**OPERATING SYSTEMS LAB**

**Scheduling Algorithms:**

1. **FCFS(FIRST COME FIRST SERVE):**

#include<stdio.h>

int find\_min(int n,int arr[][6],int i,int visit[])

{

int min=9999,flag;

int j;

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

{

if(arr[j][1]<min && visit[j]==-1)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int main()

{

int n,i,ct=0,j;

printf("enter the number of procesors: ");

scanf("%d",&n);

int arr[n][6],gantt\_chart[100],visit[n];

//P AT BT CT TAT WT

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

{

arr[i][0]=i;

visit[i]=-1;

}

printf("enter the values: \n");

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

{

printf("enter AT && BT values: ");

scanf("%d %d",&arr[i][1],&arr[i][2]);

}

int p=-1,l=0,k=0;

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

{

if(arr[i][1]==0)

{

p=i;

break;

}

}

if(p==-1)

{

gantt\_chart[l]=-1;

l++;

p=find\_min(n,arr,1,visit);

gantt\_chart[l]=p;

l++;

visit[p]=1;

arr[p][3]=arr[p][1]+arr[p][2];

ct=arr[p][3];

k++;

}

else

{

gantt\_chart[l]=p;

l++;

k++;

visit[p]=1;

arr[p][3]=arr[p][2];

ct=arr[p][3];

}

while(1)

{

p=find\_min(n,arr,1,visit);

if(arr[p][1]>ct)

{

gantt\_chart[l]=-1;

l++;

ct=arr[p][1];

}

visit[p]=1;

k++;

gantt\_chart[l]=p;

l++;

arr[p][3]=ct+arr[p][2];

ct=arr[p][3];

if(k==n)

{

break;

}

}

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

{

arr[i][4]=arr[i][3]-arr[i][1];

arr[i][5]=arr[i][4]-arr[i][2];

}

printf("Gantt\_Chart is: ");

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

{

if(gantt\_chart[i]==-1)

{

printf("idle ");

}

else

printf("p%d ",gantt\_chart[i]);

}

printf("\n");

printf("P\tAT\tBT\tCT\tTAT\tWT\n");

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

{

printf("p%d ",i);

for(j=1;j<6;j++)

{

printf("%d\t",arr[i][j]);

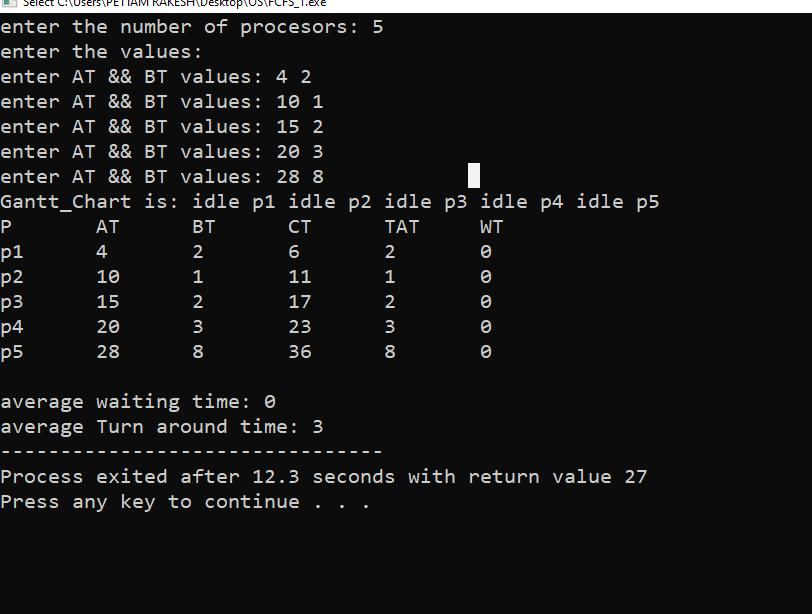
}

printf("\n");

}

}

**Output**:



**2.SJF(SHORTEST JOB FIRST)**

#include<stdio.h>

int find\_min\_at(int n,int arr[][6],int i,int visit[])

{

int min=9999,flag,ct=0;

int j;

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

{

if(arr[j][1]<min && visit[j]==-1)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int find\_min\_bt(int n,int arr[][6],int i,int visit[],int ct)

{

int min=9999,flag=-1;

int j;

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

{

if((arr[j][i]<min || (arr[j][i]==min && arr[j][1]<arr[flag][1])) && visit[j]==-1 && arr[j][1]<=ct)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int main()

{

int n,i,ct=0,j;

printf("enter the number of procesors: ");

scanf("%d",&n);

int arr[n][6],gantt\_chart[100],visit[n];

//P AT BT CT TAT WT

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

{

arr[i][0]=i;

visit[i]=-1;

}

printf("enter the values: \n");

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

{

printf("enter AT && BT values: ");

scanf("%d %d",&arr[i][1],&arr[i][2]);

}

int p=-1,l=0,k=0;

int min=999;

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

{

if(min>arr[i][1] || (min==arr[i][1] && arr[p][2]>arr[i][2]))

{

min=arr[i][1];

p=i;

}

}

ct=min;

if(min!=0)

{

gantt\_chart[l]=-1;

l++;

gantt\_chart[l]=p;

l++;

visit[p]=1;

arr[p][3]=arr[p][1]+arr[p][2];

ct=arr[p][3];

k++;

}

else

{

gantt\_chart[l]=p;

l++;

k++;

visit[p]=1;

arr[p][3]=arr[p][2];

ct=arr[p][3];

}

while(k!=n)

{

p=find\_min\_bt(n,arr,2,visit,ct);

if(p==-1)

{

gantt\_chart[l]=-1;

l++;

p=find\_min\_at(n,arr,1,visit);

ct=arr[p-1][1];

p=find\_min\_bt(n,arr,2,visit,ct);

}

visit[p]=1;

k++;

gantt\_chart[l]=p;

l++;

arr[p][3]=ct+arr[p][2];

ct=arr[p][3];

}

int sum\_wt=0,sum\_tat=0;

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

{

arr[i][4]=arr[i][3]-arr[i][1];

arr[i][5]=arr[i][4]-arr[i][2];

sum\_wt+=arr[i][5];

sum\_tat+=arr[i][4];

}

printf("Gantt\_Chart is: ");

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

{

if(gantt\_chart[i]==-1)

{

printf("idle ");

}

else

printf("p%d ",gantt\_chart[i]);

}

printf("\n");

printf("P\tAT\tBT\tCT\tTAT\tWT\n");

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

{

printf("p%d ",i);

for(j=1;j<6;j++)

{

printf("%d\t",arr[i][j]);

}

printf("\n");

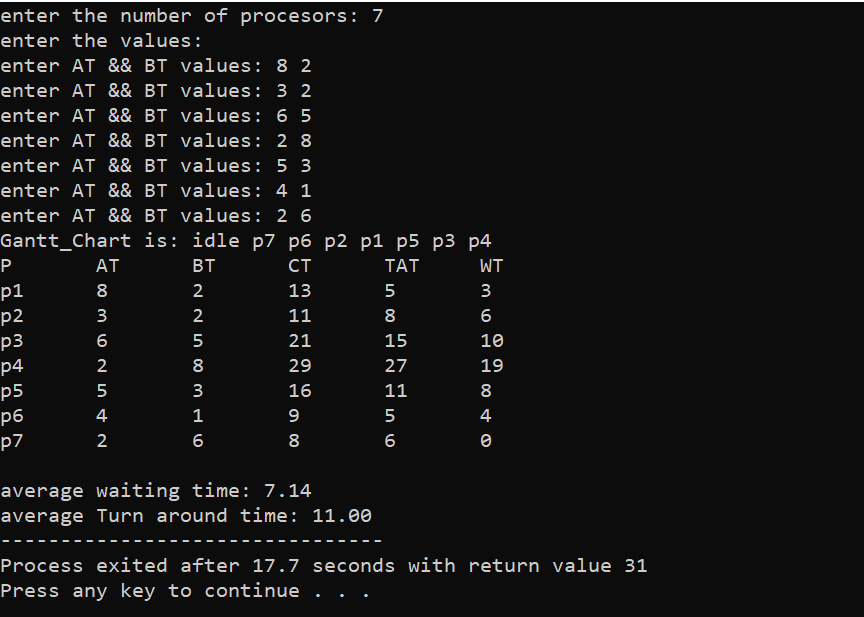
}

printf("\naverage waiting time: %d\n",sum\_wt/n);

printf("average Turn around time: %d",sum\_tat/n);

}

**Output:**



**3.PRIORITY**

#include<stdio.h>

int find\_min\_at(int n,int arr[][7],int i,int visit[])

{

int min=9999,flag,ct=0;

int j;

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

{

if(arr[j][1]<min && visit[j]==-1)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int find\_min\_priority(int n,int arr[][7],int i,int visit[],int ct)

{

int min=9999,flag=-1;

int j;

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

{

if((arr[j][i]<min || (arr[j][i]==min && arr[j][2]<arr[flag][2])) && visit[j]==-1 && arr[j][2]<=ct)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int main()

{

int n,i,ct=0,j;

printf("enter the number of procesors: ");

scanf("%d",&n);

int arr[n][7],gantt\_chart[100],visit[n];

//P PRIORITY AT BT CT TAT WT

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

{

arr[i][0]=i;

visit[i]=-1;

}

printf("enter the values: \n");

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

{

printf("enter Priority && AT && BT values: ");

scanf("%d %d %d",&arr[i][1],&arr[i][2],&arr[i][3]);

}

int p=-1,l=0,k=0;

int min=999;

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

{

if(min>arr[i][2] || (min==arr[i][2] && arr[p][1]>arr[i][1]))

{

min=arr[i][2];

p=i;

}

}

ct=min;

if(min!=0)

{

gantt\_chart[l]=-1;

l++;

gantt\_chart[l]=p;

printf("%d\n",gantt\_chart[l]);

l++;

visit[p]=1;

arr[p][4]=ct+arr[p][3];

ct=arr[p][4];

k++;

}

else

{

gantt\_chart[l]=p;

l++;

k++;

visit[p]=1;

arr[p][4]=arr[p][3];

ct=arr[p][4];

}

while(k!=n)

{

p=find\_min\_priority(n,arr,1,visit,ct);

if(p==-1)

{

gantt\_chart[l]=-1;

l++;

p=find\_min\_at(n,arr,2,visit);

ct=arr[p][2];

p=find\_min\_priority(n,arr,1,visit,ct);

}

visit[p]=1;

k++;

gantt\_chart[l]=p;

l++;

arr[p][4]=ct+arr[p][3];

ct=arr[p][4];

}

printf("Gantt\_Chart is: ");

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

{

if(gantt\_chart[i]==-1)

{

printf("idle ");

}

else

{

printf("p%d ",gantt\_chart[i]);

}

}

float sum\_wt=0,sum\_tat=0;

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

{

arr[i][5]=arr[i][4]-arr[i][2];

arr[i][6]=arr[i][5]-arr[i][3];

sum\_wt+=arr[i][6];

sum\_tat+=arr[i][5];

}

printf("\n");

printf("P\tPrior\tAT\tBT\tCT\tTAT\tWT\n");

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

{ printf("p%d ",i);

for(j=1;j<7;j++)

{

printf("%d\t",arr[i][j]);

}

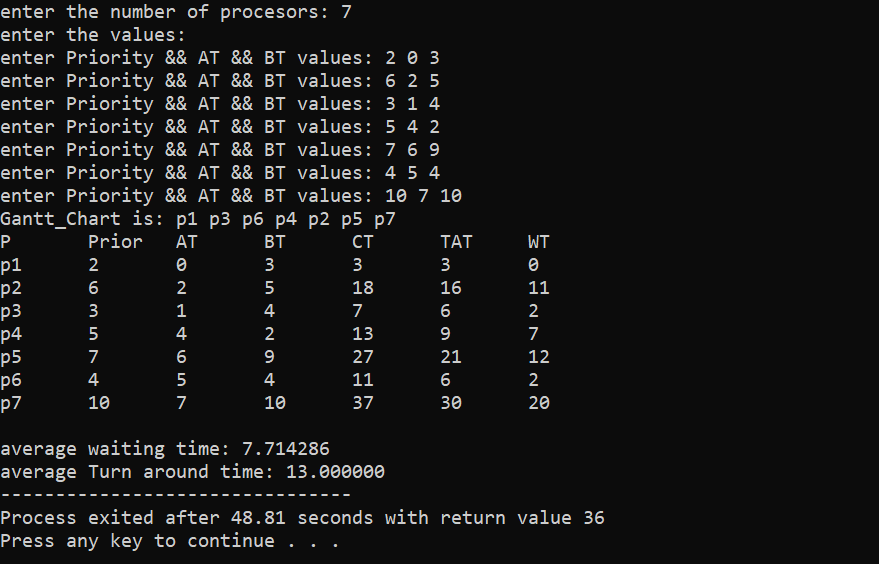
printf("\n");

}

printf("\naverage waiting time: %f",sum\_wt/n);

printf("\naverage Turn around time: %f",sum\_tat/n);

}



**3.SRTF(SHORTEST REMAIN TIME FIRST)**

#include<stdio.h>

int find\_min\_at(int n,int arr[][6],int i,int visit[])

{

int min=9999,flag;

int j;

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

{

if(arr[j][1]<min && visit[j]==-1)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int find\_min(int n,int arr[][6],int i,int visit[],int ct)

{

int min=9999,flag=-1;

int j;

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

{

if((arr[j][i]<min || (arr[j][i]==min && arr[j][1]<=arr[flag][1])) && visit[j]==-1 && arr[j][1]<=ct)

{

min=arr[j][i];

flag=j;

}

}

if(flag==-1)

{

return -1;

}

else

{

return flag;

}

}

int main()

{

int n,i,ct=0,j;

printf("enter the number of procesors: ");

scanf("%d",&n);

int arr[n][6],gantt\_chart[100],visit[n];

int bt[n];

//P AT BT CT TAT WT

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

{

arr[i][0]=i;

visit[i]=-1;

}

printf("enter the values: \n");

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

{

printf("enter AT && BT values: ");

scanf("%d %d",&arr[i][1],&arr[i][2]);

arr[i][3]=0;

bt[i]=arr[i][2];

}

int p=-1,l=0,k=0;

int min=999;

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

{

if(min>arr[i][1] || (min==arr[i][1] && arr[p][2]>arr[i][2]))

{

min=arr[i][1];

p=i;

}

}

ct=min;

if(min!=0)

{

gantt\_chart[l]=-1;

l++;

gantt\_chart[l]=p;

l++;

arr[p][2]=arr[p][2]-1;

arr[p][3]++;

ct=arr[p][3];

}

else

{

gantt\_chart[l]=p;

l++;

arr[p][2]=arr[p][2]-1;

arr[p][3]++;

ct=arr[p][3];

}

if(arr[p][2]==0)

{

visit[p]=1;

k++;

}

while(k!=n)

{

p=find\_min(n,arr,2,visit,ct);

if(p==-1)

{

gantt\_chart[l]=-1;

l++;

p=find\_min\_at(n,arr,1,visit);

ct=arr[p][1];

p=find\_min(n,arr,2,visit,ct);

}

arr[p][2]--;

if(gantt\_chart[l-1]==p)

{

arr[p][3]=ct+1;

ct++;

}

else

{

gantt\_chart[l]=p;

l++;

arr[p][3]=ct+1;

ct++;

}

if(arr[p][2]==0)

{

visit[p]=1;

k++;

}

}

float sum\_wt=0,sum\_tat=0;

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

{

arr[i][4]=arr[i][3]-arr[i][1];

arr[i][5]=arr[i][4]-bt[i];

sum\_wt+=arr[i][5];

sum\_tat+=arr[i][4];

}

printf("Gantt\_Chart is: ");

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

{

if(gantt\_chart[i]==-1)

{

printf("idle ");

}

else

printf("p%d ",gantt\_chart[i]);

}

printf("\n");

printf("P\tAT\tBT\tCT\tTAT\tWT\n");

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

{

printf("p%d ",i);

arr[i][2]=bt[i];

for(j=1;j<6;j++)

{

printf("%d\t",arr[i][j]);

}

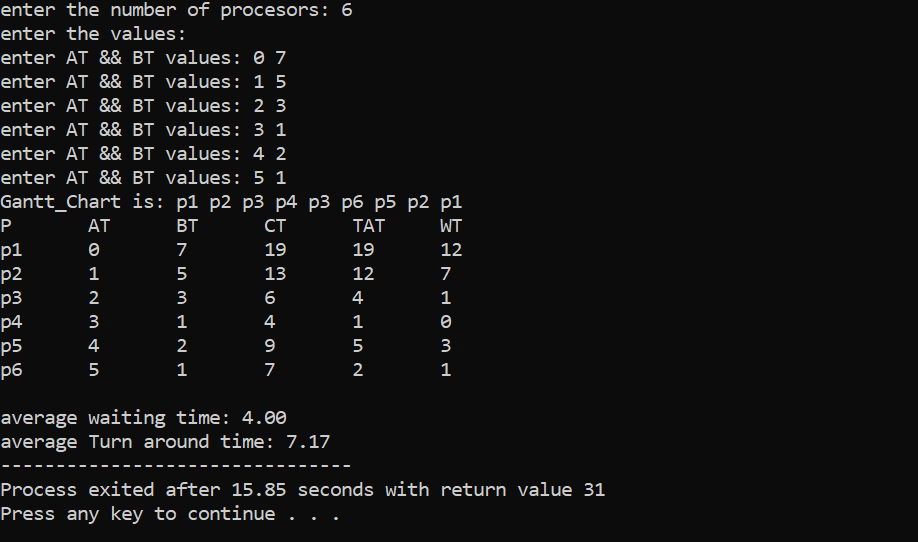
printf("\n");

}

printf("\naverage waiting time: %.2f",sum\_wt/n);

printf("\naverage Turn around time: %.2f",sum\_tat/n);

}



1. **PREEMTIVE PREORITY:**

#include<stdio.h>

int find\_min\_at(int n,int arr[][7],int i,int visit[])

{

int min=9999,flag,ct=0;

int j;

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

{

if((min>arr[i][2] || (min==arr[i][2] && (arr[flag][1]>arr[i][1] ||(arr[flag][1]==arr[i][1] && arr[flag][3]>arr[i][3])))) && visit[j]==-1)

{

min=arr[j][i];

flag=j;

}

}

return flag;

}

int find\_min\_priority(int n,int arr[][7],int i,int visit[],int ct)

{

int min=9999,flag=-1;

int j;

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

{

if((arr[j][i]<min || (arr[j][i]==min && (arr[j][3]<arr[flag][3] || (arr[j][3]==arr[flag][3] && arr[j][2]<arr[flag][2])))) && visit[j]==-1 && arr[j][2]<=ct)

{

min=arr[j][i];

flag=j;

}

}

if(flag==-1)

{

return -1;

}

else

{

return flag;

}

}

int main()

{

int n,i,ct=0,j;

printf("enter the number of procesors: ");

scanf("%d",&n);

int arr[n][7],gantt\_chart[100],visit[n];

int bt[n];

//P PRIORITY AT BT CT TAT WT

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

{

arr[i][0]=i;

visit[i]=-1;

}

printf("enter the values: \n");

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

{

printf("enter Priority && AT && BT values: ");

scanf("%d %d %d",&arr[i][1],&arr[i][2],&arr[i][3]);

arr[i][4]=0;

bt[i]=arr[i][3];

}

int p=-1,l=0,k=0;

int min=999;

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

{

if(min>arr[i][2] || (min==arr[i][2] && (arr[p][1]>arr[i][1] ||(arr[p][1]==arr[i][1] && arr[p][3]>arr[i][3]))))

{

min=arr[i][2];

p=i;

}

}

ct=0;

if(min!=0)

{

ct=min;

gantt\_chart[l]=-1;

l++;

gantt\_chart[l]=p;

l++;

ct++;

arr[p][3]--;

arr[p][4]=ct;

}

else

{

gantt\_chart[l]=p;

l++;

arr[p][3]--;

ct++;

arr[p][4]=ct;

}

if(arr[p][3]==0)

{

k++;

visit[p]=1;

}

while(k!=n)

{

p=find\_min\_priority(n,arr,1,visit,ct);

if(p==-1)

{

gantt\_chart[l]=-1;

l++;

p=find\_min\_at(n,arr,2,visit);

ct=arr[p][2];

p=find\_min\_priority(n,arr,1,visit,ct);

}

arr[p][3]--;

if(gantt\_chart[l-1]==p)

{

ct++;

arr[p][4]=ct;

}

else

{

gantt\_chart[l]=p;

l++;

ct++;

arr[p][4]=ct;

}

if(arr[p][3]==0)

{

visit[p]=1;

k++;

}

}

float sum\_wt=0,sum\_tat=0;

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

{

arr[i][5]=arr[i][4]-arr[i][2];

arr[i][6]=arr[i][5]-bt[i];

sum\_wt+=arr[i][6];

sum\_tat+=arr[i][5];

}

printf("Gantt\_Chart is: ");

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

{

if(gantt\_chart[i]==-1)

{

printf("idle ");

}

else

printf("p%d ",gantt\_chart[i]);

}

printf("\n");

printf("P\tPrior\tAT\tBT\tCT\tTAT\tWT\n");

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

{

printf("p%d ",i);

arr[i][3]=bt[i];

for(j=1;j<7;j++)

{

printf("%d\t",arr[i][j]);

}

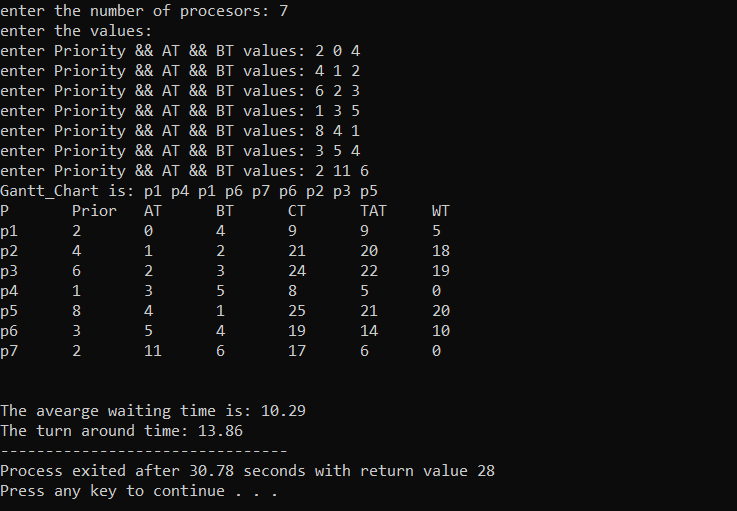
printf("\n");

}

printf("\n\nThe avearge waiting time is: %.2f",sum\_wt/n);

printf("\nThe turn around time: %.2f",sum\_tat/n);

}



**6.ROUND ROBIN(RR)**

#include<stdio.h>

int queue[100];

int st=-1,end=-1;

int e=0;

int find\_min(int n,int arr[][6],int i,int vis[])

{

int min=9999,flag;

int j;

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

{

if(arr[j][1]<min && vis[j]==-1)

{

min=arr[j][1];

flag=j;

}

}

return flag;

}

int find\_min1(int n,int arr[][6],int visit[],int ct)

{

int i,min=9999,flag=-1;

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

{

if(arr[i][1]<min && arr[i][1]<=ct && visit[i]==-1)

{

min=arr[i][1];

flag=i;

}

}

return flag;

}

int find\_process(int n,int arr[][6],int visit[],int ct)

{

int p=find\_min1(n,arr,visit,ct);

while(p!=-1)

{

enque(p);

visit[p]=1;

e++;

p=find\_min1(n,arr,visit,ct);

}

}

int enque(int p)

{

if(st==-1)

{

st=0;

end=0;

queue[end]=p;

}

else{

end++;

queue[end]=p;

}

}

int deque()

{

if(st==-1)

{

return -1;

}

if(st==end)

{

int r=queue[st];

st=-1;

end=-1;

return r;

}

else{

int r=queue[st];

st++;

return r;

}

}

int main()

{

int n,i,ct=0,j,TQ;

printf("enter the number of procesors: ");

scanf("%d",&n);

int arr[n][6],gantt\_chart[100],visit[n],vis[n];

int bt[n];

printf("enter the time quantum value: ");

scanf("%d",&TQ);

//P AT BT CT TAT WT

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

{

arr[i][0]=i;

visit[i]=-1;

vis[i]=-1;

}

printf("enter the values: \n");

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

{

printf("enter AT && BT values: ");

scanf("%d %d",&arr[i][1],&arr[i][2]);

bt[i]=arr[i][2];

}

int p=-1,l=0,k=0;

int min=999;

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

{

if(min>arr[i][1])

{

min=arr[i][1];

}

}

ct=0;

if(min!=0)

{

ct=min;

gantt\_chart[l]=-1;

l++;

}

find\_process(n,arr,visit,ct);

p=deque();

gantt\_chart[l]=p;

l++;

if(arr[p][2]>TQ)

{

arr[p][2]=(arr[p][2]-TQ);

ct=ct+TQ;

}

else{

ct=ct+arr[p][2];

arr[p][3]=ct;

arr[p][2]=0;

vis[p]=1;

k++;

}

while(k!=n)

{

if(e!=n)

{

find\_process(n,arr,visit,ct);

if(arr[p][2]!=0)

{

enque(p);

}

}

p=deque();

if(p==-1)

{

gantt\_chart[l]=-1;

l++;

p=find\_min(n,arr,1,vis);

ct=arr[p][1];

find\_process(n,arr,visit,ct);

p=deque();

}

if(gantt\_chart[l-1]!=p)

{

gantt\_chart[l]=p;

l++;

}

if(arr[p][2]>TQ)

{

arr[p][2]=arr[p][2]-TQ;

ct=ct+TQ;

if(e==n)

{

enque(p);

}

}

else{

ct=ct+arr[p][2];

arr[p][3]=ct;

arr[p][2]=0;

vis[p]=1;

k++;

}

}

float sum\_wt=0,sum\_tat=0;

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

{

arr[i][4]=arr[i][3]-arr[i][1];

arr[i][5]=arr[i][4]-bt[i];

sum\_wt+=arr[i][5];

sum\_tat+=arr[i][4];

}

printf("Gantt\_Chart is: ");

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

{

if(gantt\_chart[i]==-1)

{

printf("idle ");

}

else

printf("p%d ",gantt\_chart[i]);

}

printf("\n");

printf("P\tAT\tBT\tCT\tTAT\tWT\n");

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

{

printf("p%d ",i);

arr[i][2]=bt[i];

for(j=1;j<6;j++)

{

printf("%d\t",arr[i][j]);

}

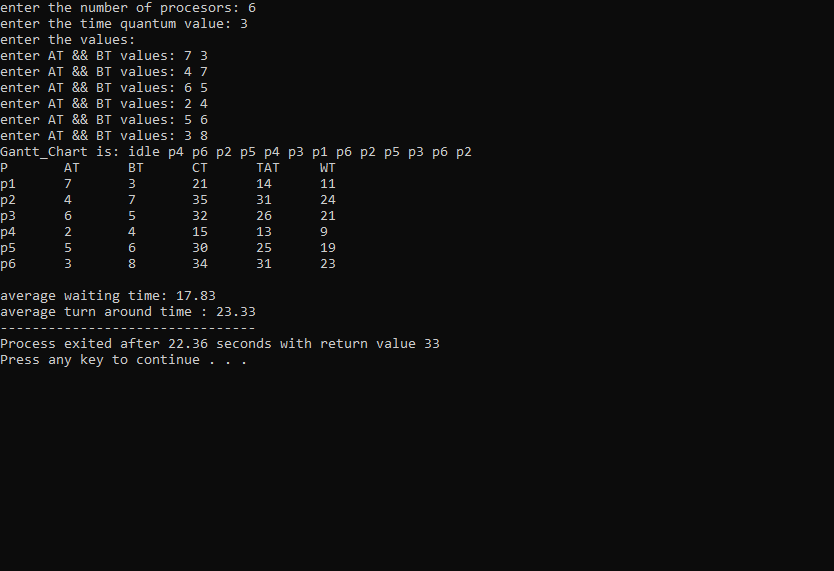
printf("\n");

}

printf("\naverage waiting time: %.2f",sum\_wt/n);

printf("\naverage turn around time : %.2f",sum\_tat/n);

}



**Memory Allocations**

1. **Fixed Partition First Fit:**

#include<stdio.h>

int main()

{

int n,i,j;

printf("enter the number of process: ");

scanf("%d",&n);

int m,size[n],a[n];

printf("enter the sizes of process: \n");

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

{

printf("process-%d size: ",i);

scanf("%d",&size[i]);

}

printf("enter the number of blocks in memory: ");

scanf("%d",&m);

int block[m],visit[m];

printf("Enter the sizes of each blocks: \n");

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

{

printf("size of block-%d ",i);

scanf("%d",&block[i]);

visit[i]=-1;

}

int k=1,count=0;

while(k<=n){

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

{

if(block[i]>=size[k] && visit[i]==-1)

{

visit[i]=k;

block[i]=block[i]-size[k];

k++;

break;

}

}

if(i>m)

{

a[count]=k;

count++;

printf("process-%d is can not insert into memory\n",k);

k++;

}

}

printf("the process allocated are: \n");

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

{

if(visit[i]!=-1)

{

printf("block-%d: p%d\n",i,visit[i]);}

}

int sum=0;

printf("internal fragment of memory: ");

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

{

if(visit[i]!=-1)

{

sum=sum+block[i];

}

}

printf("%d\n",sum);

if(count!=0)

{

sum=0;

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

{

if(visit[i]==-1){

sum=sum+block[i];

}

}

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

{

if(sum>=size[a[i]])

{

break;

}

}

if(i<count)

{

printf("external segment: %d",sum);

}

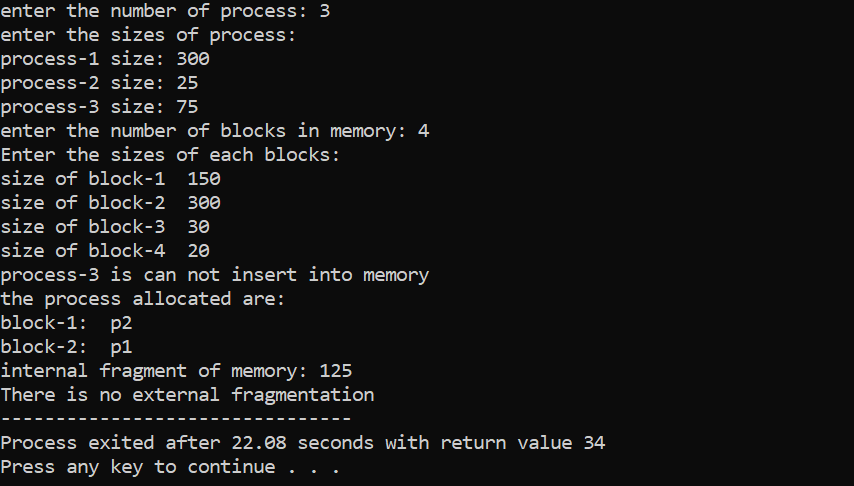
else{

printf("There is no external fragmentation");

}

}

}



1. **Fixed Partition Best Fit:**

#include<stdio.h>

int main()

{

int n,i,j;

printf("enter the number of process: ");

scanf("%d",&n);

int m,size[n];

printf("enter the sizes of process: \n");

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

{

printf("process-%d size: ",i);

scanf("%d",&size[i]);

}

printf("enter the number of blocks in memory: ");

scanf("%d",&m);

int block[m],visit[m];

printf("Enter the sizes of each blocks: \n");

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

{

printf("size of block-%d ",i);

scanf("%d",&block[i]);

visit[i]=-1;

}

int a[n];

int k=1,count=0;

while(k<=n){

int min=999,flag=0;

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

{

if(block[i]<min && block[i]>=size[k] && visit[i]==-1)

{

min=block[i];

flag=i;

}

}

if(flag!=0)

{

visit[flag]=k;

block[flag]=block[flag]-size[k];

k++;

}

else

{

a[count]=k;

count++;

printf("process-%d is can not insert into memory\n",k);

k++;

}

}

printf("the process allocated are: \n");

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

{

if(visit[i]!=-1)

{

printf("block-%d: p%d\n",i,visit[i]);

}

}

int sum=0;

printf("internal fragment of memory: ");

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

{

if(visit[i]!=-1)

{

sum=sum+block[i];

}

}

printf("%d\n",sum);

if(count!=0)

{

sum=0;

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

{

if(visit[i]==-1){

sum=sum+block[i];

}

}

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

{

if(sum>=size[a[i]])

{

break;

}

}

if(i<count)

{

printf("external fragment: %d",sum);

}

else

{

printf("There is no external fragmentation");

}

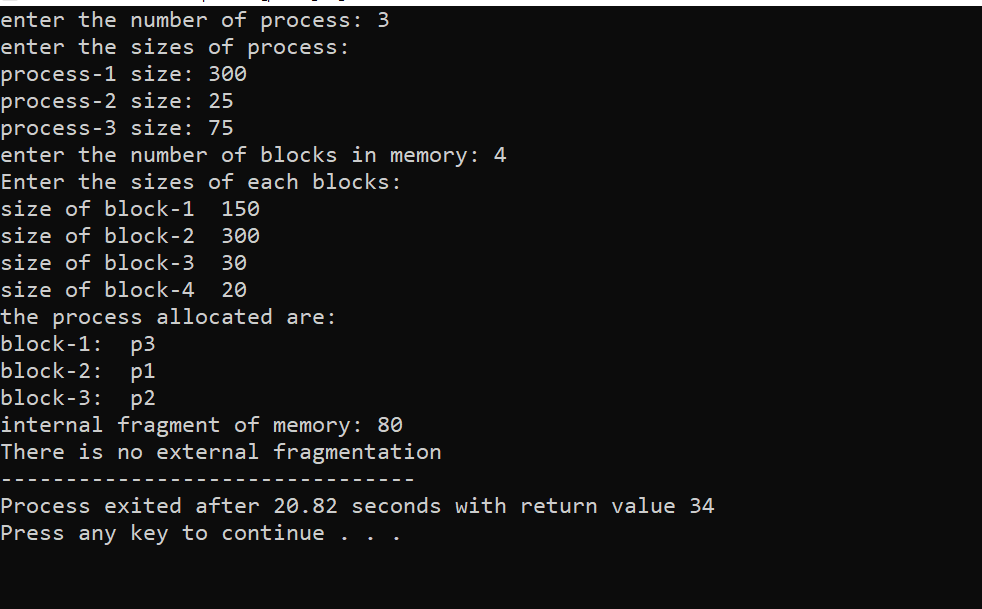
}

else

{

printf("There is no external fragmentation");

}

}

1. **Fixed Partition Worst Fit:**

#include<stdio.h>

int main()

{

int n,i,j;

printf("enter the number of process: ");

scanf("%d",&n);

int m,size[n];

printf("enter the sizes of process: \n");

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

{

printf("process-%d size: ",i);

scanf("%d",&size[i]);

}

printf("enter the number of blocks in memory: ");

scanf("%d",&m);

int block[m],visit[m];

printf("Enter the sizes of each blocks: \n");

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

{

printf("size of block-%d ",i);

scanf("%d",&block[i]);

visit[i]=-1;

}

int a[n];

int k=1,count=0;

while(k<=n){

int min=0,flag=0;

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

{

if(block[i]>min && block[i]>=size[k] && visit[i]==-1)

{

min=block[i];

flag=i;

}

}

if(flag!=0)

{

visit[flag]=k;

block[flag]=block[flag]-size[k];

k++;

}

else

{

a[count]=k;

count++;

printf("process-%d is can not insert into memory\n",k);

k++;

}

}

printf("the process allocated are: \n");

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

{

if(visit[i]!=-1)

{

printf("block-%d: p%d\n",i,visit[i]);

}

}

int sum=0;

printf("internal fragment of memory: ");

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

{

if(visit[i]!=-1)

{

sum=sum+block[i];

}

}

printf("%d\n",sum);

if(count!=0)

{

sum=0;

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

{

if(visit[i]==-1){

sum=sum+block[i];

}

}

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

{

if(sum>=size[a[i]])

{

break;

}

}

if(i<count)

{

printf("external fragment: %d",sum);

}

else

{

printf("There is no external fragmentation");

}

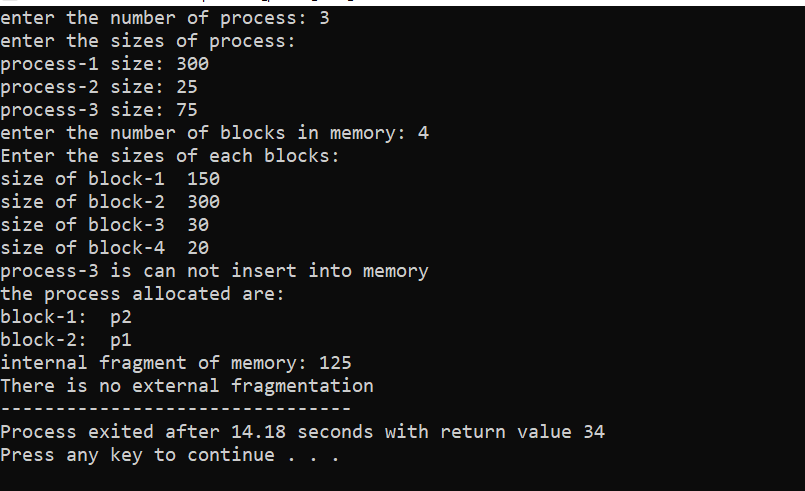
}

else

{

printf("There is no external fragmentation");

}

}

1. **Variable Partition First Fit:**

#include<stdio.h>

int main()

{

int n,i,j;

printf("enter the number of process: ");

scanf("%d",&n);

int m,size[n],a[n],store[n];

printf("enter the sizes of process: \n");

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

{

printf("process-%d size: ",i);

scanf("%d",&size[i]);

store[i]=-1;

}

printf("enter the number of blocks in memory: ");

scanf("%d",&m);

int block[m],visit[m];

printf("Enter the sizes of each blocks: \n");

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

{

printf("size of block-%d ",i);

scanf("%d",&block[i]);

visit[i]=-1;

}

int k=1,count=0;

while(k<=n){

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

{

if(block[i]>=size[k])

{

store[k]=i;

block[i]=block[i]-size[k];

k++;

visit[i]=1;

break;

}

}

if(i>m)

{

a[count]=k;

count++;

printf("process-%d is can not insert into memory\n",k);

k++;

}

}

printf("the process allocated are: \n");

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

{

if(store[i]!=-1)

{

printf("p%d: at block-%d\n",i,store[i]);

}

}

int sum=0;

printf("internal fragment of memory: ");

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

{

if(visit[i]!=-1)

{

sum=sum+block[i];

}

}

printf("%d\n",sum);

if(count!=0)

{

sum=0;

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

{

if(visit[i]==-1){

sum=sum+block[i];

}

}

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

{

if(sum>=size[a[i]])

{

break;

}

}

if(i<count)

{

printf("external segment: %d",sum);

}

else

{

printf("There is no external fragmentation");

}

}

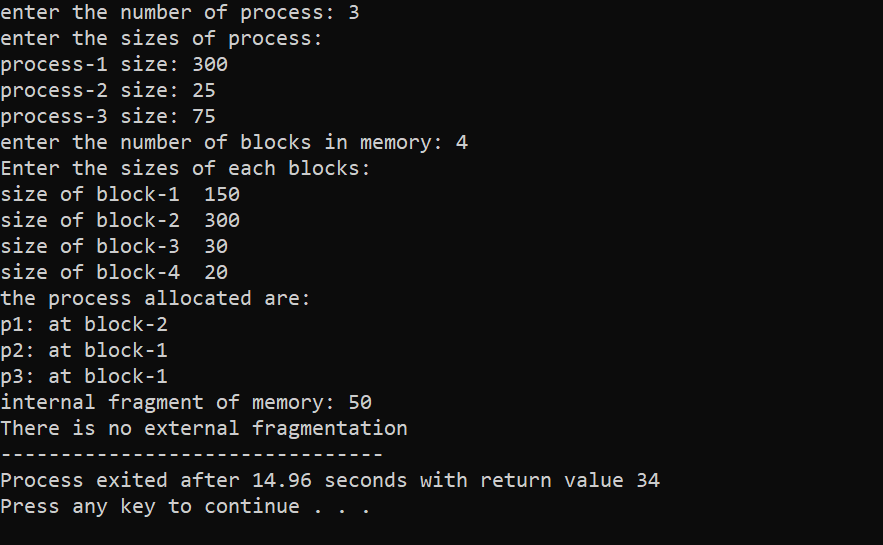
else

{

printf("There is no external fragmentation");

}

}



1. **Variable Partition Best Fit:**

#include<stdio.h>

int main()

{

int n,i,j;

printf("enter the number of process: ");

scanf("%d",&n);

int m,size[n],a[n],store[n];

printf("enter the sizes of process: \n");

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

{

printf("process-%d size: ",i);

scanf("%d",&size[i]);

store[i]=-1;

}

printf("enter the number of blocks in memory: ");

scanf("%d",&m);

int block[m],visit[m];

printf("Enter the sizes of each blocks: \n");

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

{

printf("size of block-%d ",i);

scanf("%d",&block[i]);

visit[i]=-1;

}

int k=1,count=0;

while(k<=n){

int min=999,flag=0;

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

{

if(block[i]<min && block[i]>=size[k])

{

min=block[i];

flag=i;

}

}

if(flag!=0)

{

store[k]=flag;

visit[flag]=1;

block[flag]=block[flag]-size[k];

k++;

}

else

{

a[count]=k;

count++;

printf("process-%d is can not insert into memory\n",k);

k++;

}

}

printf("the process allocated are: \n");

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

{

if(store[i]!=-1)

{

printf("p%d: at block-%d\n",i,store[i]);

}

}

int sum=0;

printf("internal fragment of memory: ");

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

{

if(visit[i]!=-1)

{

sum=sum+block[i];

}

}

printf("%d\n",sum);

if(count!=0)

{

sum=0;

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

{

if(visit[i]==-1){

sum=sum+block[i];

}

}

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

{

if(sum>=size[a[i]])

{

break;

}

}

if(i<count)

{

printf("external segment: %d",sum);

}

else

{

printf("There is no external fragmentation");

}

}

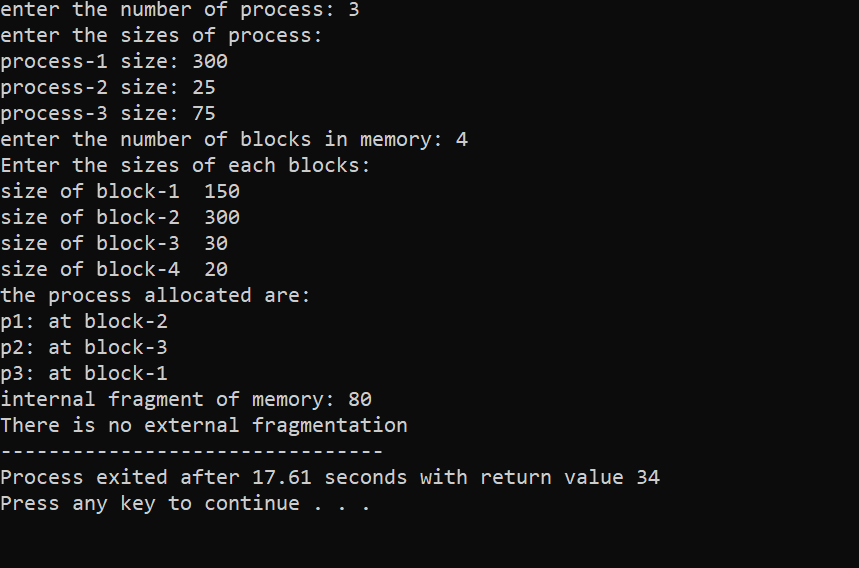
else

{

printf("There is no external fragmentation");

}

}



1. **Variable Partition Worst Fit:**

#include<stdio.h>

int main()

{

int n,i,j;

printf("enter the number of process: ");

scanf("%d",&n);

int m,size[n],a[n],store[n];

printf("enter the sizes of process: \n");

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

{

printf("process-%d size: ",i);

scanf("%d",&size[i]);

store[i]=-1;

}

printf("enter the number of blocks in memory: ");

scanf("%d",&m);

int block[m],visit[m];

printf("Enter the sizes of each blocks: \n");

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

{

printf("size of block-%d ",i);

scanf("%d",&block[i]);

visit[i]=-1;

}

int k=1,count=0;

while(k<=n){

int min=0,flag=0;

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

{

if(block[i]>min && block[i]>=size[k])

{

min=block[i];

flag=i;

}

}

if(flag!=0)

{

store[k]=flag;

visit[flag]=1;

block[flag]=block[flag]-size[k];

k++;

}

else

{

a[count]=k;

count++;

printf("process-%d is can not insert into memory\n",k);

k++;

}

}

printf("the process allocated are: \n");

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

{

if(store[i]!=-1)

{

printf("p%d: at block-%d\n",i,store[i]);

}

}

int sum=0;

printf("internal fragment of memory: ");

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

{

if(visit[i]!=-1)

{

sum=sum+block[i];

}

}

printf("%d\n",sum);

if(count!=0)

{

sum=0;

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

{

if(visit[i]==-1){

sum=sum+block[i];

}

}

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

{

if(sum>=size[a[i]])

{

break;

}

}

if(i<count)

{

printf("external segment: %d",sum);

}

else

{

printf("There is no external fragmentation");

}

}

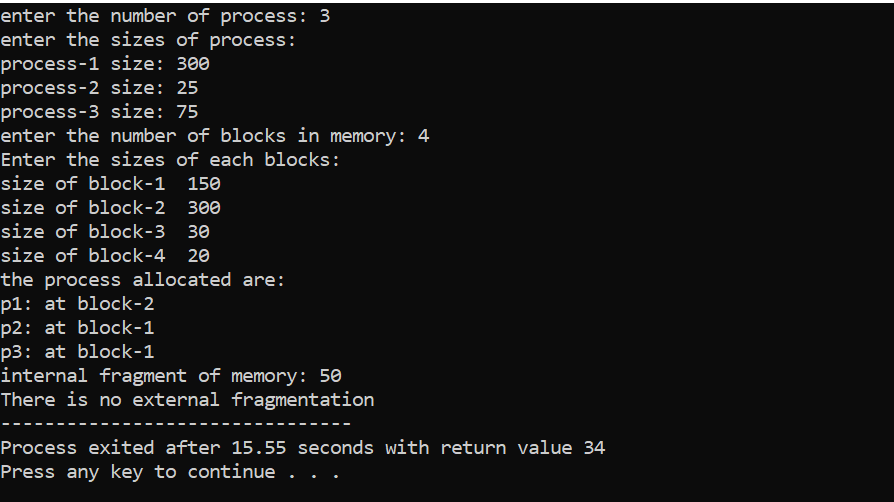
else

{

printf("There is no external fragmentation");

}

}



**Bankers Algorithm**

1. **Safe Sequence:**

#include<stdio.h>

int n,r;

int allocation[100][100],max[100][100],need[100][100],available[100];

int check(int i,int n,int r,int work[])

{

int j,flag=-1;

for(j=1;j<=r;j++)

{

if(work[j]<need[i][j])

{

flag=1;

break;

}

}

return flag;

}

int printing(int sequence[])

{

int i;

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

{

printf("p%d\t",sequence[i]);

}

printf("\n");

}

int Safety\_sequence(int sequence[],int finish[],int work[],int k)

{

int i,j;

if(k>n)

{

printing(sequence);

return;

}

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

{

int p;

p=check(i,n,r,work);

if(p==-1 && finish[i]==-1)

{

sequence[k]=i;

finish[i]=1;

for(j=1;j<=r;j++)

{

work[j]=work[j]+allocation[i][j];

}

Safety\_sequence(sequence,finish,work,k+1);

for(j=1;j<=r;j++)

{

work[j]=work[j]-allocation[i][j];

}

finish[i]=-1;

}

}

}

int main()

{

int i,j;

printf("enter the number of process: ");

scanf("%d",&n);

printf("enter the number of resorces: ");

scanf("%d",&r);

int finish[n],work[r],sequence[n];

printf("enter the allocation matrix: \n");

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

{

finish[i]=-1;

printf("process-%d Allocation: ",i);

for(j=1;j<=r;j++)

{

scanf("%d",&allocation[i][j]);

}

}

printf("enter the max need of process: \n");

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

{

printf("process-%d max need: ",i);

for(j=1;j<=r;j++)

{

scanf("%d",&max[i][j]);

}

}

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

{

for(j=1;j<=r;j++)

{

need[i][j]=max[i][j]-allocation[i][j];

}

}

printf("enter the available instances: ");

int k=1;

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

{

scanf("%d",&available[i]);

work[i]=available[i];

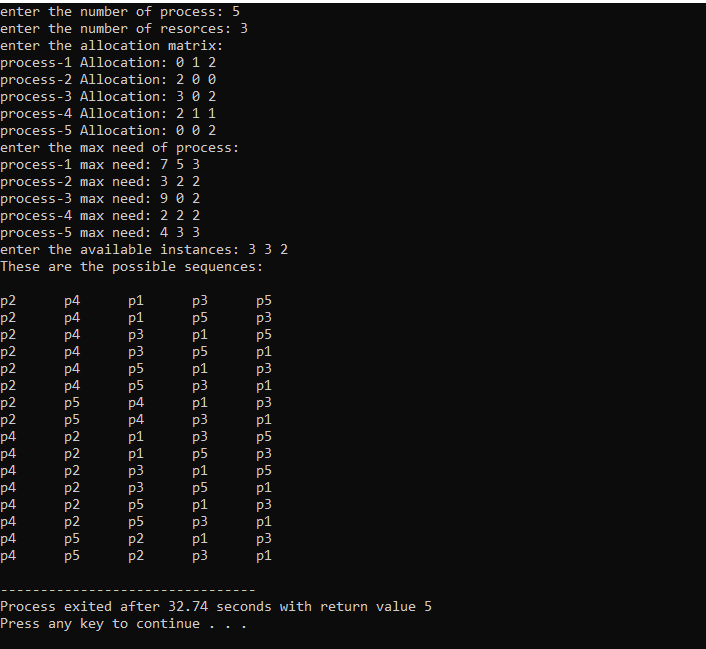
}

//here starts logic

printf("These are the possible sequences: \n\n");

Safety\_sequence(sequence,finish,work,k);

}



1. **Resource Request Algorithm:**

#include<stdio.h>

int n,r,l=0;

int allocation[100][100],max[100][100],need[100][100],available[100];

int check2(int i,int request[],int req\_arr[][r])

{

int j,flag=1;

for(j=1;j<=r;j++)

{

if(req\_arr[i][j]>need[request[i]][j])

{

flag=0;

break;

}

}

return flag;

}

int check(int i,int n,int r,int work[])

{

int j,flag=-1;

for(j=1;j<=r;j++)

{

if(work[j]<need[i][j])

{

flag=1;

break;

}

}

return flag;

}

int printing(int sequence[],int k)

{

int i;

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

{

printf("p%d\t",sequence[i]);

}

printf("\n");

}

int Safety\_sequence(int sequence[],int finish[],int work[],int k)

{

int i,j;

if(k>l+n)

{

printing(sequence,k);

return;

}

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

{

int p;

p=check(i,n,r,work);

if(p==-1 && finish[i]==-1)

{

sequence[k]=i;

finish[i]=1;

for(j=1;j<=r;j++)

{

work[j]=work[j]+allocation[i][j];

}

Safety\_sequence(sequence,finish,work,k+1);

for(j=1;j<=r;j++)

{

work[j]=work[j]-allocation[i][j];

}

finish[i]=-1;

}

}

}

int main()

{

int i,j;

printf("enter the number of process: ");

scanf("%d",&n);

printf("enter the number of resorces: ");

scanf("%d",&r);

int finish[n],work[r],sequence[100];

printf("enter the allocation matrix: \n");

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

{

finish[i]=-1;

printf("process-%d Allocation: ",i);

for(j=1;j<=r;j++)

{

scanf("%d",&allocation[i][j]);

}

}

printf("enter the max need of process: \n");

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

{

printf("process-%d max need: ",i);

for(j=1;j<=r;j++)

{

scanf("%d",&max[i][j]);

}

}

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

{

for(j=1;j<=r;j++)

{

need[i][j]=max[i][j]-allocation[i][j];

}

}

printf("enter the available instances: ");

int k=1;

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

{

scanf("%d",&available[i]);

work[i]=available[i];

}

printf("enter number of the requesting process: ");

int c;

scanf("%d",&c);

int request[c],req\_arr[c][r];

printf("enter the requesting processes: ");

for(i=1;i<=c;i++){

scanf("%d",&request[i]);

}

printf("enter the process request instances: ");

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

{

printf("request of process %d:",request[i]);

for(j=1;j<=r;j++)

{

scanf("%d",&req\_arr[i][j]);

}

}

//resource request process alogorithem;

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

{

if(check2(i,request,req\_arr) && check(i,n,r,work))

{

sequence[k]=request[i];

k++;

for(j=1;j<=r;j++)

{

available[j]-=req\_arr[i][j];

allocation[request[i]][j]+=req\_arr[i][j];

need[request[i]][j]-=req\_arr[i][j];

}

for(j=1;j<=r;j++)

{

if(need[request[i]][j]!=0)

{

break;

}

}

if(j>r)

{

int t=1;

for(t=1;t<=r;t++)

{

available[t]+=allocation[request[i]][t];

}

finish[request[i]]=1;

l++;

}

}

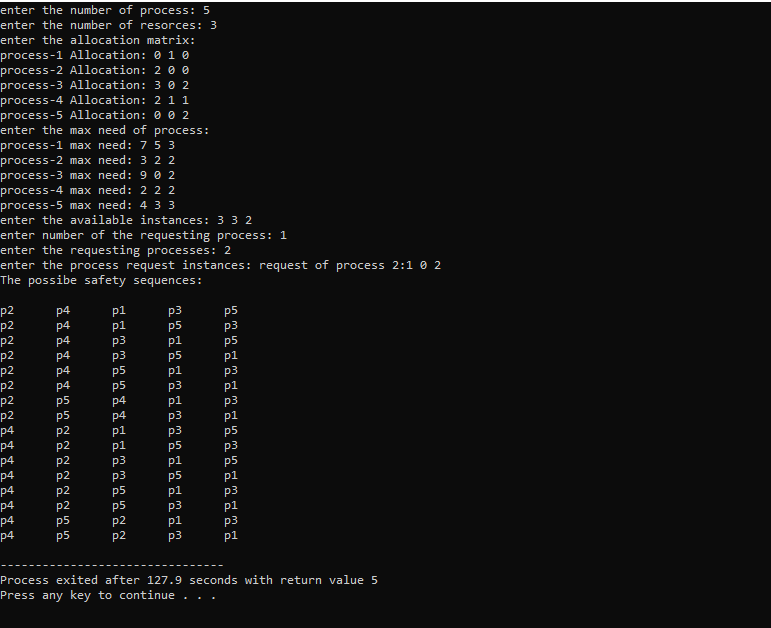
}

//here starts logic

printf("The possibe safety sequences: \n\n");

Safety\_sequence(sequence,finish,work,k);

}



**Page Replacement Algorithms**

1. **FIFO(FIRST IN FIRST OUT)**

#include<stdio.h>

int pagefound(int page,int frames[],int l)

{

int i;

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

{

if(frames[i]==page)

{

return 1;

}

}

return 0;

}

void print(int l,int frames[])

{

int i;

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

{

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

}

}

int main()

{

int n;

printf("enter the number of pages: ");

scanf("%d",&n);

int pages[n];

printf("enter the pages: \n");

int i;

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

{

scanf("%d",&pages[i]);

}

printf("enter the number of frames: ");

int f;

scanf("%d",&f);

int frames[f];

int empty=f;

int hit=0,fault=0;

int l=0,top=0;

printf("\n\n");

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

{

printf("%d Frames: ",pages[i]);

if(empty!=0)

{

if(pagefound(pages[i],frames,l)==0)

{

frames[l]=pages[i];

fault++;

l++;

empty--;

}

else

{

hit++;

}

}

else{

if(pagefound(pages[i],frames,l)==0)

{

int j;

frames[top]=pages[i];

fault++;

top=(top+1)%f;

}

else

{

hit++;

}

}

print(l,frames);

printf("\n");

}

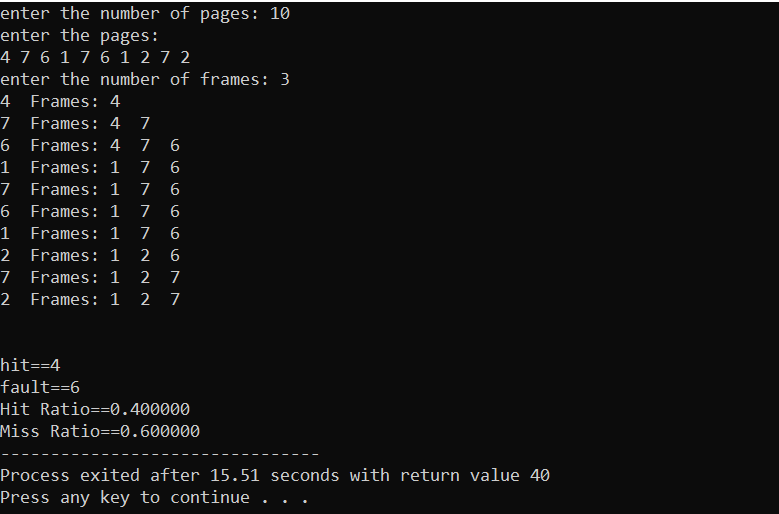
printf("\n\n");

printf("hit==%d\nfault==%d",hit,fault);

printf("\n");

printf("Hit Ratio==%f\nMiss Ratio==%f",(float)hit/n,(float)fault/n);

}



1. **OPTIMAL PAGE REPLACEMENT**

#include<stdio.h>

int pagefound(int page,int frames[],int l)

{

int i;

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

{

if(frames[i]==page)

{

return 1;

}

}

return 0;

}

void print(int l,int frames[])

{

int i;

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

{

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

}

}

int check(int i,int pages[],int frame,int n)

{

int j;

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

{

if(pages[j]==frame)

{

return j;

}

}

return 999;

}

int main()

{

int n;

printf("enter the number of pages: ");

scanf("%d",&n);

int pages[n];

printf("enter the pages: \n");

int i;

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

{

scanf("%d",&pages[i]);

}

printf("enter the number of frames: ");

int f;

scanf("%d",&f);

int frames[f];

int empty=f;

int hit=0,fault=0;

int l=0;

printf("\n\n");

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

{

printf("%d Frames: ",pages[i]);

if(empty!=0)

{

if(pagefound(pages[i],frames,l)==0)

{

frames[l]=pages[i];

fault++;

l++;

empty--;

}

else

{

hit++;

}

}

else{

if(pagefound(pages[i],frames,l)==0)

{

int visit[f],j;

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

{

visit[j]=check(i+1,pages,frames[j],n);

}

int max=visit[0],q=0;

for(j=1;j<f;j++)

{

if(max<visit[j])

{

max=visit[j];

q=j;

}

}

frames[q]=pages[i];

fault++;

}

else

{

hit++;

}

}

print(l,frames);

printf("\n");

}

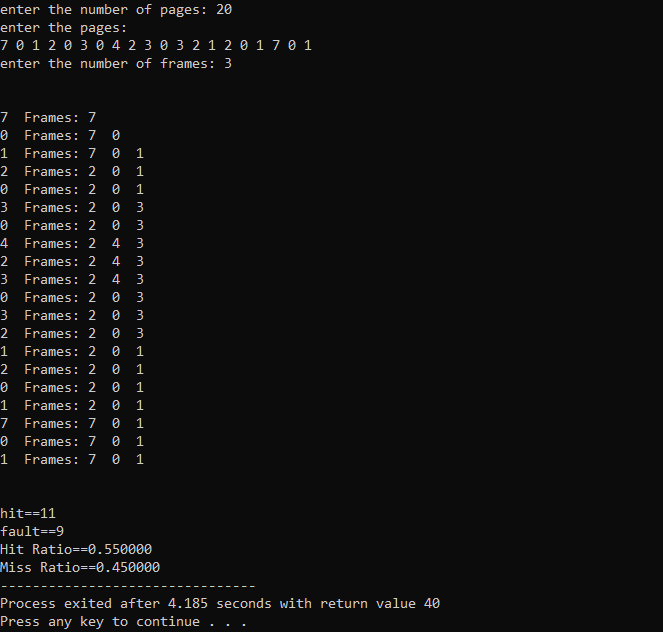
printf("\n\n");

printf("hit==%d\nfault==%d",hit,fault);

printf("\n");

printf("Hit Ratio==%f\nMiss Ratio==%f",(float)hit/n,(float)fault/n);

}



1. **LRU(LEAST RECENTLY USED)**

#include <stdio.h>

//user-defined function

int findLRU(int time[], int n)

{

int i, minimum = time[0], pos = 0;

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

{

if (time[i] < minimum)

{

minimum = time[i];

pos = i;

}

}

return pos;

}

//main function

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter the pages: ");

for (i = 0; i < no\_of\_pages; ++i)

{

scanf("%d", &pages[i]);

}

for (i = 0; i < no\_of\_frames; ++i)

{

frames[i] = -1;

}

printf("\n\n");

for (i = 0; i < no\_of\_pages; ++i)

{

printf("%d Frames: ",pages[i]);

flag1 = flag2 = 0;

for (j = 0; j < no\_of\_frames; ++j)

{

if (frames[j] == pages[i])

{

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if (flag1 == 0)

{

for (j = 0; j < no\_of\_frames; ++j)

{

if (frames[j] == -1)

{

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if (flag2 == 0)

{

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

for (j = 0; j < no\_of\_frames; ++j)

{

printf("%d ", frames[j]);

}

printf("\n");

}

int hit=no\_of\_pages-faults;

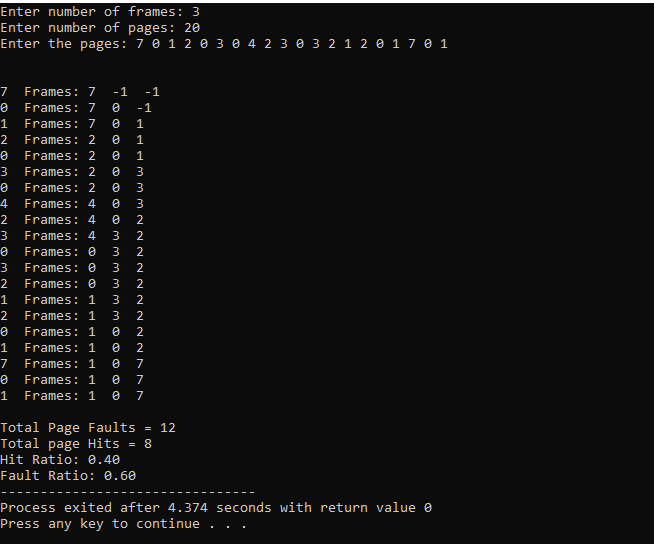
printf("\nTotal Page Faults = %d", faults);

printf("\nTotal page Hits = %d", hit);

printf("\nHitRatio: %.2f\nFault Ratio: %.2f",(float)hit/no\_of\_pages,(float)faults/no\_of\_pages);

return 0;

}



**DISK SCHEDULE ALGORITHM**

1. **FCFS(FIRST COME FIRST SERVE)**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n,i;

printf("enter the number of requests: ");

scanf("%d",&n);

int request[n];

printf("enter the requests: ");

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

{

scanf("%d",&request[i]);

}

int m;

printf("enter the position of readwrite track: ");

scanf("%d",&m);

printf("%d-->",m);

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

{

printf("%d-->",request[i]);

}

int THM=0;

THM=THM+abs(request[0]-m);

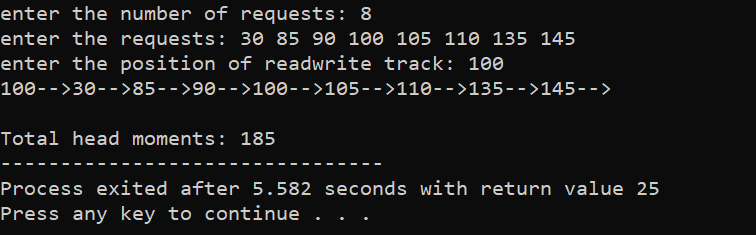
for(i=1;i<n;i++)

{

THM+=abs(request[i]-request[i-1]);

}

printf("\n\nTotal head moments: %d",THM);

}

1. **SCAN**

#include<stdio.h>

#include<stdlib.h>

int sort(int arr[],int n)

{

int i,j;

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

{

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

{

if(arr[i]>arr[j])

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

}

int main()

{

int n,i;

printf("enter the number of requests: ");

scanf("%d",&n);

int request[n+2];

printf("enter the requests: ");

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

{

scanf("%d",&request[i]);

}

request[0]=0;

int m;

printf("enter the position of readwrite track: ");

scanf("%d",&m);

int t;

printf("enter the total number of tracks: ");

scanf("%d",&t);

request[n+1]=t-1;

sort(request,n+2);

int index;

for(i=0;i<n+2;i++)

{

if(request[i]>m)

{

break;

}

}

index=i;

printf("enter the direction(R/L): ");

char c[1];

scanf("%s",c);

printf("%d-->",m);

for(i=index;i<n+2;i++)

{

printf("%d-->",request[i]);

}

for(i=index-1;i>0;i--)

{

printf("%d-->",request[i]);

}

int thm=0;

thm+=abs(request[index]-m);

for(i=index+1;i<n+2;i++)

{

thm+=abs(request[i]-request[i-1]);

}

thm+=abs(request[i-1]-request[index-1]);

for(i=index-2;i>0;i--)

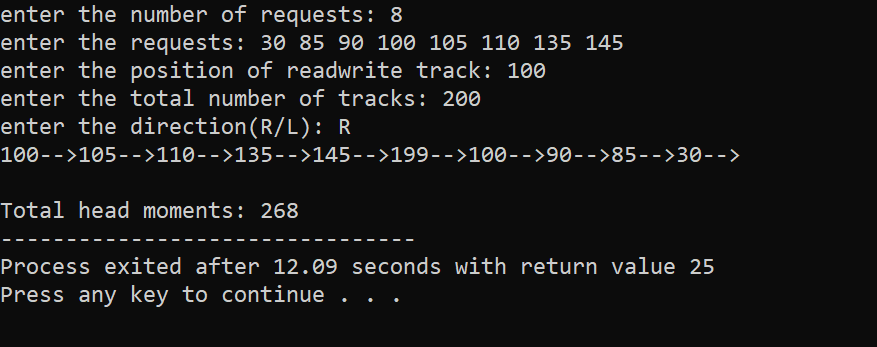
{

thm+=abs(request[i]-request[i+1]);

}

printf("\n\nTotal head moments: %d",thm);

}



1. **C-SCAN**

#include<stdio.h>

#include<stdlib.h>

int sort(int arr[],int n)

{

int i,j;

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

{

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

{

if(arr[i]>arr[j])

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

}

int main()

{

int n,i;

printf("enter the number of requests: ");

scanf("%d",&n);

int request[n+2];

printf("enter the requests: ");

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

{

scanf("%d",&request[i]);

}

request[0]=0;

int m;

printf("enter the position of readwrite track: ");

scanf("%d",&m);

int t;

printf("enter the total number of tracks: ");

scanf("%d",&t);

request[n+1]=t-1;

sort(request,n+2);;

int index;

for(i=0;i<n+2;i++)

{

if(request[i]>m)

{

break;

}

}

index=i;

printf("enter the direction(R/L): ");

char c[1];

scanf("%s",c);

printf("%d-->",m);

for(i=index;i<n+2;i++)

{

printf("%d-->",request[i]);

}

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

{

printf("%d-->",request[i]);

}

int thm=0;

thm+=abs(request[index]-m);

for(i=index+1;i<n+2;i++)

{

thm+=abs(request[i]-request[i-1]);

}

thm+=abs(request[i-1]-0);

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

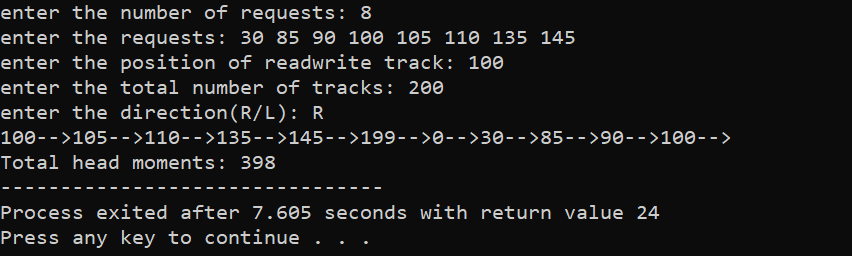
{

thm+=abs(request[i]-request[i+1]);

}

printf("\nTotal head moments: %d",thm);

}



**Multilevel Queue:**

#include<stdio.h>

int main()

{

int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;

float wtavg, tatavg;

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

scanf("%d",&n);

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

{

p[i] = i;

printf("Enter the Burst Time of Process%d:", i);

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

printf("System/User Process (0/1) ? ");

scanf("%d", &su[i]);

}

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

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

if(su[i] > su[k])

{

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=su[i];

su[i]=su[k];

su[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

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

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

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

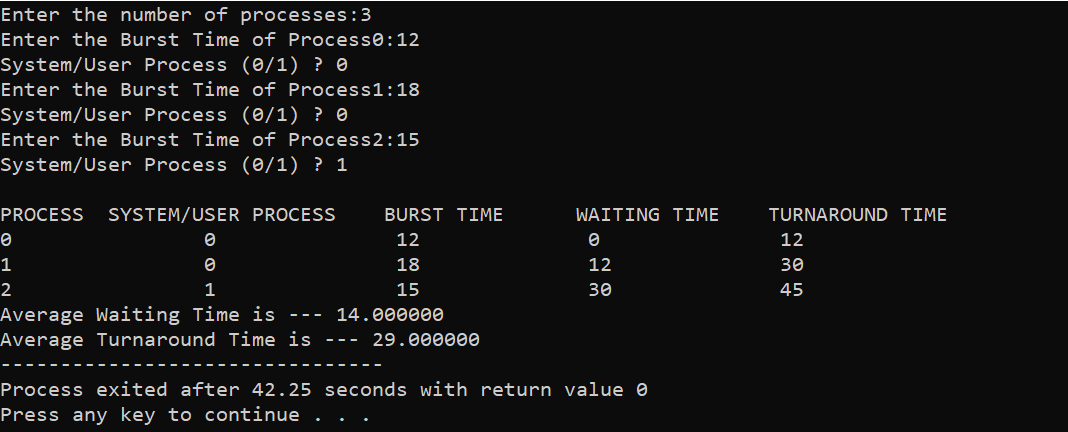
printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is --- %f",wtavg/n);

printf("\nAverage Turnaround Time is --- %f",tatavg/n);

return 0;

}



**Dining Philosoher:**

#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{

int taken;

}ForkAvil[n];

struct philosp{

int left;

int right;

}Philostatus[n];

void goForDinner(int philID){

if(Philostatus[philID].left==10 && Philostatus[philID].right==10)

printf("Philosopher %d completed his dinner\n",philID+1);

else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){

printf("Philosopher %d completed his dinner\n",philID+1);

Philostatus[philID].left = Philostatus[philID].right = 10;

int otherFork = philID-1;

if(otherFork== -1)

otherFork=(n-1);

ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;

printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);

compltedPhilo++;

}

else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){

if(philID==(n-1)){

if(ForkAvil[philID].taken==0){

ForkAvil[philID].taken = Philostatus[philID].right = 1;

printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);

}else{

printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);

}

}else{

int dupphilID = philID;

philID-=1;

if(philID== -1)

philID=(n-1);

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;

printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);

}else{

printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);

}

}

}

else if(Philostatus[philID].left==0){

if(philID==(n-1)){

if(ForkAvil[philID-1].taken==0){

ForkAvil[philID-1].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by philosopher %d\n",philID,philID+1);

}else{

printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);

}

}else{

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);

}else{

printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);

}

}

}else{}

}

int main(){

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

ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){

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

goForDinner(i);

printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);

}

return 0;

}

