

Write a C program to simulate disk scheduling algorithms:

- (a) FCFS
- (b) SCAN
- (c) c-SCAN

Code:

(a) FCFS:

```
#include<stdio.h>

#include<stdlib.h>

int main()

{

    int RQ[100],i,n,TotalHeadMoment=0,initial;

    printf("Enter the number of Requests\n");

    scanf("%d",&n);

    printf("Enter the Requests sequence\n");

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

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

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

    scanf("%d",&initial);

    // logic for FCFS disk scheduling
```

```

for(i=0;i<n;i++)
{
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
    initial=RQ[i];
}

printf("Total head moment is %d",TotalHeadMoment);
return 0;
}

```

(b) SCAN:

```

#include<stdio.h>

#include<stdlib.h>

int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
        scanf("%d",&RQ[i]);

```

```
printf("Enter initial head position\n");
scanf("%d",&initial);

printf("Enter total disk size\n");
scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

// logic for Scan disk scheduling
```

```
/*logic for sort the request array */

for(i=0;i<n;i++)
{
    for(j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }
    }
}
```

```
    }

}

int index;

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

{

    if(initial<RQ[i])

    {

        index=i;

        break;

    }

}

// if movement is towards high value

if(move==1)

{

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

    {

        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

        initial=RQ[i];

    }

}
```

```

}

// last movement for max size

TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);

initial = size-1;

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

{

    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

    initial=RQ[i];

}

}

// if movement is towards low value

else

{

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

    {

        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

        initial=RQ[i];

    }

    // last movement for min size

    TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
}

```

```

initial =0;

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

{
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

    initial=RQ[i];

}

}

printf("Total head movement is %d",TotalHeadMoment);

return 0;
}

```

(c) c-SCAN:

```

#include<stdio.h>

#include<stdlib.h>

int main()

{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

    printf("Enter the number of Requests\n");

    scanf("%d",&n);

    printf("Enter the Requests sequence\n");

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

```

```
scanf("%d",&RQ[i]);  
printf("Enter initial head position\n");  
scanf("%d",&initial);  
printf("Enter total disk size\n");  
scanf("%d",&size);  
printf("Enter the head movement direction for high 1 and for low 0\n");  
scanf("%d",&move);  
  
// logic for C-Scan disk scheduling
```

```
/*logic for sort the request array */  
for(i=0;i<n;i++)  
{  
    for( j=0;j<n-i-1;j++)  
    {  
        if(RQ[j]>RQ[j+1])  
        {  
            int temp;  
            temp=RQ[j];  
            RQ[j]=RQ[j+1];  
        }  
    }  
}
```

```
RQ[j+1]=temp;  
}  
  
}  
  
}  
  
int index;  
for(i=0;i<n;i++)  
{  
    if(initial<RQ[i])  
    {  
        index=i;  
        break;  
    }  
}  
  
// if movement is towards high value  
if(move==1)  
{  
    for(i=index;i<n;i++)
```

```

{

    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

    initial=RQ[i];

}

// last movement for max size

TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);

/*movement max to min disk */

TotalHeadMoment=TotalHeadMoment+abs(size-1-0);

initial=0;

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

{

    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

    initial=RQ[i];

}

}

// if movement is towards low value

else

{

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

{

```

```

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

// last movement for min size

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);

/*movement min to max disk */

TotalHeadMoment=TotalHeadMoment+abs(size-1-0);

initial =size-1;

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

{

    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

    initial=RQ[i];

}

printf("Total head movement is %d",TotalHeadMoment);

return 0;
}

```

OUTPUT

```
Enter the number of Requests  
8  
Enter the Requests sequence  
95 180 34 119 11 123 62 64  
Enter initial head position  
50  
Total head moment is 644
```

```
Enter the number of Requests  
6  
Enter the Requests sequence  
90 120 30 60 50 80  
Enter initial head position  
70  
Enter total disk size  
200  
Enter the head movement direction for high 1 and for low 0  
0  
Total head movement is 190
```

```
Enter the number of Requests  
3  
Enter the Requests sequence  
2 1 0  
Enter initial head position  
1  
Enter total disk size  
3  
Enter the head movement direction for high 1 and for low 0  
1  
Total head movement is 4
```

Write a C program to simulate page replacement algorithms:

- (a) FIFO
- (b) LRU
- (c) Optimal

Code:

```
#include<stdio.h>

int n, nf, i, j, k;

int in[100];

int p[50];

int hit=0;

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)
```

```
{
```

```
    hit=0;
```

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

```
{
```

```
    if(p[j]==data)
```

```
{
```

```
        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])  
                    {  
                        found=1;  
                    }  
                }  
                if(found==0)  
                {  
                    in[i]=pg;  
                    hit++;  
                }  
            }  
        }  
    }  
}
```

```
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();

}

int main()

{

    int choice;

    while(1)

    {

        printf("\nPage    Replacement    Algorithms\n1.Enter
data\n2.FIFO\n3.Optimal\n4.LRU\n5.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;

default: return 0;
    break;

}

}

}
```

Write C program to simulate Disk Scheduling

```
#include <stdio.h>
int n, nf
int in[100];
int p[50];
int hit = 0;
int c, j, k
int pageFaultCnt = 0;
void getData()
{
    printf("Enter length of page reference sequence:");
    scanf("%d", &n);
    printf("Enter the page reference sequence:");
    for (i=0; i<n; i++)
    {
        scanf("%d", &in[i]);
    }
    printf("Enter no of frames:");
    scanf("%d", &nf);
}
void initialize()
{
    p[0] = 0;
    for (i=0; i<nf; i++)
    {
        p[i] = 9999;
    }
    int i;
    for (i=0; i<nf; i++)
    {
        if (p[i] == 9999)
        {
            p[i] = in[0];
            hit++;
        }
    }
}
```

```

    kid = 1
    break;
}

3
releas hit;
3
int get hit address(int data)
{
    int hitaid;
    for (k=0; k < nf; k++)
    {
        if (PCK[k] == data)
        {
            hitaid = k;
            break;
        }
    }
    releas hitaid;
}

void dispares()
{
    for (k=0; k < nf; k++)
        if (PCK[k] != 9999)
            printf ("%d", PCK[k]);
}

3
void usrgfault cont()
{
    printf ("in total no of page faults: %d", pgfaultcnt);
}

3
void ff0()
{
    initalloc();
    for (i=0; i<n; i++)
    {
}

```

```

    if ("In for %d:", in [0]);
    if (cas bit cin [i][j] == 0)
    {
        for (k = 0; k < nf - 1; k++)
            r[k] = p[k + 1];
        r[nf] = in [i];
        pagefault cnt++;
        despage();
    }
    else
        coutf ("No page fault");
    stoppage fault cnt();
    board optimal();
}

initialise();
cout near [50];
for (i = 0; i < n; i++)
{
    coutf ("In for %d:", in [i]);
    if (cas bit cin [i][j] == 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;

3 else
    found = 0;
    {
        if (!found)
            real [j] = 0;
        int max = -9999;
        int repindex;
        for (i=0; i<n; i++)
            if (area [ij] > max)
                max = real [ij];
            repindex = ij;
    }
    p [repindex] = in [i];
    pgfaultcnt++;
    dispages();
}
else
    printf ("No page fault");
}

3 dispfault();
void disp()
{
    initialize();
    int least [50];
    for (i=0; i<n; i++)
    {
        printf ("%n Pos of di ", in [i]);
        if (asHit (in [i]) > 20)

```

```

    {
        if (Cj == 0; j < N; j++)
            int pg = p[j];
        int found = 0;
        for (k=0; k >= 0; k--)
            {
                if (kg == c[k])
                    least[j] = k;
                found = 1;
                break;
            }
        else
            found = 0;
        if (!found)
            least[j] = 9999;
        min = 9999;
        int remainder;
        if (least[j] < min)
            min = least[j];
        remainder = j;
        cout << "Page address: ";
        display();
        cout << endl;
        cout << "Page fault count: ";
        cout << pgfault << endl;
        cout << "No page fault! ";
        cout << endl;
        cout << "Display page fault count: ";
        displaypgfault();
        cout << endl;
        cout << "int main() ";
        cout << endl;
        cout << "int choice; ";
        cout << endl;
        cout << "while(1) ";
        cout << endl;
        cout << "printf(\"In Page replacement Algorithm -1) Enter data\\n\\n- FIFO\\n3. Optimal\\n4. LRU\\n5. MRU\\n6. Custom choice\\n\") ";
    }

```

```

Scans()
switch (choice) {
    case 1: getData();
        break;
    case 2: fifo();
        break;
    case 3: optimise();
        break;
    case 4: lru();
        break;
    default:
        return 0;
        break;
}

```

3 3
3

Output

1) FFS

enter no of requests

3

enter the sequence

55 12 37 22

enter head position

20

Total head. is 118

2) SCAN & CSAN

enter the requests sequence

33 12 37 22 40

enter head position

20

enter total size

100

enter the head direction

high 1 for 1000
Total head movement for SCAN is 93
Total head movement for CSAN is 172

✓ 10/10
N 28/8/2^o