1. **Write a program to create process.**

#include <stdio.h>

#include <unistd.h>

intmain()

{

fork();

printf("Operating system\n");

}

**output**



Manita silwal student

1. **Write a program to create process and print process id.**

#include <stdio.h>

#include <unistd.h>

intmain()

{

intpid;

pid=fork();

printf("BCA 4th semester\n");

if(pid==0)

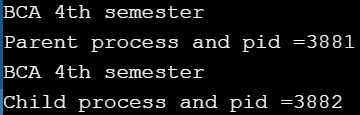
printf("Child process and pid =%d\n",getpid());

else

printf("Parent process and pid =%d\n",getpid());

}

**Output**



1. **Write a program to implement interprocess communication.**

#include <stdio.h>

#include <pthread.h>

// A normal C function that is executed as a thread

// when its name is specified in pthread\_create()

void \*thread1f(void \*arg);

void \*thread2f(void \*arg);

int turn=1;

int main()

{

pthread\_t thid1;

pthread\_t thid2;

pthread\_create(&thid1,NULL,&thread1f,NULL);

pthread\_create(&thid2,NULL,&thread2f,NULL);

pthread\_join(thid1,NULL);

pthread\_join(thid2,NULL);

return 0;

}

void \*thread1f(void \*arg)

{

int a=0;

while (a++ <20)

{

while(turn!=1);

fputc('b',stderr);

turn =0;

}

}

void \*thread2f(void \*arg)

{

int b=0;

while (b++ <20)

{

while(turn!=0);

fputc('a',stderr);

turn =1;

}

}

**Output**



1. **Write a program to simulate first come first serve (FCFS) process scheduling.**

#include<stdio.h>

void main()

{

intn,bt[20],wt[20],tat[20],i,j;

float avwt=0,avtat=0;

printf("Enter total number of processes(maximum 20) ");

scanf("%d",&n);

printf("Enter Process Burst Time\n");

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

{

printf("P[%d]: ",i+1);

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

}

wt[0]=0; //waiting time for first process is 0

for(i=1;i<n;i++) //calculating waiting time

{

wt[i]=0;

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

{

wt[i]=wt[i]+bt[j];

}

}

printf("\nProcess\tBurstTime\tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++) //calculating turnaround time

{

tat[i]=bt[i]+wt[i];

avwt=avwt+wt[i];

avtat=avtat+tat[i];

printf("\nP[%d]\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

avwt=avwt/i;

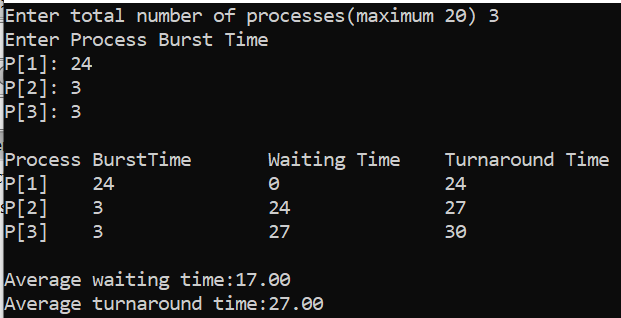
avtat=avtat/i;

printf("\n\nAverage waiting time:%.2f",avwt);

printf("\nAverage turnaround time:%.2f",avtat);

}

**Output**



1. **Write a program to simulate shortest job first (SJF).**

#include<stdio.h>

void main()

{

intn,bt[20],wt[20],tat[20],i,j,p[20],temp;

float avwt=0,avtat=0;

printf("Enter total number of processes(maximum 20) ");

scanf("%d",&n);

printf("Enter Process Burst Time\n");

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

{

printf("P[%d]: ",i+1);

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

p[i]=i+1; //contains process number

}

//sorting burst time in ascending order

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

{

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

{

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

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

wt[0]=0; //waiting time for first process is 0

for(i=1;i<n;i++) //calculating waiting time

{

wt[i]=0;

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

{

wt[i]=wt[i]+bt[j];

}

}

printf("\nProcess\tBurstTime\tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++) //calculating turnaround time

{

tat[i]=bt[i]+wt[i];

avwt=avwt+wt[i];

avtat=avtat+tat[i];

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

}

avwt=avwt/i;

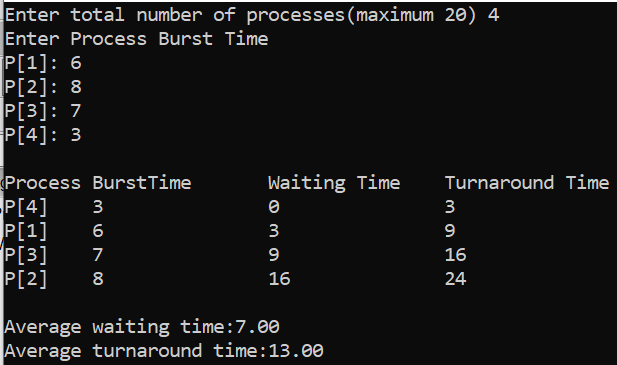
avtat=avtat/i;

printf("\n\nAverage waiting time:%.2f",avwt);

printf("\nAverage turnaround time:%.2f",avtat);

}

**Output**



1. **Write a program to simulate priority scheduling.**

#include<stdio.h>

void main()

{

intn,bt[20],wt[20],tat[20],p[20],pri[20],i,j,temp;

float avwt=0,avtat=0;

printf("Enter total number of processes(maximum 20) ");

scanf("%d",&n);

printf("Enter Process Burst Time and Priority of Process\n");

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

{

p[i]=i;

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

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

}

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

{

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

{

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

{

temp=p[i];

p[i]=p[j];

p[j]=temp;

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=pri[i];

pri[i]=pri[j];

pri[j]=temp;

}

}

}

wt[0]=0; //waiting time for first process is 0

for(i=1;i<n;i++) //calculating waiting time

{

wt[i]=0;

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

{

wt[i]=wt[i]+bt[j];

}

}

printf("\nProcess\tPriority\tBurstTime\tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++) //calculating turnaround time

{

tat[i]=bt[i]+wt[i];

avwt=avwt+wt[i];

avtat=avtat+tat[i];

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

}

avwt=avwt/i;

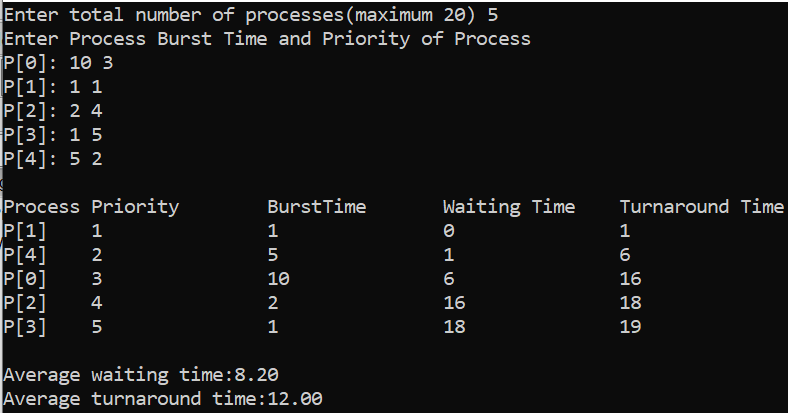
avtat=avtat/i;

printf("\n\nAverage waiting time:%.2f",avwt);

printf("\nAverage turnaround time:%.2f",avtat);

}

**Output**



1. **Write a program to simulate Round Robin (RR) scheduling.**

#include<stdio.h>

void main()

{

inti,j,n,bt[10],wt[10],tat[10],t,ct[10],max;

float avwt=0,avtat=0,temp=0;

printf("Enter total number of processes(maximum 20) ");

scanf("%d",&n);

printf("Enter Process Burst Time\n");

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

{

printf("P[%d]: ",i+1);

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

ct[i]=bt[i];

}

printf("Enter the size of time slice ");

scanf("%d",&t);

max=bt[0];

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

if(max<bt[i])

max=bt[i];

for(j=0;j<(max/t)+1;j++)

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

if(bt[i]!=0)

if(bt[i]<=t)

{

tat[i]=temp+bt[i];

temp=temp+bt[i];

bt[i]=0;

}

else

{

bt[i]=bt[i]-t;

temp=temp+t;

}

printf("\nProcess\tBurstTime\tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++) //calculating turnaround time

{

wt[i]=tat[i]-ct[i];

avtat=avtat+tat[i];

avwt=avwt+wt[i];

printf("\nP[%d]\t%d\t\t%d\t\t%d",i+1,ct[i],wt[i],tat[i]);

}

avwt=avwt/i;

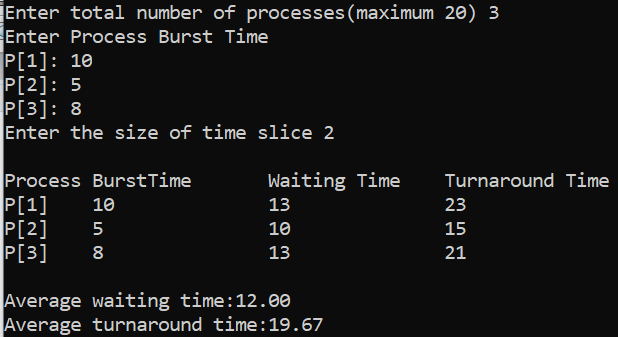
avtat=avtat/i;

printf("\n\nAverage waiting time:%.2f",avwt);

printf("\nAverage turnaround time:%.2f",avtat);

}

**Output**



1. **Write a program to simulate producer-consumer problem using semaphores.**

#include<stdio.h>

void main()

{

intbuffer[10],bufsize,in,out,produce,consume,choice;

in=0,out=0,bufsize=10;

while(choice!=3)

{

printf("\n1. Produce");

printf("\n2. Consume");

printf("\n3. Exit");

printf("\n\nEnter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

if((in+1)%bufsize==out)

printf("\nBuffer is full");

else

{

printf("Enter the value ");

scanf("%d",&produce);

buffer[in]=produce;

in=(in+1)%bufsize;

}

break;

case 2:

if(in==out)

printf("\nBuffer is Empty");

else

{

consume=buffer[out];

printf("\nThe consumed value is %d",consume);

out=(out+1)%bufsize;

}

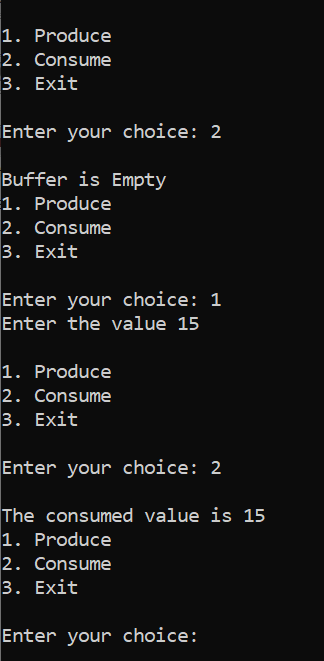
break;

}

}

}

**Output**



1. **Write a program to simulate Banker’s algorithm for the purpose of deadlock avoidance.**

// Banker's Algorithm

#include <stdio.h>

intmain()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k;

n = 5; // Number of processes

m = 3; // Number of resources

intalloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

intmax[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

intavail[3] = { 3, 3, 2 }; // Available Resources

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

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

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

printf("Following is the SAFE Sequence\n");

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

{

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

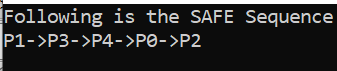
}

printf("P%d\n",ans[n-1]);

return (0);

}

**Output**



1. **Write a program to simulate first fit algorithm.**

#include<stdio.h>

void main()

{

intbsize[10],psize[10],bno,pno,flags[10],allocation[10],i,j;

for(i=0;i<10;i++)

{

flags[i]=0;

allocation[i]=-1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: \n");

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

{

printf("Block[%d]: ",i+1);

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

}

printf("\nEnter no. of processes: \n");

scanf("%d",&pno);

printf("\nEnter size of process: ");

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

{

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

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

}

for(i=0;i<pno;i++) //allocation as per first fit

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

if(flags[j]==0&&bsize[j]>=psize[i])

{

allocation[j]=i;

flags[j]=1;

break;

}

//display allocation details

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

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

{

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

if(flags[i]==1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

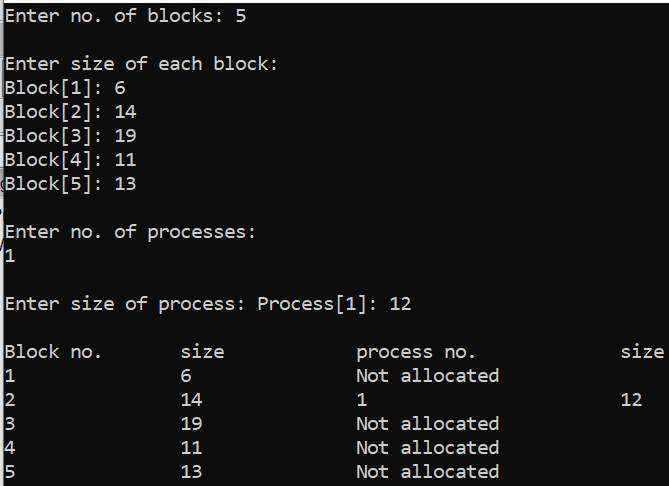
else

printf("Not allocated");

}

}

**Output**



1. **Write a program to simulate best fit algorithm**

#include<stdio.h>

void main()

{

int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;

static intbarray[20],parray[20];

printf("\n\t\t\tMemory Management Scheme - Best Fit");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

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

scanf("%d",&np);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block no.%d:",i);

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

}

printf("\nEnter the size of the processes :-\n");

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

{

printf("Process no.%d:",i);

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

}

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

{

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

{

if(barray[j]!=1)

{

temp=b[j]-p[i];

if(temp>=0)

if(lowest>temp)

{

parray[i]=j;

lowest=temp;

}

}

}

fragment[i]=lowest;

barray[parray[i]]=1;

lowest=10000;

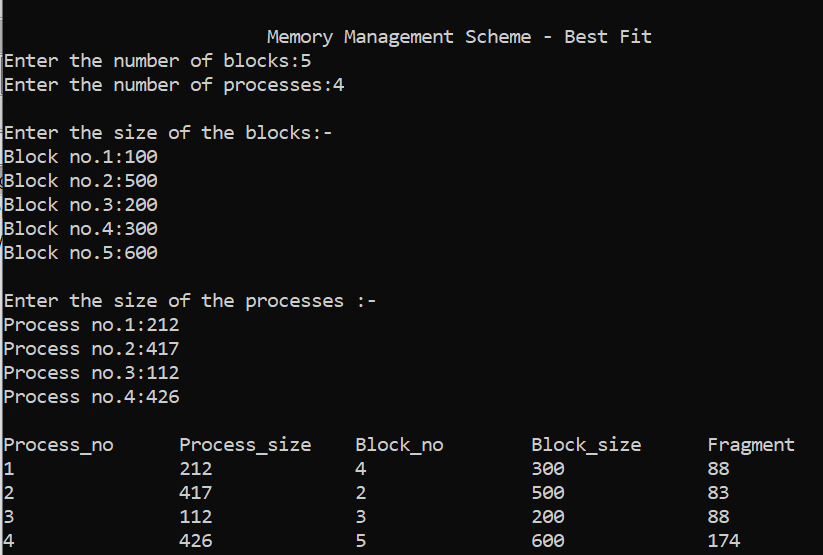
}

printf("\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment");

for(i=1;i<=np &&parray[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);

}



1. **Write a program to implement first come first serve (FCFS) disk scheduling algorithm.**

#include<stdio.h>

intmain()

{

inti,j,sum=0,n;

intar[20],tm[20];

int disk;

printf("enter number of location\t");

scanf("%d",&n);

printf("enter position of head\t");

scanf("%d",&disk);

printf("enter elements of disk queue\n");

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

{

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

tm[i]=disk-ar[i];

if(tm[i]<0)

{

tm[i]=ar[i]-disk;

}

disk=ar[i];

sum=sum+tm[i];

}

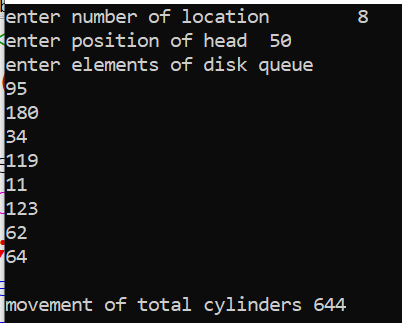
printf("\nmovement of total cylinders %d",sum);

getch();

return 0;

}

**Output**



1. **Write a program to implement shortest seek time first (SSTF) disk scheduling algorithm.**

#include<conio.h>

#include<stdio.h>

struct di

{

intnum;

int flag;

};

intmain()

{

inti,j,sum=0,n,min,loc,x,y;

struct di d[20];

int disk;

intar[20],a[20];

printf("enter number of location\t");

scanf("%d",&n);

printf("enter position of head\t");

scanf("%d",&disk);

printf("enter elements of disk queue\n");

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

{

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

d[i]. flag=0;

}

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

{ x=0; min=0;loc=0;

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

{

if(d[j].flag==0)

{

if(x==0)

{

ar[j]=disk-d[j].num;

if(ar[j]<0){ar[j]=d[j].num-disk;

}

min=ar[j];loc=j;x++;

}

else

{

ar[j]=disk-d[j].num;

if(ar[j]<0)

{

ar[j]=d[j].num-disk;

}

}

if(min>ar[j])

{

min=ar[j]; loc=j;

}

}

}

d[loc].flag=1;

a[i]=d[loc].num-disk;

if(a[i]<0)

{

a[i]=disk-d[loc].num;

}

disk=d[loc].num;

}

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

{

sum=sum+a[i];

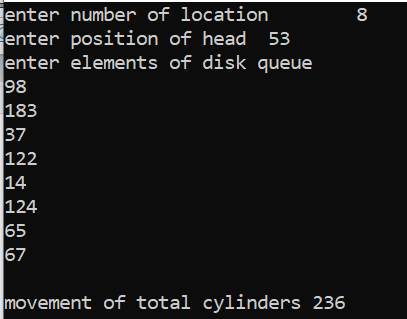
}

printf("\nmovement of total cylinders %d",sum);

return 0;

}

**Output**



1. **Write a program to implement sequential file allocation.**

#include<stdio.h>

void main()

{

intf[50],i,st,j,len,c,k;

for(i=0;i<50;i++)

f[i]=0;

X:

printf("\n Enter the starting block & length of file ");

scanf("%d%d",&st,&len);

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1;

printf("\n%d->%d",j,f[j]);

}

else

{

printf("Block already allocated");

break;

}

if(j==(st+len))

printf("\n\n The file is allocated to disk");

printf("\n If you want to enter more files?(y-1/n-0) ");

scanf("%d",&c);

if(c==1)

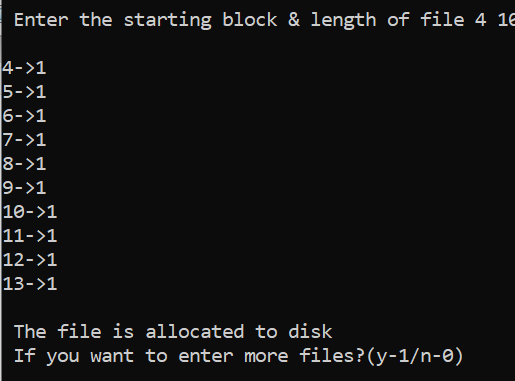
goto X;

else

exit(0);

}

**Output**



1. **Write a program to implement linked file allocation technique.**

#include<stdio.h>

void main()

{

intf[50],p,i,j,k,a,st,len,n,c;

for(i=0;i<50;i++)

f[i]=0;

printf("Enter how many blocks that are already allocated ");

scanf("%d",&p);

printf("\nEnter the blocks no.s that are already allocated ");

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

{

scanf("%d",&a);

f[a]=1;

}

X:

printf("Enter the starting index block & length ");

scanf("%d%d",&st,&len);

k=len;

for(j=st;j<(k+st);j++)

{

if(f[j]==0)

{

f[j]=1;

printf("\n%d->%d",j,f[j]);

}

else

{

printf("\n %d->file is already allocated",j);

k++;

}

}

printf("\n If u want to enter one more file? (yes-1/no-0)");

scanf("%d",&c);

if(c==1)

goto X;

else

exit(0);

}

**Output**

