***OS EXP 4:Schedulling Algorithms***

***1)First Come First Serve Without Arrival***

#include<stdio.h>

///Non-Arrival First Come First Serve

///Adnan Ismail Shah Muzavor

#include<stdlib.h>

/// Sorting key will help in sorting

/// when we want to sort values based on some

/// some other aspect

typedef struct array

{

int b\_time; ///burst\_time

int s\_key; ///sorting key eg priority

} Prc;

///To find average

float avg(int \*arr,int n)

{

float avg=0.0;

int i=0;

for(i=0; i<n; i++) avg+=(float)arr[i];

return (avg/(float)n);

}

///To display the result

void display(int \*wt,int \*tt,int n)

{

int i;

printf("Process\t\tWaiting time\t\tTurnaround time\n");

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

{

printf(" p%d \t\t%d\t\t\t%d\n",i+1,wt[i],tt[i]);

}

printf(" avg \t\t%.2f\t\t\t%.2f\n",avg(wt,n),avg(tt,n));

}

void FCFS(Prc p1[],int n)

{

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

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

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

int sum=0,i,gt\_itr=0;

for(i=0;i<n;i++) gt[i]=0;

wt[0]=0; //Initially waiting time is zero for first process in p1

gt[gt\_itr]=0;

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

{

///Calculate waiting time

if(i>0)

{

/// for i=0, we already know waiting time is zero

/// otherwise it is burst time of prev process

sum=sum+p1[i-1].b\_time; /// Burst time of ith process

wt[i]=sum;

}

///Calculate turnround time

tt[i]=p1[i].b\_time+sum;

gt[gt\_itr]=sum;

gt\_itr++;

}

gt[gt\_itr]=sum+p1[n-1].b\_time;

gt\_itr++;

display(wt,tt,n);

ganttchart(p1,gt,n,gt\_itr);

}

void ganttchart(Prc \*p,int\*gt,int n,int n2){

int i;

printf("\nGantt chart is: \n");

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

printf("------");

}

printf("\n");

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

printf(" p%d |",i+1);

}

printf("\n");

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

printf("------");

}

printf("\n");

for(i=0; i<n2; i++)

{

printf("%d ",gt[i]);

}

}

int main()

{

int n,i;

printf("Enter the number of processes: ");

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n);

printf("\nEnter the burst time for: \n");

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

{

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

scanf("%d",&p1[i].b\_time);

p1[i].s\_key=-1;

//printf("\n");

}

printf("Process\t\tBurst time\n");

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

{

printf(" p%d \t\t%d\n",i+1,p1[i].b\_time);

}

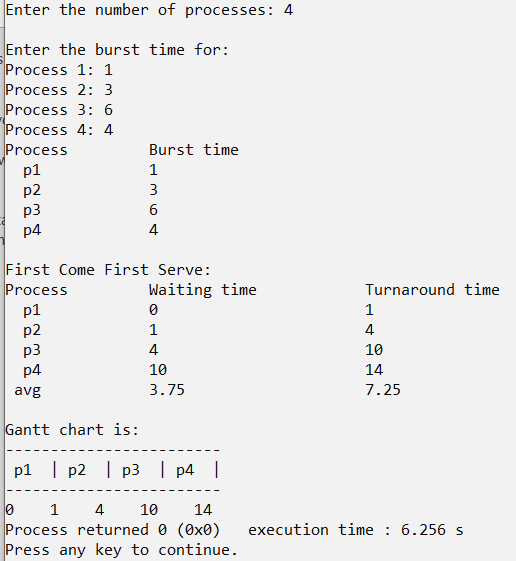
printf("\nFirst Come First Serve:\n");

FCFS(p1,n);

return 0;

}

**OUTPUT:**

******

***2)Shortest Job First Without Arrival***

#include<stdio.h>

///Non-Arrival Shortest job first

///Adnan Ismail Shah Muzavor

#include<stdlib.h>

/// Sorting key will help in sorting

/// when we want to sort values based on some

/// some other aspect

typedef struct array

{

int b\_time; ///burst\_time

int s\_key; ///sorting key eg priority

} Prc;

///In THIS ALGO WE NEED TO SORT NUMBERS

///I.E BURST TIME FROM SMALL TO LARGE SO THAT

///SHORTST PROCESS EXECUTES FIRST

void merge(int l,int m,int h,Prc p1[],int order[])

{

///id s\_key is one use sort key else sort directly

int temp[h-l+1];

int new\_order[h-l+1];

int i=l,j=m+1,k=0;

while(i<=m && j<=h)

{

///if sorting based on B\_time directly s\_key=0

if(p1[i].b\_time<=p1[j].b\_time)

{

temp[k]=p1[i].b\_time;

new\_order[k]=order[i];

i++;

k++;

}

else

{

temp[k]=p1[j].b\_time;

new\_order[k]=order[j];

j++;

k++;

}

}

///Add remaining elemnts

while(i<=m)

{

temp[k]=p1[i].b\_time;

new\_order[k]=order[i];

k++;

i++;

}

///Add remaining elemnts

while(j<=h)

{

temp[k]=p1[j].b\_time;

new\_order[k]=order[j];

k++;

j++;

}

///Modify original array

i=l,k=0;

while(i<=h)

{

p1[i].b\_time=temp[k];

order[i]=new\_order[k];

i++;

k++;

}

}

void mergesort(int i,int j,Prc p1[],int order[])

{

if(i<j)

{

int m=(i+j)/2;

mergesort(i,m,p1,order);

mergesort(m+1,j,p1,order);

merge(i,m,j,p1,order);

}

}

///To find average

float avg(int \*arr,int n)

{

float avg=0.0;

int i=0;

for(i=0; i<n; i++) avg+=(float)arr[i];

return (avg/(float)n);

}

///To display the result

void display(int \*order,int \*wt,int \*tt,int n)

{

int i;

printf("Process\t\tWaiting time\t\tTurnaround time\n");

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

{

printf(" p%d \t\t%d\t\t\t%d\n",order[i]+1,wt[i],tt[i]);

}

printf(" avg \t\t%.2f\t\t\t%.2f\n",avg(wt,n),avg(tt,n));

}

void SJF(Prc p1[],int \*order,int n)

{

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

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

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

int sum=0,i,gt\_itr=0;

for(i=0;i<n;i++) gt[i]=0;

wt[0]=0; //Initially waiting time is zero for first process in p1

gt[gt\_itr]=0;

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

{

///Calculate waiting time

if(i>0)

{

/// for i=0, we already know waiting time is zero

/// otherwise it is burst time of prev process

sum=sum+p1[i-1].b\_time; /// Burst time of ith process

wt[i]=sum;

}

///Calculate turnround time

tt[i]=p1[i].b\_time+sum;

gt[gt\_itr]=sum;

gt\_itr++;

}

gt[gt\_itr]=sum+p1[n-1].b\_time;

gt\_itr++;

display(order,wt,tt,n);

ganttchart(p1,order,gt,n,gt\_itr);

}

void ganttchart(Prc \*p,int\*order,int\*gt,int n,int n2){

int i;

printf("\nGantt chart is: \n");

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

printf("-----");

}

printf("\n");

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

printf(" p%d |",order[i]+1);

}

printf("\n");

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

printf("-----");

}

printf("\n");

for(i=0; i<n2; i++)

{

printf("%d ",gt[i]);

}

}

int main()

{

int n,i;

printf("Enter the number of processes: ");

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n); ///Storing values of the array

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

for(i=0;i<n;i++) pr\_order[i]=i;

printf("\nEnter the burst time for: \n");

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

{

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

scanf("%d",&p1[i].b\_time);

p1[i].s\_key=-1;

}

printf("Process\t\tBurst time\n");

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

{

printf(" p%d \t\t%d\n",pr\_order[i]+1,p1[i].b\_time);

}

mergesort(0,n-1,p1,pr\_order); ///Sort based on burst time only

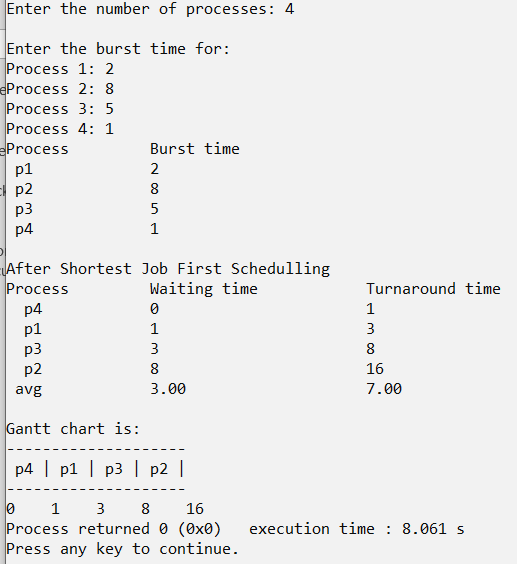
printf("\nAfter Shortest Job First Schedulling\n");

SJF(p1,pr\_order,n);

return 0;

}

**Output:**

****

***3)Non Preemtive Priority Without Arrival***

#include<stdio.h>

///Non-Arrival Non-preemtive priority

///Adnan Ismail Shah Muzavor

#include<stdlib.h>

///Sorting key will help in sorting

/// when we want to sort values based on some

/// some other aspect

typedef struct array

{

int b\_time; ///burst\_time

int s\_key; ///sorting key i.e priority

} Prc;

///In THIS ALGO WE NEED TO SORT NUMBERS

///I.E BURST TIME FROM SMALL TO LARGE SO THAT

///SHORTST PROCESS EXECUTES FIRST

void merge(int l,int m,int h,Prc p1[],int order[])

{

///id s\_key is one use sort key else sort directly

int temp[h-l+1];

int new\_order[h-l+1];

int prty[h-l+1];

int i=l,j=m+1,k=0;

while(i<=m && j<=h)

{

if(p1[i].s\_key<=p1[j].s\_key)

{

temp[k]=p1[i].b\_time;

new\_order[k]=order[i];

prty[k]=p1[i].s\_key;

i++;

k++;

}

else

{

temp[k]=p1[j].b\_time;

prty[k]=p1[j].s\_key;

new\_order[k]=order[j];

j++;

k++;

}

}

///Add remaining elemnts

while(i<=m)

{

temp[k]=p1[i].b\_time;

prty[k]=p1[i].s\_key;

new\_order[k]=order[i];

k++;

i++;

}

///Add remaining elemnts

while(j<=h)

{

temp[k]=p1[j].b\_time;

prty[k]=p1[j].s\_key;

new\_order[k]=order[j];

k++;

j++;

}

///Modify original array

i=l,k=0;

while(i<=h)

{

p1[i].b\_time=temp[k];

order[i]=new\_order[k];

p1[i].s\_key=prty[k];

i++;

k++;

}

}

void mergesort(int i,int j,Prc p1[],int order[])

{

if(i<j)

{

int m=(i+j)/2;

mergesort(i,m,p1,order);

mergesort(m+1,j,p1,order);

merge(i,m,j,p1,order);

}

}

///To find average

///To find average

float avg(int \*arr,int n)

{

float avg=0.0;

int i=0;

for(i=0; i<n; i++) avg+=(float)arr[i];

return (avg/(float)n);

}

///To display the result

void display(int \*order,int \*wt,int \*tt,int n)

{

int i;

printf("Process\t\tWaiting time\t\tTurnaround time\n");

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

{

printf(" p%d \t\t%d\t\t\t%d\n",order[i]+1,wt[i],tt[i]);

}

printf(" avg \t\t%.2f\t\t\t%.2f\n",avg(wt,n),avg(tt,n));

}

void NPP(Prc p1[],int\*order,int n)

{

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

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

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

int sum=0,i,gt\_itr=0;

wt[0]=0; //Initially waiting time is zero for first process in p1

gt[gt\_itr]=0;

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

{

///Calculate waiting time

if(i>0)

{

/// for i=0, we already know waiting time is zero

/// otherwise it is burst time of prev process

sum=sum+p1[i-1].b\_time; /// Burst time of ith process

wt[i]=sum;

}

///Calculate turnround time

tt[i]=p1[i].b\_time+sum;

gt[gt\_itr]=sum;

gt\_itr++;

}

gt[gt\_itr]=sum+p1[n-1].b\_time;

gt\_itr++;

display(order,wt,tt,n);

ganttchart(p1,order,gt,n,gt\_itr);

}

void ganttchart(Prc \*p,int\*order,int\*gt,int n,int n2){

int i;

printf("\nGantt chart is: \n");

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

printf("-----");

}

printf("\n");

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

printf(" p%d |",order[i]+1);

}

printf("\n");

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

printf("-----");

}

printf("\n");

for(i=0; i<n2; i++)

{

printf("%d ",gt[i]);

}

}

int main()

{

int n,i;

printf("Enter the number of processes: ");

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n); ///Storing values of the array

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

for(i=0; i<n; i++) pr\_order[i]=i;

printf("\nEnter the burst time and priority for: \n");

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

{

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

scanf("%d%d",&p1[i].b\_time,&p1[i].s\_key);

}

printf("Process\t\tBurst time\t\tPriority\n");

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

{

printf(" p%d \t\t%d\t\t\t%d\n",pr\_order[i]+1,p1[i].b\_time,p1[i].s\_key);

}

mergesort(0,n-1,p1,pr\_order); ///Sort based on s only s\_key i.e priority

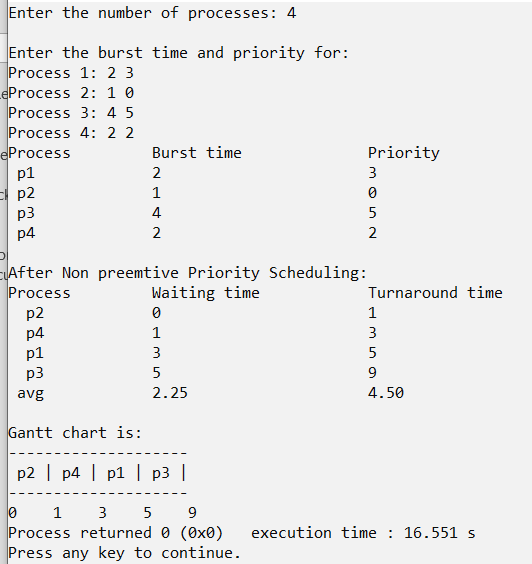
printf("\nAfter Non preemtive Priority Scheduling:\n");

NPP(p1,pr\_order,n);

return 0;

}

**Output:**

****

**4)First Come First Serve With Arrival**

#include<stdio.h>

///With Arrival First Come First Serve

///Adnan Ismail Shah Muzavor

#include<stdlib.h>

/// Sorting key will help in sorting

/// when we want to sort values based on some

/// some other aspect

typedef struct array

{

int b\_time; ///burst\_time

int a\_time; ///arrival time

int s\_key; ///sorting key eg priority

} Prc;

///To find average

float avg(int \*arr,int n)

{

float avg=0.0;

int i=0;

for(i=0; i<n; i++) avg+=(float)arr[i];

return (avg/(float)n);

}

///To display the result

void display(int \*order,int \*wt,int \*tt,int n)

{

int i;

printf("Process\t\tWaiting time\t\tTurnaround time\n");

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

{

printf(" p%d \t\t%d\t\t\t%d\n",order[i]+1,wt[i],tt[i]);

}

printf("avg \t\t%.2f\t\t\t%.2f\n",avg(wt,n),avg(tt,n));

}

void merge(int l,int m,int h,Prc p1[],int order[])

{

///id s\_key is one use sort key else sort directly

int temp[h-l+1];

int temp2[h-l+1];

int new\_order[h-l+1];

int i=l,j=m+1,k=0;

while(i<=m && j<=h)

{

if(p1[i].a\_time<=p1[j].a\_time)

{

temp[k]=p1[i].b\_time;

temp2[k]=p1[i].a\_time;

new\_order[k]=order[i];

i++;

k++;

}

else

{

temp[k]=p1[j].b\_time;

temp2[k]=p1[j].a\_time;

new\_order[k]=order[j];

j++;

k++;

}

}

///Add remaining elemnts

while(i<=m)

{

temp[k]=p1[i].b\_time;

temp2[k]=p1[i].a\_time;

new\_order[k]=order[i];

k++;

i++;

}

///Add remaining elemnts

while(j<=h)

{

temp[k]=p1[j].b\_time;

temp2[k]=p1[j].a\_time;

new\_order[k]=order[j];

k++;

j++;

}

///Modify original array

i=l,k=0;

while(i<=h)

{

p1[i].b\_time=temp[k];

p1[i].a\_time=temp2[k];

order[i]=new\_order[k];

i++;

k++;

}

}

///sort p1 based on a\_key i.e arrival time

void mergesort(int i,int j,Prc p1[],int order[])

{

if(i<j)

{

int m=(i+j)/2;

mergesort(i,m,p1,order);

mergesort(m+1,j,p1,order);

merge(i,m,j,p1,order);

}

}

///Display the input/sorted/processed input

void raw\_display(int order[],Prc p1[],int n)

{

int i;

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

{

printf(" p%d \t\t%d\t\t\t%d\n",order[i]+1,p1[i].a\_time,p1[i].b\_time);

}

}

void FCFS\_A(Prc p1[],int order[],int n)

{

int i,sum=0,gt\_itr=0;

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

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

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

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

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

{

bb[i]=0;

gt[i]=0;

}

///Before doing anything for sort processes based on arrival time

///As process which arrived first will be served first

mergesort(0,n-1,p1,order);

///On this order run first come firts serve aogorithm

wt[0]=0; ///Initially waiting time is zero for first process in p1

gt[0]=p1[0].a\_time; ///As we will be executing processes right from first process which min arrival time

///It's not necesssary that this arrival time will always be zero

sum=p1[0].a\_time; ///AHence w eare adding up this time in sum as well!!

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

{

///Calculate waiting time

if(i>0)

{

/// for i=0, we already know waiting time is zero

/// otherwise it is burst time of prev process

sum=sum+p1[i-1].b\_time; /// Burst time of ith process

wt[i]=(sum-p1[i].a\_time);

///If process arrived much later while prev process was executing

if(p1[i].a\_time>sum)

{

///Thr process doent has to wait

wt[i]=0;

bb[i]=100;

gt[gt\_itr]=sum;

gt\_itr++;

///Add to burst time and time of this blank slot

sum+=(p1[i].a\_time-sum); //Add time taken by blank slot

}

gt[gt\_itr]=sum;

}

///Calculate turnround time

tt[i]=p1[i].b\_time+wt[i];

gt\_itr++;

}

///Add burst time of last process

gt[gt\_itr]=sum+p1[n-1].b\_time;

gt\_itr++;

printf("\n");

display(order,wt,tt,n);

ganttchart(order,gt,bb,n,gt\_itr);

}

void ganttchart(int \*order,int\*gt,int\* bb,int n,int n2)

{

int i;

printf("\nGantt chart is: \n");

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

{

if(bb[i]) printf("------");

printf("------");

}

printf("\n");

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

{

if(bb[i]==0) printf(" p%d |",order[i]+1);

else printf(" --| p%d |",order[i]+1);

}

printf("\n");

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

{

if(bb[i]) printf("------");

printf("------");

}

printf("\n");

for(i=0; i<n2; i++)

{

printf("%d ",gt[i]);

}

}

int main()

{

int n,i;

printf("Enter the number of processes: ");

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n);

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

for(i=0; i<n; i++) pr\_order[i]=i;

printf("\nEnter the BURST time and ARRIVAL time for: \n");

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

{

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

scanf("%d",&p1[i].b\_time);

scanf("%d",&p1[i].a\_time);

p1[i].s\_key=-1;

}

printf("Process\t\tArrival time\t\tBurst time\n");

raw\_display(pr\_order,p1,n);

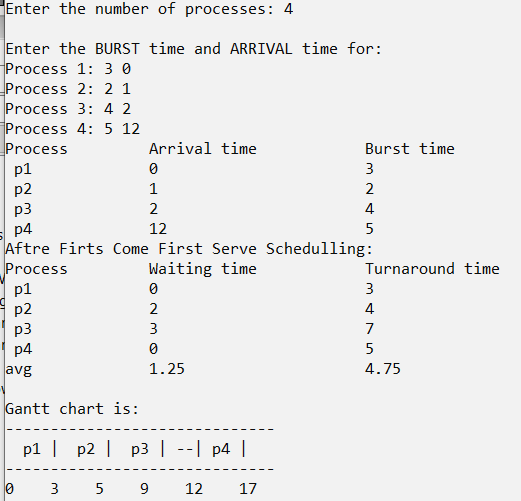
printf("Aftre Firts Come First Serve Schedulling: ");

FCFS\_A(p1,pr\_order,n);

return 0;

}

**Output:**

****

**5)Shortest Job First With Arrival**

#include<stdio.h>

///With Arrival Shortest job first

///Adnan Ismail Shah Muzavor

#include<stdlib.h>

/// Sorting key will help in sorting

/// when we want to sort values based on some

/// some other aspect

typedef struct array

{

int b\_time; ///burst\_time

int a\_time; ///arrival time

int s\_key; ///sorting key eg priority

} Prc;

///To find average

float avg(int \*arr,int n)

{

float avg=0.0;

int i=0;

for(i=0; i<n; i++) avg+=(float)arr[i];

return (avg/(float)n);

}

///To display the result

void display(int \*order,int \*wt,int \*tt,int n)

{

int i;

printf("Process\t\tWaiting time\t\tTurnaround time\n");

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

{

printf("p%d \t\t%d\t\t\t%d\n",order[i],wt[i],tt[i]);

}

printf("avg \t\t%f\t\t%.2f\n",avg(wt,n),avg(tt,n));

}

///Display the input/sorted/processed input

void raw\_display(int order[],Prc p1[],int n)

{

int i;

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

{

printf(" p%d \t\t%d\t\t\t%d\n",order[i],p1[i].a\_time,p1[i].b\_time);

}

}

int decide\_first(Prc p1[],int processed[],int n,int prev\_burst\_time)

{

///To keep track of process to be executed first

int first\_p=-1,min\_arrival\_time=1000,min\_burst\_time=1000,i=0;

///Choose the first process to execute

if(prev\_burst\_time==-1)

{

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

{

///THe proces has arrival time lesser thn minimum arrival time so far

if(p1[i].a\_time!=min\_arrival\_time && p1[i].a\_time<min\_arrival\_time)

{

min\_arrival\_time=p1[i].a\_time;

min\_burst\_time=p1[i].b\_time;

first\_p=i;

}

///If process has same minimum arrival time, thn check for burst time

///Consider the one with minimum burst time

else if(p1[i].a\_time==min\_arrival\_time && p1[i].b\_time<min\_burst\_time)

{

min\_arrival\_time=p1[i].a\_time;

min\_burst\_time=p1[i].b\_time;

first\_p=i;

}

}

}

else

{

///No process executed therefore return process that will execute first

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

{

///If this process is not already included

if(!processed[i])

{

///If proces had arrived while prev was executing

if( p1[i].a\_time<=prev\_burst\_time)

{

///If we have not selected any process as of now

if(first\_p==-1)

{

first\_p=i; ///Choose this process and thn look for other process;

min\_burst\_time=p1[i].b\_time;

continue;

}

///If this process has lesser burst time thn proces we had arrived during this within same range

///i.e while prev process was executing

if(p1[i].b\_time<min\_burst\_time)

{

min\_burst\_time=p1[i].b\_time;

first\_p=i;

}

}

}

}

}

if(first\_p==-1)

{

int min\_till\_now=1000;

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

{

if(!processed[i])

{

int min\_diff=p1[i].a\_time-prev\_burst\_time;

if(min\_diff<min\_till\_now)

{

first\_p=i;

min\_till\_now=min\_diff;

}

}

}

};

if(first\_p==-1) return -1;

processed[first\_p]=1;

return first\_p;

}

int max(int a,int b)

{

if(a>b) return a;

return b;

}

void decide\_order(Prc p1[],int order[],int n)

{

int soln[n]; ///Processes as per order of execution

Prc soln\_p[n];

int new\_order[n];

int k=0,i=0;

///To keep track of process to be executed first

int prev\_process,max\_b\_time,min\_arrival\_time=1000,min\_burst\_time=1000;

int processed[n+1]; ///Non of pocesses are processed

for(i=0; i<n; i++) processed[i]=0;

///k<n i/e while all n processes are not scheduled

i=0;

while(i<n)

{

if(k==0)

{

soln[k]=decide\_first(p1,processed,n,-1);

new\_order[k]=soln[k];

prev\_process=soln[k];

max\_b\_time=p1[soln[k]].a\_time;

k++;

i++;

}

else

{

max\_b\_time+=p1[prev\_process].b\_time;

int p=decide\_first(p1,processed,n,max\_b\_time);

if(p==-1)

{

i++;

continue; ///No iteration

}

new\_order[k]=p;

soln[k]=p;

prev\_process=soln[k];

k++;

i++;

}

}

///As per solution order design the arrange the processes

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

{

soln\_p[i]=p1[soln[i]];

}

///Rearrange processes

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

{

p1[i]=soln\_p[i];

order[i]=new\_order[i];

}

}

void SJF\_A(Prc p1[],int order[],int n)

{

int i,sum=0,gt\_itr=0;

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

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

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

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

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

{

bb[i]=0;

gt[i]=0;

}

decide\_order(p1,order,n);

///On this order run first come firts serve aogorithm

wt[0]=0; ///Initially waiting time is zero for first process in p1

gt[0]=p1[0].a\_time; ///As we will be executing processes right from first process which min arrival time

///It's not necesssary that this arrival time will always be zero

sum=p1[0].a\_time; ///AHence w eare adding up this time in sum as well!!

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

{

///Calculate waiting time

if(i>0)

{

/// for i=0, we already know waiting time is zero

/// otherwise it is burst time of prev process

sum=sum+p1[i-1].b\_time; /// Burst time of ith process

wt[i]=(sum-p1[i].a\_time);

///If process arrived much later while prev process was executing

if(p1[i].a\_time>sum)

{

///Thr process doent has to wait

wt[i]=0;

bb[i]=100;

gt[gt\_itr]=sum;

gt\_itr++;

///Add to burst time and time of this blank slot

sum+=(p1[i].a\_time-sum); //Add time taken by blank slot

}

gt[gt\_itr]=sum;

}

///Calculate turnround time

tt[i]=p1[i].b\_time+wt[i];

gt\_itr++;

}

///Add burst time of last process

gt[gt\_itr]=sum+p1[n-1].b\_time;

gt\_itr++;

printf("\n");

display(order,wt,tt,n);

ganttchart(order,gt,bb,n,gt\_itr);

}

void ganttchart(int \*order,int\*gt,int\* bb,int n,int n2)

{

int i;

printf("\nGantt chart is: \n");

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

{

if(bb[i]) printf("-----");

printf("-----");

}

printf("\n");

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

{

if(bb[i]==0) printf(" p%d |",order[i]+1);

else printf(" --| p%d |",order[i]+1);

}

printf("\n");

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

{

if(bb[i]) printf("-----");

printf("-----");

}

printf("\n");

for(i=0; i<n2; i++)

{

printf("%d ",gt[i]);

}

}

int main()

{

int n,i;

printf("Enter the number of processes: ");

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n);

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

for(i=0; i<n; i++) pr\_order[i]=i;

printf("\nEnter the BURST time and ARRIVAL time for: \n");

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

{

printf("Process %d: ",i);

scanf("%d",&p1[i].b\_time);

scanf("%d",&p1[i].a\_time);

p1[i].s\_key=-1;

}

printf("Process\t\tArrival time\t\tBurst time\n");

raw\_display(pr\_order,p1,n);

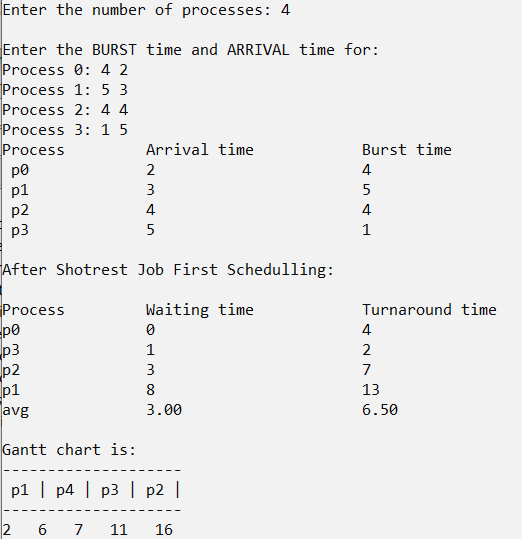
printf("\nAfter Shotrest Job First Schedulling:\n");

SJF\_A(p1,pr\_order,n);

return 0;

}

**Output:**

****

**6)Non preemtive priority With Arrival**

#include<stdio.h>

///With Arrival Non Pre-emtive priority

///Adnan Ismail Shah Muzavor

#include<stdlib.h>

/// Sorting key will help in sorting

/// when we want to sort values based on some

/// some other aspect

typedef struct array

{

int b\_time; ///burst\_time

int a\_time; ///arrival time

int s\_key; ///sorting key eg priority

} Prc;

///To find average

float avg(int \*arr,int n)

{

float avg=0.0;

int i=0;

for(i=0; i<n; i++) avg+=(float)arr[i];

return (avg/(float)n);

}

///To display the result

void display(int \*order,int \*wt,int \*tt,int n)

{

int i;

printf("Process\t\tWaiting time\t\tTurnaround time\n");

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

{

printf("p%d \t\t%d\t\t\t%d\n",order[i]+1,wt[i],tt[i]);

}

printf("avg \t\t%.2f\t\t\t%.2f\n",avg(wt,n),avg(tt,n));

}

///Display the input/sorted/processed input

void raw\_display(int order[],Prc p1[],int n)

{

int i;

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

{

printf(" p%d \t\t%d\t\t\t%d\t\t\t%d\n",order[i]+1,p1[i].a\_time,p1[i].b\_time,p1[i].s\_key);

}

}

int decide\_first(Prc p1[],int processed[],int n,int prev\_burst\_time)

{

int first\_p=-1,min\_arrival\_time=1000,highest\_priority=1000,i=0,min\_b\_time=1000; ///this is used only when processes hv sme priority

///Choose the first process to execute

if(prev\_burst\_time==-1)

{

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

{

///THe proces has arrival time lesser thn minimum arrival time so far

if(p1[i].a\_time!=min\_arrival\_time && p1[i].a\_time<min\_arrival\_time)

{

min\_arrival\_time=p1[i].a\_time;

highest\_priority=p1[i].s\_key;

first\_p=i;

}

///If process has same minimum arrival time, thn check for burst time

///Consider the one with minimum burst time

else if(p1[i].a\_time==min\_arrival\_time && p1[i].s\_key<highest\_priority)

{

min\_arrival\_time=p1[i].a\_time;

highest\_priority=p1[i].s\_key;

first\_p=i;

}

}

}

else

{

///No process executed therefore return process that will execute first

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

{

///If this process is not already included

if(!processed[i])

{

///If proces had arrived while prev was executing

if( p1[i].a\_time<prev\_burst\_time)

{

///If we have not selected any process as of now

if(first\_p==-1)

{

first\_p=i; ///Choose this process and thn look for other process;

highest\_priority=p1[i].s\_key;

min\_b\_time=p1[i].b\_time;

continue;

}

///If this process has lesser burst time thn proces we had arrived during this within same range

///i.e while prev process was executing

if(p1[i].s\_key<highest\_priority)

{

highest\_priority=p1[i].s\_key;

min\_b\_time=p1[i].b\_time;

first\_p=i;

}

else if(p1[i].s\_key==highest\_priority && p1[i].b\_time<min\_b\_time){

min\_b\_time=p1[i].b\_time;

first\_p=i;

}

}

}

}

}

///No new process found to be arrive within given burst time

if(first\_p==-1)

{

int min\_till\_now=1000;

highest\_priority=10000;

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

{

if(!processed[i])

{

int min\_diff=p1[i].a\_time-prev\_burst\_time;

if(min\_diff<min\_till\_now && p1[i].s\_key<highest\_priority )

{

first\_p=i;

min\_till\_now=min\_diff;

highest\_priority=p1[i].s\_key;

}

else if(min\_diff==min\_till\_now && p1[i].s\_key<highest\_priority){

first\_p=i;

highest\_priority=p1[i].s\_key;

}

}

}

};

if(first\_p==-1) return -1;

processed[first\_p]=1;

return first\_p;

}

int max(int a,int b)

{

if(a>b) return a;

return b;

}

void decide\_order(Prc p1[],int order[],int n)

{

int soln[n]; ///Processes as per order of execution

Prc soln\_p[n];

int new\_order[n];

int k=0,i=0;

///To keep track of process to be executed first

int prev\_process,max\_b\_time,min\_arrival\_time=1000,min\_burst\_time=1000;

int processed[n+1]; ///Non of pocesses are processed

for(i=0; i<n; i++) processed[i]=0;

///k<n i/e while all n processes are not scheduled

i=0;

while(i<n)

{

if(k==0)

{

soln[k]=decide\_first(p1,processed,n,-1);

new\_order[k]=soln[k];

prev\_process=soln[k];

max\_b\_time=p1[soln[k]].a\_time;

k++;

i++;

}

else

{

max\_b\_time+=p1[prev\_process].b\_time;

int p=decide\_first(p1,processed,n,max\_b\_time);

if(p==-1)

{

i++;

continue; ///No iteration

}

new\_order[k]=p;

soln[k]=p;

prev\_process=soln[k];

k++;

i++;

}

}

///As per solution order design the arrange the processes

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

{

soln\_p[i]=p1[soln[i]];

}

///Rearrange processes

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

{

p1[i]=soln\_p[i];

order[i]=new\_order[i];

}

}

void NPP\_A(Prc p1[],int order[],int n)

{

int i,sum=0,gt\_itr=0;

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

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

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

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

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

{

bb[i]=0;

gt[i]=0;

}

decide\_order(p1,order,n);

///On this order run first come firts serve aogorithm

wt[0]=0; ///Initially waiting time is zero for first process in p1

gt[0]=p1[0].a\_time; ///As we will be executing processes right from first process which min arrival time

///It's not necesssary that this arrival time will always be zero

sum=p1[0].a\_time; ///AHence w eare adding up this time in sum as well!!

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

{

///Calculate waiting time

if(i>0)

{

/// for i=0, we already know waiting time is zero

/// otherwise it is burst time of prev process

sum=sum+p1[i-1].b\_time; /// Burst time of ith process

wt[i]=(sum-p1[i].a\_time);

///If process arrived much later while prev process was executing

if(p1[i].a\_time>sum)

{

///Thr process doent has to wait

wt[i]=0;

bb[i]=100;

gt[gt\_itr]=sum;

gt\_itr++;

///Add to burst time and time of this blank slot

sum+=(p1[i].a\_time-sum); //Add time taken by blank slot

}

gt[gt\_itr]=sum;

}

///Calculate turnround time

tt[i]=p1[i].b\_time+wt[i];

gt\_itr++;

}

///Add burst time of last process

gt[gt\_itr]=sum+p1[n-1].b\_time;

gt\_itr++;

printf("\n");

display(order,wt,tt,n);

ganttchart(order,gt,bb,n,gt\_itr);

}

void ganttchart(int \*order,int\*gt,int\* bb,int n,int n2)

{

int i;

printf("\nGantt chart is: \n");

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

{

if(bb[i]) printf("-----");

printf("-----");

}

printf("\n");

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

{

if(bb[i]==0) printf(" p%d |",order[i]+1);

else printf(" --| p%d |",order[i]+1);

}

printf("\n");

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

{

if(bb[i]) printf("-----");

printf("-----");

}

printf("\n");

for(i=0; i<n2; i++)

{

printf("%d ",gt[i]);

}

}

int main()

{

int n,i;

printf("Enter the number of processes: ");

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n);

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

for(i=0; i<n; i++) pr\_order[i]=i;

printf("\nEnter the BURST time,PRIORITY , ARRIVAL time for: \n");

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

{

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

scanf("%d",&p1[i].b\_time);

scanf("%d",&p1[i].s\_key);

scanf("%d",&p1[i].a\_time);

}

printf("Process\t\tArrival time\t\tBurst time\t\tPriority\n");

raw\_display(pr\_order,p1,n);

///FCFS\_A(p1,pr\_order,n);

NPP\_A(p1,pr\_order,n);

return 0;

}

**Output:**

