#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

struct Process

{

char pname[10];

int arr\_time;

int burst\_time;

int priority;

int wt,tat;

};

void Input\_Process(struct Process\* P ,int n,bool priority);

void SRTF(struct Process\* P ,int n);

void Output\_Process(struct Process\* P ,int n);

int main()

{

bool flag=false;

int n,choice;

printf("Enter number of processes:");

scanf("%d",&n);

struct Process\* P = (struct Process\*)malloc(n\*sizeof(struct Process));

SRTF(P,n);

Output\_Process(P,n);

}

void Input\_Process(struct Process\* P ,int n,bool priority)

{

int i,j;

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

{

fflush(stdin);

printf("\nEnter %d process name:",i+1);

scanf("%s",P[i].pname);

printf("Enter %d arrival time:",i+1);

scanf("%d",&P[i].arr\_time);

printf("Enter %d burst time:",i+1);

scanf("%d",&P[i].burst\_time);

if(priority)

{

printf("Enter %d priority:",i+1);

scanf("%d",&P[i].priority);

}

else

P[i].priority = 1;

}

//to sort by arrival time

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

{

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

{

if(P[j].arr\_time > P[j+1].arr\_time)

{

struct Process temp = P[j];

P[j] = P[j+1];

P[j+1]=temp;

}

}

}

}

void SRTF(struct Process\* P ,int n)

{

Input\_Process(P,n,false);

//to sort by shortest burst time

int i,j;

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

{

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

{

if(P[j].burst\_time > P[j+1].burst\_time && P[j].arr\_time == P[j+1].arr\_time)

{

struct Process temp = P[j];

P[j] = P[j+1];

P[j+1]=temp;

}

}

}

// R - array of remaining time of Processes.

int\* R = (int\*)malloc(n\*sizeof(int));

// Copy the burst time into R[]

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

R[i] = P[i].burst\_time;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

// Process until all processes gets completed

while (complete != n)

{

// Find process with minimum remaining time among the processes that arrives till the current time.

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

{

if ((P[j].arr\_time <= t) &&

(R[j] < minm) && R[j] > 0) {

minm = R[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

R[shortest]--;

// Update minimum

minm = R[shortest];

if (minm == 0)

minm = INT\_MAX;

// If a process gets completely executed

if (R[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current process

finish\_time = t + 1;

// Calculate waiting time

P[shortest].wt = finish\_time - P[shortest].burst\_time - P[shortest].arr\_time;

if (P[shortest].wt < 0)

P[shortest].wt = 0;

}

// Increment time

t++;

}

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

P[i].tat = P[i].wt + P[i].burst\_time;

}

void Output\_Process(struct Process\* P ,int n)

{

int i;

printf("\nNAME\t\tARRIVAL\_TIME\t\tBURST\_TIME\t\tPRIORITY\tWAITING\_TIME\tTURN\_AROUND\_TIME\n");

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

printf("%s\t\t%d\t\t\t%d\t\t\t%d\t\t%d\t\t%d\n",P[i].pname,P[i].arr\_time,P[i].burst\_time,P[i].priority,P[i].wt,P[i].tat);

double wt=0,tat=0;

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

{

wt += P[i].wt;

tat += P[i].tat;

}

wt /= n;

tat /= n;

printf("\nAverage:\nWaiting Time = %f\nTurn Around Time = %f",wt,tat);

}

STEP 1- Traverse until all process gets completely executed.

           a) Find process with minimum remaining time at every single timed lap.

           b) Reduce its time by 1.

           c) Check if its remaining time becomes 0

           d) Increment the counter of process completion.

           e) Completion time of current process = current\_time +1;

           f) Calculate the waiting time for each completed process.

                wt[i]= Completion time - arrival\_time-burst\_time

           g)Increment time lap by one.

STEP 2- Find turnaround time (waiting\_time+burst\_time).

To implement scheduling algorithm SRTF (Shortest Remaining Time First) also called Pre-emptive SJF (Shortest Job First). In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

This algorithm is to implement using C language by Entering no. of Processes and their arrival and burst time.