#include<stdio.h>struct process

{

intprocessID; intburstTime; intarrivalTime; int priority; intwaitTime;

};

inttotal\_time,burst\_time=0; //to calculate whole complete completion time and burst time int total=-1,i=-1; //to calcualte no of process entered in cpu and i to put result in cpustruct process queue[100],result[100],swap;

//function to ask number of process to be created by user intprocess\_create()

{

int n;

printf("enter the number of process you want to enter:");

scanf("%d",&n); return n;

}

//function to execute the process and finish it void execute()

{

if(total>=0)

{

intwait,j;

//to increase the priority and decrease the burst time of priority in excecutionif(burst\_time!=0 && queue[0].burstTime!=0)

{

queue[0].burstTime--; burst\_time--;

queue[0].priority++;

queue[0].arrivalTime=total\_time+1;

total\_time++;

//to increase the wait and priority of waiting process

for(wait=1;wait<=total;wait++)

{

queue[wait].priority+=2;

queue[wait].waitTime=++queue[wait].waitTime;

}

}

//if process gets completed ,it is put in result queue

if(queue[0].burstTime==0)

{

i++;

result[i]=queue[0]; for(wait=0;wait<total;wait++)

{

queue[wait]=queue[wait+1];

}

total--;

}

//to sort the process again in by priority for(wait=0;wait<total;wait++)

{

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

{

if(queue[wait].priority<=queue[j].priority)

{

swap=queue[wait]; queue[wait]=queue[j]; queue[j]=swap;

}

}

}

if(queue[0].priority<=queue[1].priority && total>=1)

{

swap=queue[0];

for(wait=0;wait<total;wait++)

{

queue[wait]=queue[wait+1];

}

queue[total]=swap;

}

}

}

/\*to give the process id,bursttime,arrival time

and sort the array according to arrival time and if arrival time is equal then sort according to burst time in main\*/ void main()

{

intl,j,n=process\_create(),count=0; float averageWaitTime=0; struct process pcreate[n]; for(l=0;l<n;l++)

{

pcreate[l].processID=l+1;

printf("\nEnter the arrival time of process[%d]: ",l+1); scanf("%d",&pcreate[l].arrivalTime); printf("\nEnter the burst time of process[%d]: ",l+1); scanf("%d",&pcreate[l].burstTime); pcreate[l].priority=0; pcreate[l].waitTime=0;

burst\_time=burst\_time+pcreate[l].burstTime;

}

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

{

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

{

if(pcreate[l].arrivalTime<pcreate[j].arrivalTime)

{

swap=pcreate[l]; pcreate[l]=pcreate[j]; pcreate[j]=swap;

}

if(pcreate[l].arrivalTime==pcreate[j].arrivalTime)

{

if(pcreate[l].burstTime<=pcreate[j].burstTime)

{

swap=pcreate[l]; pcreate[l]=pcreate[j]; pcreate[j]=swap;

}

}

}

}// printing the sorted process id with respect to arrival time and if arrival time is equal than burst time.

printf("VALUES ENTERED:\n\*(TABLE SORTED ACCORDING TO THE AARIVAL TIME)\n\n"); printf(" PROCESS TABLE \n");

printf("\n.............................................\n"); printf(" PROCESS ID ARRIVAL TIME SERVICE TIME \n");

printf("\n.............................................\n");

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

{

printf(" %d %d %d\n",pcreate[l].processID,pcreate[l].arrivalTime,pcreate[l].burstTime );

}

total\_time=pcreate[0].arrivalTime; for(j=pcreate[0].arrivalTime;j<=pcreate[n-1].arrivalTime;j++)

{

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

{

if(pcreate[l].arrivalTime==j && count!=n)

{

total++;

queue[total]=pcreate[l];

count++;

}

if(count==n)

break;

}

execute(); total\_time++;

while(burst\_time!=0 && count==n)

{

execute(); total\_time++;

}

if(count==n)

break;

}

printf("PROCESS IN ORDER OF THEIR COMPLETION:\n\n");

printf(" FINAL PROCESS EXECUTION TABLE \n");

printf("................................................................................\n");

printf(" PROCESS ID ARRIVAL TIME SERVICE TIME WAITING TIME\n"); printf("................................................................................\n");

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

{

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

{

if(result[l].processID==pcreate[j].processID)

{

printf(" %d %d %d

%d\n",result[l].processID,pcreate[j].arrivalTime,pcreate[j].burstTime,result[l].waitTime);

break;

}

}

averageWaitTime+=(result[l].waitTime);

}

printf("AVERAGE WAITING TIME :%f\n",averageWaitTime/n);

}