**PROGRAM**

echo "Enter a number"

read n

if [ `expr $n % 2` -eq 0 ]

then

echo "number is even"

else

echo "number is odd"

fi

**OUTPUT**

Enter a number

2

number is even

**PROGRAM**

echo "Enter two numbers"

read a b

if [ $a -gt $b ]

then

echo "$a is the largest"

else

echo "$b is the largest"

fi

**OUTPUT**

Enter two numbers

3 5

5 is the largest

**PROGRAM**

echo "Enter the number"

read num

fact=1

while [ $num -gt 1 ]

do

fact=$((fact\*num))

num=$((num-1))

done

echo "Factorial is $fact"

**OUTPUT**

Enter the number

4

Factorial is 24

**PROGRAM**

echo "Enter the number of terms"

read n

a=0

b=1

echo "Fibonacci series is : "

echo $a

echo $b

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

do

next=$((a+b))

echo $next

a=$b

b=$next

done

**OUTPUT**

Enter the number of terms

5

Fibonacci series is :

0 1 1 2 3

**PROGRAM**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

pid\_t p;

p = fork();

if(p==-1)

{ printf("There is an error while calling fork()"); }

if(p==0)

{ printf("\nWe are in the child process"); }

else

{ printf("\nWe are in the parent process"); }

return 0;

}

**OUTPUT**

We are in the parent process

**PROGRAM**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int pid,status,exitch;

if((pid = fork()) == -1)

{

perror("Error!!");

exit(0);

}

if(pid == 0)

{

sleep(3);

printf("\n Child process");

exit(0);

}

else

{

printf("\n Parent process");

if((exitch = wait(&status))==1)

{

perror("Duringwait");

exit(0);

}

printf("\n Parent exiting!!");

exit(0);

}

return 0;

}

**OUTPUT**

Child process Parent process

Parent exiting!!

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

int main(void)

{

char \*path1[10];

struct stat \*nfile;

nfile=(struct stat \*)malloc(sizeof(struct stat));

printf("Enter name of file whose statistics has to be printed: ");

scanf("%s",&path1);

stat(path1,nfile);

printf("User ID: %d \n", nfile->st\_uid);

printf("Block Size: %d \n", nfile->st\_blksize);

printf("Last Access Time: %d \n", nfile->st\_atime);

printf("Production Mode: %d \n", nfile->st\_mode);

printf("Size of file: %d \n", nfile->st\_nlink);

}

**OUTPUT**

Enter the name of the file whose statistics have to be printed: stat.c

User ID: 621

Block size: 4096

Last access time: 1145148485

Production mode: 33204

Size of file: 654

**PROGRAM**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int pid;

char\*args[] = {"/bin/ls","-l",0};

printf("\n Parent process ");

pid =fork();

if (pid ==0)

{

execv("/bin/ls",args);

printf("\n Child Process");

}

else

{

printf("\n Parent Process");

exit(0);

}

}

**OUTPUT**

Parent process

Parent Process

**PROGRAM**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<stdlib.h>

int main()

{

int pid;

pid=getpid();

printf("%d",pid);

pid=getppid();

printf("%d",pid);

}

**OUTPUT**

PID : 3798

PPID : 3797

**PROGRAM**

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

struct dirent \*dptr;

int main(int argc,char \*argv[])

{

char buff[256];

DIR \*dirp;

printf("\n Enter directory name :");

scanf("%s",buff);

if((dirp=opendir(buff))==NULL)

{

printf("Error");

exit(1);

}

while((dptr=readdir(dirp)) !=NULL)

{ printf("%s\n",dptr->d\_name); }

closedir(dirp);

}

**OUTPUT**

Test

T1 T2 T3 T4

**PROGRAM**

#include <stdio.h>

intp[30],bt[30],tot\_tat=0,wt[30],n,tot\_wt=0,tat[30],fcfs\_wt=0,fcfs\_tat=0;

float awt,avg\_tat,avg\_wt;

int wt\_tat(int \*a,int \*b);

int main()

{

int i;

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

scanf("%d",&n);

printf("Enter burst time for each process :\n");

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

{

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

p[i]=i;

}

printf("\nFCFS algorithm\n");

wt\_tat(&fcfs\_tat,&fcfs\_wt);

printf("\n\nTotal turn around time = %d",fcfs\_tat);

printf("\n\nAverage turn around time = %d",fcfs\_tat/n);

printf("\n\nTotal Waiting time = %d",fcfs\_wt);

printf("\n\nAverage Waiting time = %d",fcfs\_wt/n);

}

int wt\_tat(int \*a,int \*b)

{

int i;

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

{

if(i==0)

tat[i]=bt[i];

else

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

tot\_tat=tot\_tat+tat[i];

}

\*a=tot\_tat;

wt[0]=0;

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

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

tot\_wt=tot\_wt+wt[i]; }

\*b=tot\_wt;

printf("\nProcess\t\tBurst Time\tTurnaround time\tWaiting time");

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

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

return 0;

}

**OUTPUT**

Enter the number of processes :

5

Enter burst time for each process :

9 3 8 4 2

FCFS algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Turnaround time | Waiting time |
| process(0) | 9 | 9 | 0 |
| process(1) | 3 | 12 | 9 |
| process(2) | 8 | 20 | 12 |
| process(3) | 4 | 24 | 20 |
| process(4) | 2 | 26 | 24 |

Total turn around time = 91

Average turn around time = 18

Total Waiting time = 65

Average Waiting time = 13

**PROGRAM**

#include <stdio.h>

int p[30],bt[30],tot\_tat=0,wt[30],n,tot\_wt=0,tat[30],sjf\_wt=0,sjf\_tat=0;

float awt,avg\_tat,avg\_wt;

int wt\_tat(int \*a,int \*b);

int swap(int \*a,int \*b);

int sort();

int main()

{

int i;

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

scanf("%d",&n);

printf("Enter burst time for each process :\n");

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

{

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

p[i]=i;

}

sort();

wt\_tat(&sjf\_tat,&sjf\_wt);

printf("\n\nTotal turn around time = %d",sjf\_tat);

printf("\n\nAverage turn around time = %d",sjf\_tat/n);

printf("\n\nTotal Waiting time = %d",sjf\_wt);

printf("\n\nAverage Waiting time = %d",sjf\_wt/n);

}

int sort()

{

int i,j;

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

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

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

{

swap(&bt[j],&bt[i]);

swap(&p[j],&p[i]);

}

}

}

return 0;

}

int swap(int \*a,int \*b)

{

int t;

t=\*a;

\*a=\*b;

\*b=t;

}

int wt\_tat(int \*a,int \*b)

{

int i;

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

{

if(i==0)

tat[i]=bt[i];

else

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

tot\_tat=tot\_tat+tat[i];

}

\*a=tot\_tat;

wt[0]=0;

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

{

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

tot\_wt=tot\_wt+wt[i];

}

\*b=tot\_wt;

printf("\nProcess\t\tBurst Time\tTurnaround time\t\tWaiting time");

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

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

return 0;}

**OUTPUT**

Enter the number of processes :

5

Enter burst time for each process :

9 3 8 4 2

SJF ALGORITHM

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Turnaround time | Waiting time |
| process(4) | 2 | 2 | 0 |
| process(1) | 3 | 5 | 2 |
| process(3) | 4 | 9 | 5 |
| process(2) | 8 | 17 | 9 |
| process(0) | 9 | 26 | 17 |

Total turn around time = 59

Average turn around time = 11

Total Waiting time = 33

Average Waiting time = 6

**PROGRAM**

#include<stdio.h>

int main()

{ int count,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

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

{printf("Enter AT and BT for Process Number %d :",count+1);

scanf("%d",&at[count]);

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

rt[count]=bt[count]; }

printf("Enter Time Quantum:\t");

scanf("%d",&time\_quantum);

printf("\n\nProcess\t | Turnaround Time | Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{ rt[count]-=time\_quantum;

time+=time\_quantum; }

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t\t%d\t\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

**OUTPUT**

Enter Total Process: 5

Enter AT and BT for Process Number 1 :4 7

Enter AT and BT for Process Number 2 :2 5

Enter AT and BT for Process Number 3 :4 6

Enter AT and BT for Process Number 4 :7 3

Enter AT and BT for Process Number 5 :2 9

Enter Time Quantum: 2

ROUND ROBIN ALGORITHM

Process | Turnaround Time | Waiting Time

P[2] 17 12

P[3] 17 11

P[4] 15 12

P[1] 21 14

P[5] 28 19

Average Waiting Time= 13.600000

Average Turnaround Time = 19.600000

**PROGRAM**

#include<stdio.h>

#define max 10

int main()

{int i,j,n,bt[max],p[max],wt[max],tat[max],pr[max],total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority For ");

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

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

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

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

p[i]=i+1; }

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

{pos=i;

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

{if(pr[j]<pr[pos])

pos=j;}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;}

wt[0]=0;

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

{wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];}

avg\_wt=total/n;

total=0;

printf("\n\nProcess\t\tBT\t\tWT\t\tTAT");

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

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

total+=tat[i];

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

avg\_tat=total/n;

printf("\n\nAverage Waiting Time = %.2f",avg\_wt);

printf("\nAvg Turn Around Time = %.2f\n",avg\_tat);

return 0;}

**OUTPUT**

Enter Total Number of Process:5

Enter Burst Time and Priority For

Enter Process 1: 5 9

Enter Process 2: 7 6

Enter Process 3: 1 5

Enter Process 4: 9 6

Enter Process 5: 8 5

PRIORITY SCHEDULING ALGORITHM

Process BT WT TAT

P3 1 0 1

P5 8 1 9

P4 9 9 18

P2 7 18 25

P1 5 25 30

Average Waiting Time = 10.00

Average Turn Around Time = 16.00

**PROGRAM**

[writer.c]

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

int main()

{

key\_t key=15678;

int shmid = shmget(key,1024,0666|IPC\_CREAT);

char\*str = (char\*)shmat(shmid,0,0);

printf("enter data to write:");

gets(str);

printf("Data written from the memory :%s\n",str);

shmdt(str);

return 0; }

[reader.c]

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

int main()

{ key\_t key=15678;

int shmid = shmget(key,1024,0666|IPC\_CREAT);

char\*str = (char\*)shmat(shmid,0,0);

printf("Data read from the memory :%s\n",str);

return 0; }

**OUTPUT**

[writer.c]

enter data to write:sneha

Data written from the memory :sneha

[reader.c]

Data read from the memory :sneha

**PROGRAM**

#include <stdio.h>

int main()

{int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix : ");

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

{printf("\nFor process %d : ", i + 1);

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

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

printf("\n\nEnter the allocation for each process : ");

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

{printf("\nFor process %d : ",i + 1);

for(j = 0; j < r; j++)scanf("%d", &alloc[i][j]);}

printf("\n\nEnter the Available Resources : ");

for(i = 0; i < r; i++) scanf("%d", &avail[i]);

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

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

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

do{printf("\n Max matrix:\tAllocation matrix:\n");

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

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

printf("%d ", Max[i][j]);

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

printf("%d ", alloc[i][j]);}

process = -1;

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

{if(completed[i] == 0)

{process = i ;

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

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

{process = -1;

break;}}}

if(process != -1)

break;}

if(process != -1)

{printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

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

{avail[j] += alloc[process][j];

alloc[process][j] = 0;Max[process][j] = 0;

completed[process] = 1;}}}

while(count != p && process != -1);

if(count == p)

{ printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]);}else

printf("\nThe system is in an unsafe state!!"); }

**OUTPUT**

Enter the no of processes : 5

Enter the no of resources : 4

Enter the Max Matrix :

For process 1 : 0 2 1 0

For process 2 : 1 6 5 2

For process 3 : 2 3 6 6

For process 4 : 0 6 5 2

For process 5 : 0 6 5 6

Enter the allocation for each process :

For process 1 : 0 1 1 0

For process 2 : 1 2 3 1

For process 3 : 1 3 6 5

For process 4 : 0 6 3 2

For process 5 : 0 0 1 4

Enter the Available Resources : 1 5 2 0

Max matrix: Allocation matrix:

0 2 1 0 0 1 1 0

1 6 5 2 1 2 3 1

2 3 6 6 1 3 6 5

0 6 5 2 0 6 3 2

0 6 5 6 0 0 1 4

Process 1 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0 0 0

1 6 5 2 1 2 3 1

2 3 6 6 1 3 6 5

0 6 5 2 0 6 3 2

0 6 5 6 0 0 1 4

Process 4 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0 0 0

1 6 5 2 1 2 3 1

2 3 6 6 1 3 6 5

0 0 0 0 0 0 0 0

0 6 5 6 0 0 1 4

Process 2 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

2 3 6 6 1 3 6 5

0 0 0 0 0 0 0 0

0 6 5 6 0 0 1 4

Process 3 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 6 5 6 0 0 1 4

Process 5 runs to completion!

The system is in a safe state!!

Safe Sequence: < 1 4 2 3 5 >

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

**OUTPUT**

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:1

Producer produces the item 3

Enter your choice:1

Buffer is full!!

Enter your choice:2

Consumer consumes item 3

Enter your choice:2

Consumer consumes item 2

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

**PROGRAM**

#include<stdio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000,highest=0;

static int bf[max],ff[max];

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

scanf("%d",&nb);

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

scanf("%d",&nf);

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

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

printf("\n BEST FIT ALGORITHM");

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

{

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

{

if(bf[j]!=1)

{

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

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}

}

frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");

for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

printf("\n FIRST FIT ALGORITHM");

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

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

{if(bf[j]!=1) //if bf[j] is not allocated

{

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

if(temp>=0)

if(highest<temp)

{

ff[i]=j;

highest=temp;

}

}

}

frag[i]=highest;

bf[ff[i]]=1;

highest=0;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

printf("\n WORST FIT ALGORITHM");

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

{

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

{

if(bf[j]!=1)

{

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

if(temp>=0)

{

ff[i]=j;

break;

}

}

}

frag[i]=temp;

bf[ff[i]]=1;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);}

**OUTPUT**

Enter the number of blocks:3

Enter the number of files:2

Enter the size of the blocks:-

Block 1:5

Block 2:2

Block 3:7

Enter the size of the files :-

File 1:1

File 2:4

BEST FIT ALGORITHM

File No File Size Block No Block Size Fragment

1 1 2 2 1

2 4 1 5 1

FIRST FIT ALGORITHM

File\_no: File\_size : Block\_no: Block\_size: Fragement

1 1 3 7 6

2 4 1 5 0

WORST FIT ALGORITHM

File\_no: File\_size : Block\_no: Block\_size: Fragement

1 1 3 7 6

2 4 1 5 6

**PROGRAM**

// FIFO ALGORITHM

#include<stdio.h>

int main()

{int i,j,n,a[50],frame[10],no,k,avail,count=0;

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

scanf("%d",&n);

printf("\n enter the page number:\n");

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

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

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

scanf("%d",&no);

for(i=0;i<no;i++)

frame[i]=-1;

j=0;

printf("\tref string\t page frames\n");

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

{printf("%d\t\t",a[i]);

avail=0;

for(k=0;k<no;k++)

if(frame[k]==a[i])

avail=1;

if(avail==0)

{ frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

printf("%d\t",frame[k]);}

printf("\n");}

printf("page fault is %d",count);

return 0; }

**OUTPUT**

enter the number of pages:

4

enter the page number:

2

6

8

9

3

enter the number of frames:

5

ref string page frames

2 2 -1 -1 -1 -1

6 2 6 -1 -1 -1

8 2 6 8 -1 -1

3 2 6 8 3 -1

page fault is 4

**PROGRAM**

// LRU ALGO

#include<stdio.h>

void main()

{int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];

printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:");

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

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

printf("Enter no of frames:");

scanf("%d",&f);

q[k]=p[k];

printf("\n\t%d\n",q[k]);

c++;

k++;

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

{c1=0;

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

{if(p[i]!=q[j])

c1++;}

if(c1==f)

{c++;

if(k<f)

{q[k]=p[i];

k++;

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

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

printf("\n");

}else

{for(r=0;r<f;r++)

{c2[r]=0;

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

{if(q[r]!=p[j])

c2[r]++;

else

break; }}

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

b[r]=c2[r];

for(r=0;r<f;r++) {

for(j=r;j<f;j++) {

if(b[r]<b[j]) {

t=b[r];

b[r]=b[j];

b[j]=t; }}}

for(r=0;r<f;r++) {

if(c2[r]==b[0])

q[r]=p[i];

printf("\t%d",q[r]);}

printf("\n");}}}

printf("\nThe no of page faults is %d",c);

}

**OUTPUT**

Enter no of pages:4

Enter the reference string:2 6 8 9 3

Enter no of frames:

2

2 6

2 6 8

9 6 8

The no of page faults is 4

**PROGRAM**

//LFU ALGO

#include<stdio.h>

int main()

{int f,p;

int pages[50],frame[10],hit=0,count[50],time[50];

int i,j,page,flag,least,minTime,temp;

printf("Enter no of frames : ");

scanf("%d",&f);

printf("Enter no of pages : ");

scanf("%d",&p);

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

{frame[i]=-1;}

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

{count[i]=0;}

printf("Enter page no : \n");

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

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

printf("\n");

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

count[pages[i]]++;

time[pages[i]]=i;

flag=1;

least=frame[0];

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

if(frame[j]==

-1 || frame[j]==pages[i])

{if(frame[j]!=

-1)

{hit++; }

flag=0;

frame[j]=pages[i];

break; }

if(count[least]>count[frame[j]]) {

least=frame[j]; }}

if(flag) {

minTime=50;

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

if(count[frame[j]]==count[least] && time[frame[j]]<minTime) {

temp=j;

minTime=time[frame[j]]; }}

count[frame[temp]]=0;

frame[temp]=pages[i]; }

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

printf("%d ",frame[j]); }

printf("\n");

}

printf("Page hit = %d",hit);

return 0;

}

**OUTPUT**

Enter no of frames : 5

Enter no of pages : 4

Enter page no :

2 6 8 9 3

2 -1 -1 -1 -1

2 6 -1 -1 -1

2 6 8 -1 -1

2 6 8 9 -1

Page hit = 0

**PROGRAM**

//FCFS disk scheduling

#include<stdio.h>

int main()

{

int queue[20],n,head,i,j,k,seek=0,max,diff;

float avg;

printf("Enter the max range of disk\n");

scanf("%d",&max);

printf("Enter the size of queue request\n");

scanf("%d",&n);

printf("Enter the queue of disk positions to be read\n");

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

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

printf("Enter the initial head position\n");

scanf("%d",&head);

queue[0]=head;

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

{

diff=abs(queue[j+1]-queue[j]);

seek+=diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("Total seek time is %d\n",seek);

avg=seek/(float)n;

printf("Average seek time is %f\n",avg);

return 0;

}

**OUTPUT**

Enter the max range of disk

200

Enter the size of queue request

8

Enter the queue of disk positions to be read

90 120 35 122 38 128 65 68

Enter the initial head position

50

Disk head moves from 50 to 90 with seek 40

Disk head moves from 90 to 120 with seek 30

Disk head moves from 120 to 35 with seek 85

Disk head moves from 35 to 122 with seek 87

Disk head moves from 122 to 38 with seek 84

Disk head moves from 38 to 128 with seek 90

Disk head moves from 128 to 65 with seek 63

Disk head moves from 65 to 68 with seek 3

Total seek time is 482

Average seek time is 60.250000

**PROGRAM**

//SCAN disk scheduling

#include main()

{ int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0; printf("enter the no of tracks to be traveresed"); scanf("%d'",&n);

printf("enter the position of head");

scanf("%d",&h);

t[0]=0;t[1]=h;

printf("enter the tracks");

for(i=2;

it[j+1]) { temp=t[j];

t[j]=t[j+1];

t[j+1]=temp; } } }

for(i=0;iatr[j+1])

d[j]=atr[j]-atr[j+1];

else d[j]=atr[j+1]-atr[j];

sum+=d[j]; }

printf("\nAverage header movements:%f",(float)sum/n); getch(); }

**OUTPUT**

Enter no.of tracks:9

Enter track position:55 58 60 70 18 90 150 160 184

Tracks traversed Difference between tracks

150 50

160 10

184 24

90 94

70 20

60 10

58 2

55 3

18 37

Average header movements: 27.77

**PROGRAM**

//CSCAN disk scheduling

#include main()

{ int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;

clrscr();

printf("enter the no of tracks to be traveresed"); scanf("%d'",&n);

printf("enter the position of head");

scanf("%d",&h); t[0]=0;t[1]=h;

printf("enter total tracks");

scanf("%d",&tot);

t[2]=tot-1;

printf("enter the tracks");

for(i=3;i<=n+2;i++) scanf("%d",&t[i]);

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

for(j=0;j<=(n+2)-i-1;j++)

if(t[j]>t[j+1]) { temp=t[j];

t[j]=t[j+1]; t[j+1]=temp; }

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

if(t[i]==h) j=i;

break;

p=0;

while(t[j]!=tot-1) {

atr[p]=t[j];

j++; p++; }

atr[p]=t[j];

p++; i=0;

while(p!=(n+3) && t[i]!=t[h]) { atr[p]=t[i];

i++; p++; } for(j=0;

jatr[j+1]) d[j]=atr[j]-atr[j+1];

else d[j]=atr[j+1]-atr[j];

sum+=d[j]; 35 }

printf("total header movements%d",sum);

printf("avg is %f",(float)sum/n); getch(); }

**OUTPUT**

Enter the track position : 55 58 60 70 18 90 150 160 184 Enter starting position : 100

Tracks traversed Difference between tracks

150 50

160 10

184 24

18 240

55 37

58 3

60 2

70 10

90 20

Average header movements: 35.7777779