Write the program to simulate Round Robin (RR) scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Also give the time quantum as input. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

char pname[10];

int start,end;

}g[30],g1[30];

int n,time,prev,k,tq;

void getinput()

{

int i;

printf("\nEnter No of Process: ");

scanf("%d",&n);

printf("\nEnter Time quantum: ");

scanf("%d",&tq);

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

{

printf("\nEnter Process Name: ");

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

printf("Arrival Time:");

scanf("%d",&tab[i].at);

printf("Burst Time: ");

scanf("%d",&tab[i].bt);

tab[i].tbt=tab[i].bt;

}

}

void printinput()

{

// int TWT=0,TTAT=0;

int i;

printf("\nPname\tAT\tBT");

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

printf("\n%s\t%d\t%d",tab[i].pname,tab[i].at,tab[i].tbt);

}

void printoutput()

{

int TWT=0,TTAT=0,i;

float ATAT,AWT;

printf("\nPname\tAT\tBT\tFT\tWT\tTAT");

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].ft-tab[i].at-tab[i].bt,tab[i].ft-tab[i].at);

TWT=TWT+(tab[i].ft-tab[i].at-tab[i].bt);

TTAT=TTAT+(tab[i].ft-tab[i].at);

}

printf("\nTotal WT: %d",TWT);

printf("\nTotal TAT:%d",TTAT);

AWT=(float)TWT/n;

ATAT=(float)TTAT/n;

printf("\nAverage WT: %f",AWT);

printf("\nAverage TAT:%f",ATAT);

}

void sort()

{

int pass,i;

struct input temp;

for(pass=1;pass<n;pass++)

{

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

{

if(tab[i].at>tab[i+1].at)

{

temp=tab[i];

tab[i]=tab[i+1];

tab[i+1]=temp;

}

}

}

}

int arrived(int time)

{

int i;

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

{

if(tab[i].at<=time && tab[i].tbt!=0)

return 1;

}

return 0;

}

void processinput()

{

int finish=0,j;

int i=0;

k=0;

while(finish!=n)

{

if(arrived(time))

{

if(tab[i].tbt!=0)

{

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

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

prev=time;

strcpy(g[k++].pname,tab[i].pname);

tab[i].ft=time;

if(tab[i].tbt==0)

{

finish++;

break;

}

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

strcpy(g[k++].pname,"idle");

prev=time;

}

if(time<tab[(i+1)%n].at)

i=0;

else

i=(i+1)%n;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

printf("\nStart\tpname\tEnd");

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

{

printf("\n%d\t%s\t%d",g[i].start,g[i].pname,g[i].end);

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

g1[0]=g[0];

for(i=1;i<k;i++)

{

if(strcmp(g[i].pname,g1[j].pname)==0)

g1[j].end=g[i].end;

else

{

j++;

g1[j]=g[i];

}

}

printf("\nStart\tpname\tEnd");

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

{

printf("\n%d\t%s\t%d",g1[i].start,g1[i].pname,g1[i].end);

}

}

int main()

{

getinput();

printinput();

sort();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}