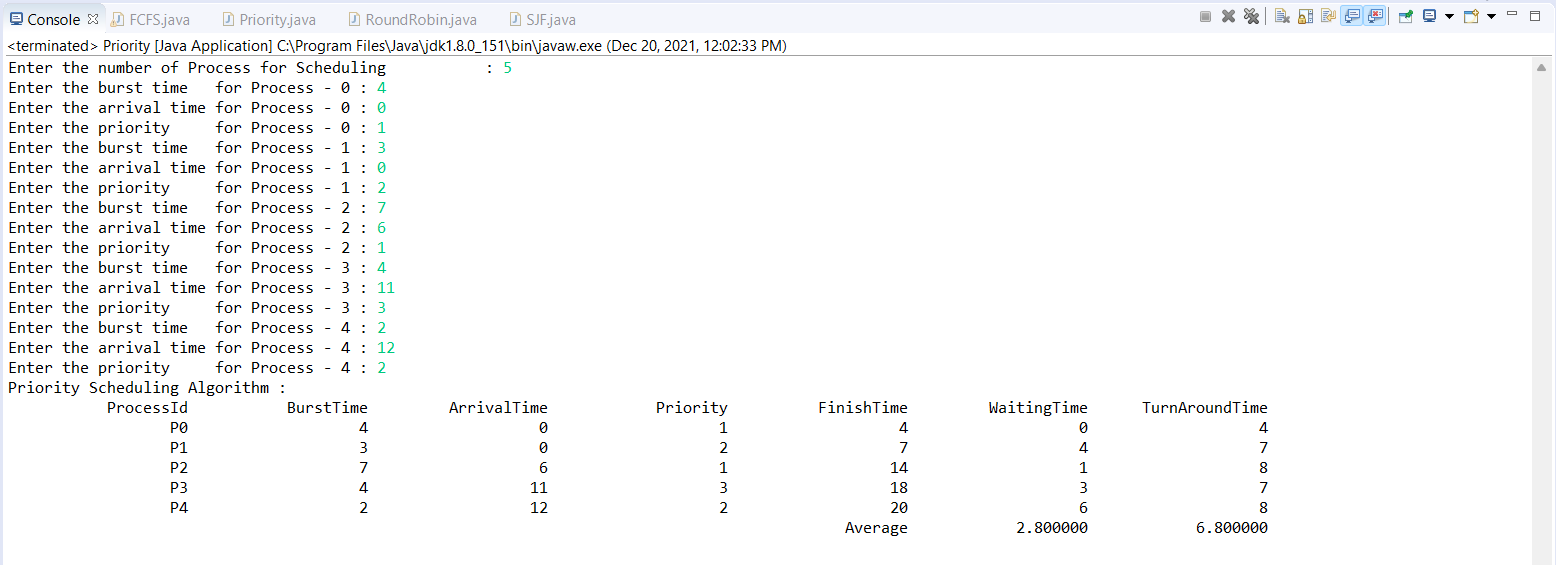
a) FCFS =>



b) SJF (Preemptive) =>



c) Priority (Non-Preemptive) =>



d) Round Robin (Preemptive) =>

