|  |  |  |
| --- | --- | --- |
| D:\College Data\formats\college icons\jspmlogo.gif | **JSPM’s**  **RAJARSHI SHAHU COLLEGE OF ENGINEERING**  **TATHAWADE, PUNE-33**  **Autonomous Institute Affiliated to Savitribai Phule Pune University** | D:\College Data\formats\college icons\rscoelogo.gif |

**OS Lab Assignments**

**Name:- Pranav Shantaram Saste**

**Class:- SY B.Tech**

**Roll No:- 15**

**Assignment No.1**

1. **write a program to create a process using fork**

#include <stdio.h>

int main()

{

int n;

printf("Enter no of process");

scanf("%d", &n);

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

{

if(fork()==0)

{

printf("[son] pid %d from [parent] pid %d\n",getpid(),getppid());

exit(0);

}

}

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

wait(NULL);

}

Output-Enter no of process4

[son] pid 1649 from [parent] pid 1648

[son] pid 1650 from [parent] pid 1648

[son] pid 1651 from [parent] pid 1648

[son] pid 1652 from [parent] pid 1648

**Assignment No.2**

**Aim: Simulation of Scheduling algorithm. a) FCFS b) SJF c) RR.**

1. **FCFS**

#include<stdio.h>

int main()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

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

scanf("%d",&n);

printf("\nEnter 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

//calculating waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

}

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

//calculating turnaround time

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

{

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

avwt+=wt[i];

avtat+=tat[i];

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

}

avwt/=i;

avtat/=i;

printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

OUTPUT:

Enter total number of processes(maximum 20):3

Enter Process Burst Time

P[1]:1

P[2]:2

P[3]:3

Process         Burst Time      Waiting Time    Turnaround Time

P[1]            1               0               1

P[2]            2               1               3

P[3]            3               3               6

Average Waiting Time:1

Average Turnaround Time:3

**b) SJF**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:n");

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

{

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

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

p[i]=i+1;

}

//sorting of burst times

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

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=(float)total/n;

total=0;

printf("n Processt Burst Time tWaitingTimetTurnaround Time");

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

{

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

total+=tat[i];

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

}

avg\_tat=(float)total/n;

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

printf("\nAverage Turnaround Time=%fn",avg\_tat);

}

OUTPUT:-

Enter number of process:5

nEnter Burst Time:np1:4

p2:3

p3:7

p4:1

2

p5:nProcesst    Burst Time    tWaiting TimetTurnaround Time

p4                1                 0            1

p5                2                 1            3

p2                3                 3            6

p1                4                 6            10

p3                7                 10           17

Average Waiting Time=4.000000

Average Turnaround Time=7.400000n

1. **RR**

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &limit);

x = limit;

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

{

printf("\nEnter Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Burst Time:\t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

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

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

printf("\nProcessID\tBurstTime\t Turnaround Time\t Waiting Time\n");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum&& temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i],

total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

printf("\nnAverage Waiting Time:\t%f", average\_wait\_time);

printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

return 0;

}

Output:-

Enter Total Number of Processes:        4

Enter Details of Process[1]

Arrival Time:   0

Burst Time:     4

Enter Details of Process[2]

Arrival Time:   1

Burst Time:     7

Enter Details of Process[3]

Arrival Time:   2

Burst Time:     5

Enter Details of Process[4]

Arrival Time:   3

Burst Time:     6

Enter Time Quantum:     3

Process IDBurst Timet Turnaround Timet Waiting Time

Process[1]              4                13              9

Process[3]              5                16              11

Process[4]              6                18              12

Process[2]              7                21              14

nAverage Waiting Time:  11.500000

Avg Turnaround Time:    17.000000

**Assignment No.3**

**Aim:Write a Program to simulate Deadlock.**

#include<stdio.h>

static int mark[20];

int i,j,np,nr;

int main()

{

int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");

scanf("%d",&np);

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

scanf("%d",&nr);

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

{

printf("\nTotal Amount of the Resource R%d: ",i+1);

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

}

printf("\nEnter the request matrix:");

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

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

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

printf("\nEnter the allocation matrix:");

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

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

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

/Available Resource calculation/

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

{

avail[j]=r[j];

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

{

avail[j]-=alloc[i][j];

}

}

//marking processes with zero allocation

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

{

int count=0;

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

{

if(alloc[i][j]==0)

count++;

else

break;

}

if(count==nr)

mark[i]=1;

}

// initialize W with avail

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

w[j]=avail[j];

//mark processes with request less than or equal to W

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

{

int canbeprocessed=0;

if(mark[i]!=1)

{

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

{

if(request[i][j]<=w[j])

canbeprocessed=1;

else

{

canbeprocessed=0;

break;

}

}

if(canbeprocessed)

{

mark[i]=1;

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

w[j]+=alloc[i][j];

}

}

}

//checking for unmarked processes

int deadlock=0;

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

if(mark[i]!=1)

deadlock=1;

if(deadlock)

printf("\n Deadlock detected");

else

printf("\n No Deadlock possible");

}

OUTPUT: Enter the no of process: 4

Enter the no of resources: 5

Total Amount of the Resource R1: 2

Total Amount of the Resource R2: 1

Total Amount of the Resource R3: 1

Total Amount of the Resource R4: 2

Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0

1 1 0 0 0

0 0 0 1 0

0 0 0 0 0

 Deadlock detected

**Assignment No.4**

**Aim: Write a Program For Inter Process Communication using Pipe.**

#include<stdio.h>

#include<unistd.h>

int main() {

int pipefds[2];

int returnstatus;

int pid;

char writemessages[2][20]={"Hi", "Hello"};

char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n");

return 1;

}

pid = fork();

// Child process

if (pid == 0) {

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);

} else { //Parent process

printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);

write(pipefds[1], writemessages[0], sizeof(writemessages[0]));

printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);

write(pipefds[1], writemessages[1], sizeof(writemessages[1]));

}

return 0;

}

**Output:-**

Parent Process - Writing to pipe - Message 1 is Hi

Parent Process - Writing to pipe - Message 2 is Hello

Child Process - Reading from pipe – Message 1 is Hi

Child Process - Reading from pipe – Message 2 is Hello

**Assignment No.5**

**Aim:Write a Program to simulate page Replacement algorithm.**

**a) FIFO**

**b) LRU**

**c) optimal.**

#include<stdio.h>

int n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data) // function to check is page already their or not

{

hit=0;

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

{

if(p[j]==data) //if page found

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void fifo()

{

initialize();

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

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

for(k=0; k<nf-1; k++)

p[k]=p[k+1];

p[k]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void optimal()

{

initialize();

int near[50];

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

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

near[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

near[j]=9999;

}

int max=-9999;

int repindex;

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

{

if(near[j]>max)

{

max=near[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void lru()

{

initialize()

int least[50];

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

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

least[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

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

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

void lfu()

{

int usedcnt[100];

int least,repin,sofarcnt=0,bn;

initialize();

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

usedcnt[i]=0;

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

{

printf("\n For %d :",in[i]);

if(isHit(in[i]))

{

int hitind=getHitIndex(in[i]);

usedcnt[hitind]++;

printf("No page fault!");

}

else

{

pgfaultcnt++;

if(bn<nf)

{

p[bn]=in[i];

usedcnt[bn]=usedcnt[bn]+1;

bn++;

}

else

{

least=9999;

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

if(usedcnt[k]<least)

{

least=usedcnt[k];

repin=k;

}

p[repin]=in[i];

sofarcnt=0;

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

if(in[i]==in[k])

sofarcnt=sofarcnt+1;

usedcnt[repin]=sofarcnt;

}

dispPages();

}

}

dispPgFaultCnt();

}

void secondchance()

{

int usedbit[50];

int victimptr=0;

initialize();

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

usedbit[i]=0;

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

{

printf("\nFor %d:",in[i]);

if(isHit(in[i]))

{

printf("No page fault!");

int hitindex=getHitIndex(in[i]);

if(usedbit[hitindex]==0)

usedbit[hitindex]=1;

}

else

{

pgfaultcnt++;

if(usedbit[victimptr]==1)

{

do

{

usedbit[victimptr]=0;

victimptr++;

if(victimptr==nf)

victimptr=0;

}

while(usedbit[victimptr]!=0);

}

if(usedbit[victimptr]==0)

{

p[victimptr]=in[i];

usedbit[victimptr]=1;

victimptr++;

}

dispPages();

}

if(victimptr==nf)

victimptr=0;

}

dispPgFaultCnt();

}

int main()

{

int choice;

while(1)

{

printf("\nPage Replacement Algorithms\n1.Enter data\n2.FIFO\n3.Optimal\n4.LRU\n5.LFU\n6.Second Chance\n7.Exit\nEnter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

getData();

break;

case 2:

fifo();

break;

case 3:

optimal();

break;

case 4:

lru();

break;

case 5:

lfu();

break;

case 6:

secondchance();

break;

default:

return 0;

break;

}

}

}

**Output:-**

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:1

Enter length of page reference sequence:9

Enter the page reference sequence:1 2 3 4 1 2 5 1 2

Enter no of frames:5

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:2

For 1 : 1

For 2 : 1 2

For 3 : 1 2 3

For 4 : 1 2 3 4

For 1 :No page fault

For 2 :No page fault

For 5 : 1 2 3 4 5

For 1 :No page fault

For 2 :No page fault

Total no of page faults:5

**Assignment No.6**

**Aim:Write a Program for File allocation Methods.**

**a) continuous**

**b) Linked**

**c) Indexed**

**a) Continuous**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int f[50], i, st, len, j, c, k, count = 0;

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

f[i]=0;

printf("Files Allocated are : \n");

x: count=0;

printf("Enter starting block and length of files: ");

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

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

if(f[k]==0)

count++;

if(len==count)

{

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

if(f[j]==0)

{

f[j]=1;

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

}

if(j!=(st+len-1))

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

}

else

printf("The file is not allocated \n");

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit();

getch();

}

**Output:-**

Files Allocated are :

                     Enter starting block and length of files: 14 3

14      1

15      1

16      1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 1

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 4

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)0

**b) Linked**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int f[50], p,i, st, len, j, c, k, a;

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

f[i]=0;

printf("Enter how many blocks already allocated: ");

scanf("%d",&p);

printf("Enter blocks already allocated: ");

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

{

scanf("%d",&a);

f[a]=1;

}

x: printf("Enter index starting block and length: ");

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

k=len;

if(f[st]==0)

{

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

{

if(f[j]==0)

{

f[j]=1;

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

}

else

{

printf("%d Block is already allocated \n",j);

k++;

}

}

}

else

printf("%d starting block is already allocated \n",st);

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit(0);

return 0;

}

**Output:-**

Enter how many blocks already allocated: 3

Enter blocks already allocated: 1

3

5

Enter index starting block and length: 2 2

2-------->1

3 Block is already allocated

4-------->1

Do you want to enter more file(Yes - 1/No - 0)0

**c) Indexed**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main()

{

int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;

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

f[i]=0;

x:printf("Enter the index block: ");

scanf("%d",&ind);

if(f[ind]!=1)

{

printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);

scanf("%d",&n);

}

else

{

printf("%d index is already allocated \n",ind);

goto x;

}

y: count=0;

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

{

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

if(f[index[i]]==0)

count++;

}

if(count==n)

{

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

f[index[j]]=1;

printf("Allocated\n");

printf("File Indexed\n");

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

printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);

}

else

{

printf("File in the index is already allocated \n");

printf("Enter another file indexed");

goto y;

}

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit(0);

getch();

}

**Output:-**

Enter the index block: 5

Enter no of blocks needed and no of files for the index 5 on the disk :

4

1

2

3

4

Allocated

File Indexed

5-------->1 : 1

5-------->2 : 1

5-------->3 : 1

5-------->4 : 1

Do you want to enter more file(Yes - 1/No - 0)1

Enter the index block: 4

4 index is already allocated

Enter the index block: 6

Enter no of blocks needed and no of files for the index 6 on the disk :

2

7

8

Allocated

File Indexed

6-------->7 : 1

6-------->8 : 1

Do you want to enter more file(Yes - 1/No - 0)0

**Assignment No.7**

**Aim:Write a Program to simulate disk Scheduling Algorithm. a) FCFS b) SCAN** **c) CSCAN d) LOOK.**

1. **FCFS**

#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

1. **SCAN**

#include<conio.h>

#include<stdio.h>

int main()

{

int i,j,sum=0,n;

int d[20];

int disk; //loc of head

int temp,max;

int dloc; //loc of disk in array

clrscr();

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]);

}

d[n]=disk;

n=n+1;

for(i=0;i<n;i++) // sorting disk locations

{

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

{

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

{

temp=d[i];

d[i]=d[j];

d[j]=temp;

}

}

}

max=d[n];

for(i=0;i<n;i++) // to find loc of disc in array

{

if(disk==d[i])

{

dloc=i;

break;

}

}

for(i=dloc;i>=0;i--)

{

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

}

printf("0 -->");

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

{

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

}

sum=disk+max;

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

getch();

return 0;

}

**Output:-**

enter number of location        8

enter position of head  53

enter elements of disk queue

98

183

37

122

14

124

65

67

53 -->37 -->14 -->0 -->65-->67-->98-->122-->124-->183-->

movement of total cylinders

**c)   CSCAN**

       #include<stdio.h>

int main()

{

int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],

temp1=0,temp2=0;

float avg;

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

scanf("%d",&max);

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

scanf("%d",&head);

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",&temp);

if(temp>=head)

{

queue1[temp1]=temp;

temp1++;

}

else

{

queue2[temp2]=temp;

temp2++;

}

}

for(i=0;i<temp1-1;i++)

{

for(j=i+1;j<temp1;j++)

{

if(queue1[i]>queue1[j])

{

temp=queue1[i];

queue1[i]=queue1[j];

queue1[j]=temp;

}

}

}

for(i=0;i<temp2-1;i++)

{

for(j=i+1;j<temp2;j++)

{

if(queue2[i]>queue2[j])

{

temp=queue2[i];

queue2[i]=queue2[j];

queue2[j]=temp;

}

}

}

for(i=1,j=0;j<temp1;i++,j++)

queue[i]=queue1[j];

queue[i]=max;

queue[i+1]=0;

for(i=temp1+3,j=0;j<temp2;i++,j++)

queue[i]=queue2[j];

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 initial head position

50

Enter the size of queue request

8

Enter the queue of disk positions to be read

90 120 35 122 38 128 65 68

Disk head moves from 50 to 65 with seek                                                                           15

Disk head moves from 65 to 68 with seek                                                                           3

Disk head moves from 68 to 90 with seek                                                                           22

Disk head moves from 90 to 120 with seek                                                                           30

Disk head moves from 120 to 122 with seek                                                                           2

Disk head moves from 122 to 128 with seek                                                                           6

Disk head moves from 128 to 200 with seek                                                                           72

Disk head moves from 200 to 0 with seek                                                                           200

Disk head moves from 0 to 35 with seek                                                                           35

Disk head moves from 35 to 38 with seek                                                                           3

Total seek time is 388

Average seek time is 48.500000

**d) LOOK.**

#include<math.h>

#include<stdio.h>

int main()

{

int i,n,j=0,k=0,x=0,l,req[50],mov=0,cp,ub,end, lower[50],upper[50], temp,a[50];

printf("enter the current position\n");

scanf("%d",&cp);

printf("enter the number of requests\n");

scanf("%d",&n);

printf("enter the request order\n");

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

{

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

}

printf("Enter the upper bound\n");

scanf("%d",&ub);

/\*break the request array into two arrays : one with requests lower than current and other with requests higher than current position. Also sort these two new arrays\*/

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

{

if(req[i]<cp)

{

lower[j]=req[i];

j++;

}

if(req[i]>cp)

{

upper[k]=req[i];

k++;

}

}

//sort the lower array in reverse order

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

{

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

{

if(lower[l]<lower[l+1])

{

temp=lower[l];

lower[l]=lower[l+1];

lower[l+1]=temp;

}

}

}

// sort the upper array in ascending order

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

{

for(l=0;l<k-1;l++)

{

if(upper[l]>upper[l+1])

{

temp=upper[l];

upper[l]=upper[l+1];

upper[l+1]=temp;

}

}

}

printf("Enter the end to which the head is moving (0 - for lower end(zero) and 1 - for upper end\n");

scanf("%d",&end);

switch(end)

{

case 0:

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

{

a[x]=lower[i];

x++;

}

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

{

a[x]=upper[i];

x++;

}

break;

case 1:

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

{

a[x]=upper[i];

x++;

}

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

{

a[x]=lower[i];

x++;

}

break;

}

mov=mov+abs(cp-a[0]);

printf("%d -> %d",cp,a[0]);

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

{

mov=mov+abs(a[i]-a[i-1]);

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

}

printf("\n");

printf("total head movement = %d\n",mov);

}

**Output:-**

enter the current position

37

enter the number of requests

5

enter the request order

90 120 35 128 37

Enter the upper bound

128

Enter the end to which the head is moving (0 - for lower end(zero) and 1 - for upper end

0

37 -> 35 -> 90 -> 120 -> 128

total head movement = 95

**Assignment No.8**

**Aim: Write a Program to implement UNIX system call –open using shell Programming**

#include <stdio.h>

#include <stdlib.h> // For exit()

int main()

{

    FILE \*fptr1, \*fptr2;

    char filename[100], c;

    printf("Enter the filename to open for reading \n");

    scanf("%s", filename);

    // Open one file for reading

    fptr1 = fopen(filename, "r");

    if (fptr1 == NULL)

    {

        printf("Cannot open file %s \n", filename);

        exit(0);

    }

    printf("Enter the filename to open for writing \n");

    scanf("%s", filename);

    // Open another file for writing

    fptr2 = fopen(filename, "w");

    if (fptr2 == NULL)

    {

        printf("Cannot open file %s \n", filename);

        exit(0);

    }

    // Read contents from file

    c = fgetc(fptr1);

    while (c != EOF)

    {

        fputc(c, fptr2);

        c = fgetc(fptr1);

    }

    printf("\nContents copied to %s", filename);

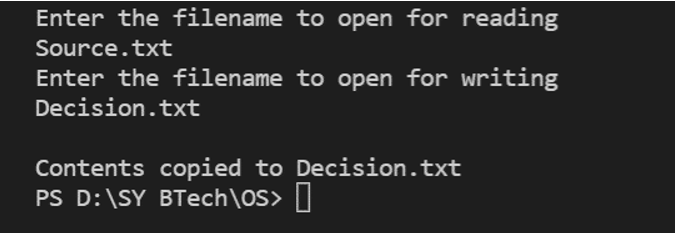
    fclose(fptr1);

    fclose(fptr2);

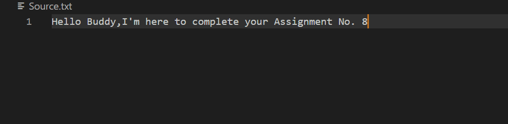
    return 0;

}

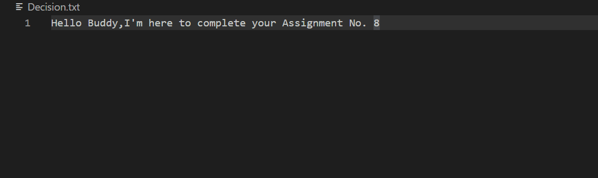
**Output:- Console Screen:**

****

**Source.txt:**

****

**Decision.txt:**

****