**OUTPUT**

Enter number of processes:

5

Process 1

Ënter arrival time, burst time, priority

0 9 1

Process 2

Ënter arrival time, burst time, priority

2 6 3

Process 3

Ënter arrival time, burst time, priority

4 5 4

Process 4

Ënter arrival time, burst time, priority

5 3 5

Process 5

Ënter arrival time, burst time, priority

8 7 2

ID Arrival Burst Priority Start End TAT WT RT

1 0 9 1 0 0 0 0 0.0

2 2 6 3 0 0 0 0 0.0

3 4 5 4 0 0 0 0 0.0

4 5 3 5 0 0 0 0 0.0

5 8 7 2 0 0 0 0 0.0

At time t = 0, Process 1 is now in running state

ID Burst Time Remaining Priority(E) Hybrid Priority

**1 9 9 1 1.0**

At time t = 2, Process 2 is now in running state

ID Burst Time Remaining Priority(E) Hybrid Priority

1 9 7 1 1.0

**2 6 6 2 1.5**

At time t = 8, Process 4 is now in running state

ID Burst Time Remaining Priority(E) Hybrid Priority

1 9 7 1 1.4285715

5 7 7 1 1.0

3 5 5 2 1.9

**4 3 3 3 2.5**

At time t = 11, Process 3 is now in running state

ID Burst Time Remaining Priority(E) Hybrid Priority

1 9 7 1 1.6428572

5 7 7 1 1.2142857

**3 5 5 2 2.2**

At time t = 16, Process 1 is now in running state

ID Burst Time Remaining Priority(E) Hybrid Priority

**1 9 7 1 2.0**

5 7 7 1 1.5714285

At time t = 23, Process 5 is now in running state

ID Burst Time Remaining Priority(E) Hybrid Priority

**5 7 7 1 2.0714285**

**id Arrival Burst Priority Start End TAT WT RT**

1 0 9 1 0 23 23 14 2.5555556

2 2 6 3 2 8 6 0 1.0

3 4 5 4 11 16 12 7 2.4

4 5 3 5 8 11 6 3 2.0

5 8 7 2 23 30 22 15 3.142857

**A.W.T.: 7.8**

**A.T.A.T.: 13.8**

**A.R.T.: 2.219682550430298**

**CODE**:

import java.lang.\*;

import java.util.\*;

class hrrn{

//HELPER FUNCTIONS

static void sortByArrival(Process ob[], int n){//Sort according to arrival time

int i, j;

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

for(j = 0; j < n - 1-i; j++){

if(ob[j].arrival\_time > ob[j+1].arrival\_time){

Process temp = ob[j];

ob[j] = ob[j+1];

ob[j+1] = temp;

}

}

}

}

static void sortByRemainingTime(Process ob[], int n){//Sort according //to arrival time

int i, j;

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

for(j = 0; j < n - 1-i; j++){

if(ob[j].remaining\_time < ob[j+1].remaining\_time){

Process temp = ob[j];

ob[j] = ob[j+1];

ob[j+1] = temp;

}

}

}

}

static boolean readyQueueIsNotEmpty(Process ob[], int n){

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

if(!ob[i].hasFinished){

return true;

}

}

return false;

}

static void display(Process ob[], int n){ System.*out*.println("\nid\tArrival\tBurst\tPriority\tStart\tEnd\tTAT\tWT\tRT");

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

System.*out*.println(ob[i].id+"\t"+ob[i].arrival\_time+"\t"+ob[i].burst\_time+"\t"+ob[i].original\_priority+"\t\t"+ob[i].start\_time+"\t"+ob[i].end\_time+"\t"+ob[i].turnaround\_time+"\t"+ob[i].wait\_time+"\t"+ob[i].response\_time);

}

}

//<-----HRRN WITH PRIORITY & PREEMPTION----->

static void PMHRRN(Process ob[], int n)

{

double awt = 0.0, atat = 0.0, art = 0.0;

int time = 0, previous=-1, current = -1;

while(*readyQueueIsNotEmpty*(ob, n))

{

*updateReadyQueue*(ob, n, time);

*assignPriority*(ob,n);

current = *findMaxPriority*(ob, n);

if(ob[current].hasArrived == false)

{

ob[current].hasArrived = true;

ob[current].start\_time = time;

}

if(ob[current].id != previous){//For display purposes, otherwise per unit Process ID will be displayed.

System.*out*.println("\nAt time t = "+time+", Process "+ob[current].id+" is now in running state");

*displayMidExecution*(ob, n);

}

ob[current].remaining\_time--;

time++;

*updateWaitingTime*(ob, n, current);

if(ob[current].remaining\_time == 0)

{

ob[current].hasFinished = true;

ob[current].end\_time = time;

ob[current].turnaround\_time = ob[current].end\_time - ob[current].arrival\_time;

ob[current].response\_time = (((float)ob[current].wait\_time+(float)ob[current].burst\_time)/(float)ob[current].burst\_time);

atat += ob[current].turnaround\_time;

awt += ob[current].wait\_time;

art += ob[current].response\_time;

}

previous = ob[current].id;

}

*sortByArrival*(ob, n);

*display*(ob, n);

awt /= n; atat /= n; art /= n;

System.*out*.println("\nA.W.T.: "+awt+"\nA.T.A.T.: "+atat+"\nA.R.T.: "+art);

}

private static void displayMidExecution(Process[] ob, int n) {

int i;

System.*out*.println("ID\tBurst Time\tRemaining\tPriority(E)\tHybrid Priority");

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

{

if(ob[i].isReady && !ob[i].hasFinished)

{

System.*out*.println(ob[i].id+"\t"+ob[i].burst\_time+"\t\t"+ob[i].remaining\_time+"\t\t"+ob[i].priority+"\t\t"+ob[i].hybrid\_priority);

}

}

}

private static void updateReadyQueue(Process[] ob, int n, int time) {

int i;

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

if(ob[i].arrival\_time <= time)

ob[i].isReady = true;

}

private static int findMaxPriority(Process[] ob, int n) {

int i,pos = -1;;float max = -1;

float p = 0.5f;

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

{

if(ob[i].isReady && !ob[i].hasFinished){

ob[i].response\_time = (((float)ob[i].wait\_time+(float)ob[i].remaining\_time)/(float)ob[i].remaining\_time);

ob[i].hybrid\_priority = ((float)ob[i].response\_time \* p) + ((1-p)\*(float)ob[i].priority);

if(ob[i].hybrid\_priority > max){

if(ob[i].hybrid\_priority == max){

if(ob[i].arrival\_time > ob[pos].arrival\_time)

continue;

}

pos = i;

max = ob[i].hybrid\_priority;

}

}

}

return pos;

}

private static void updateWaitingTime(Process[] ob, int n, int previous ) {

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

{

if (i == previous) continue;

if(ob[i].isReady && !ob[i].hasFinished)

ob[i].wait\_time++;

}

}

private static void assignPriority(Process[] ob, int n) {

int i, count = 0, previous = -1;

*sortByRemainingTime*(ob, n);

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

if(ob[i].isReady && !ob[i].hasFinished){

if(ob[i].remaining\_time == previous)

ob[i].priority = count;

else

ob[i].priority = ++count;

previous = ob[i].remaining\_time;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

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

int n = sc.nextInt();

Process p[] = new Process[n];

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

System.*out*.println("Process "+(i+1));

System.*out*.println("Ënter arrival time, burst time, priority ");

int arr = sc.nextInt();

int bur = sc.nextInt();

int pri = sc.nextInt();

p[i] = new Process(i+1, arr, bur, pri);

}

*display*(p, n);

*PMHRRN*(p, n);

sc.close();

}

}

class Process{

int id, priority, original\_priority,arrival\_time, wait\_time, burst\_time, start\_time, end\_time, turnaround\_time, remaining\_time;

float hybrid\_priority, response\_time;

boolean isReady, hasFinished, hasArrived;

//Flag = 1 when Remaining time = 0 i.e. process has finished executing

Process(int id, int a,int b, int p){

this.id = id;

this.start\_time = 0;

this.arrival\_time = a;

this.burst\_time = b;

this.original\_priority = p;

this.remaining\_time = b;

this.wait\_time = 0;

this.hasArrived = false;

this.isReady = false;

this.hasFinished = false;

}

}