LAB 11

Aim: 1. FIFO Page Replacement

2. Optimal Page Replacement

FIFO Page Replacement

Algorithm:

1. Start traversing the pages.
2. Declare the size as the length of the Page.
3. Check the need of the replacement from the page to memory.
4. Check the need of the replacement from the old page to the new page in memory.
5. Form the queue to hold all pages.
6. Insert the required page memory into the queue.
7. Check the bad replacements and page faults.
8. Get the number of processes to be inserted.
9. Show the values.

Code:

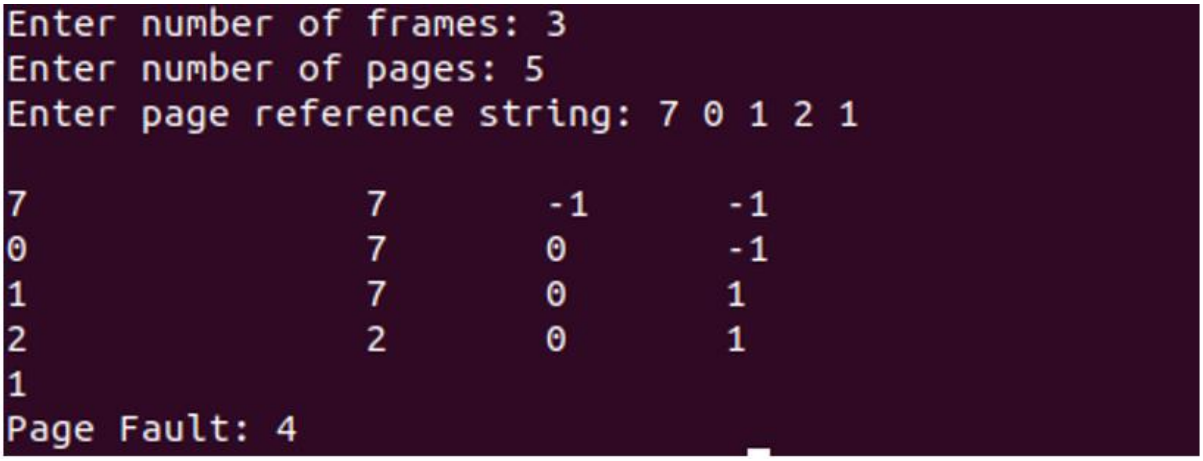
```
#include <stdio.h> int main()
{
int i, j = 0, nf, np, pg[50], frame[10], k, av, c = 0;
printf("Enter number of frames: ");
scanf("%d", &nf);
```

```
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter page reference string: "); for (i = 1; i <= np;
i++){
scanf("%d", &pg[i]);
}

for (i = 0; i < nf; i++){ frame[i] = -1;
}

printf("\n");
for (i = 1; i <= np; i++)
{
printf("%d\t\t", pg[i]); av = 0;
for (k = 0; k < nf; k++) if (frame[k] == pg[i])
av = 1; if (av == 0)
{
frame[j] = pg[i]; j = (j + 1) % nf; c++;
for (k = 0; k < nf; k++) printf("%d\t", frame[k]);
}
printf("\n");
}
printf("Page Fault: %d\n", c); return 0;
}
```

OUTPUT:



Optimal Page Replacement

Algorithm:

1. If the referred page is already present, increment the hit count.

2. If not present, find if a page is never referenced in future. If such a page exists, replace this page with a new page. If no such page exists, find a page that is referenced farthest in future. Replace this page with a new page.

Code:

```
#include<stdio.h>


int main()

{

int nf, np, fr[10], pg[30], tmp[10], f1, f2, f3, i, j, k, pos, max, fl = 0; printf("Enter number of frames: ");

scanf("%d", &nf);
```

```
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter page reference string: "); for(i = 0; i < np;
++i){
scanf("%d", &pg[i]);
}

for(i = 0; i < nf; ++i){ fr[i] = -1;
}

for(i = 0; i < np; ++i){ f1 = f2 = 0;
for(j = 0; j < nf; ++j){ if(fr[j] == pg[i]){
f1 = f2 = 1;
break;
}
}

if(f1 == 0){ for(j=0; j<nf; ++j){
if(fr[j]==-1){ fl++;
fr[j] = pg[i]; f2 = 1;
break;
}
}
} if(f2==0){ f3 = 0;
for(j=0; j<nf; j++){
tmp[j] = -1;
```

```
for(k=i + 1; k<np; k++){ if(fr[j] == pg[k]){
tmp[j] = k; break;
}
}
}
for(j = 0; j < nf; j++){ if(tmp[j] == -1){ pos = j;
f3 = 1;
break;
}
} if(f3==0){
max = tmp[0]; pos = 0;

for(j=1; j<nf; j++){ if(tmp[j] > max){

max = tmp[j]; pos = j;
}
}
}
fr[pos] = pg[i]; fl++;
}

printf("\n");

for(j=0; j<nf; j++){ printf("%d\t", fr[j]);
```

```
}
}
printf("\n\nTotal Page Faults = %d\n", fl);
return 0;
}
```

```
Enter number of frames: 3
Enter number of pages: 6
Enter page reference string: 2 3 4 2 1 1

2       -1      -1
2       3       -1
2       3       4
2       3       4
1       3       4
1       3       4

Total Page Faults = 4
```

LAB 12

Aim: 1. LRU Page Replacement

2. Comparing FIFO, Optimal & LRU Page Replacement Algorithms

LRU Page Replacement

Algorithm:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by the counter value
7. Stack them according to the selection.
8. Display the values
9. Stop the process

Code:

```
#include <stdio.h>

int LRU(int time[], int n)
{
int i, mn = time[0], p = 0;
```

```
for (i = 1; i < n; ++i)
{
if (time[i] < mn)
{
mn = time[i]; p = i;
}

}
return p;
}

int main()
{
int nf, np, fr[10], pg[30], cn = 0, time[10], f1, f2, i, j, p, fl
= 0; printf("Enter number of frames: ");
scanf("%d", &nf);
printf("Enter number of pages: "); scanf("%d", &np);
printf("Enter reference string: "); for (i = 0; i < np; ++i)
{
scanf("%d", &pg[i]);
}

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

```
fr[i] = -1;
}

for (i = 0; i < np; ++i)
{
f1 = f2 = 0;

for (j = 0; j < nf; ++j)
{
if (fr[j] == pg[i])
{
cn++; time[j] = cn; f1 = f2 = 1;
break;
}

}

if (f1 == 0)
{
for (j = 0; j < nf; ++j)
{
if (fr[j] == -1)
{
cn++; fl++;
fr[j] = pg[i]; time[j] = cn; f2 = 1;
```

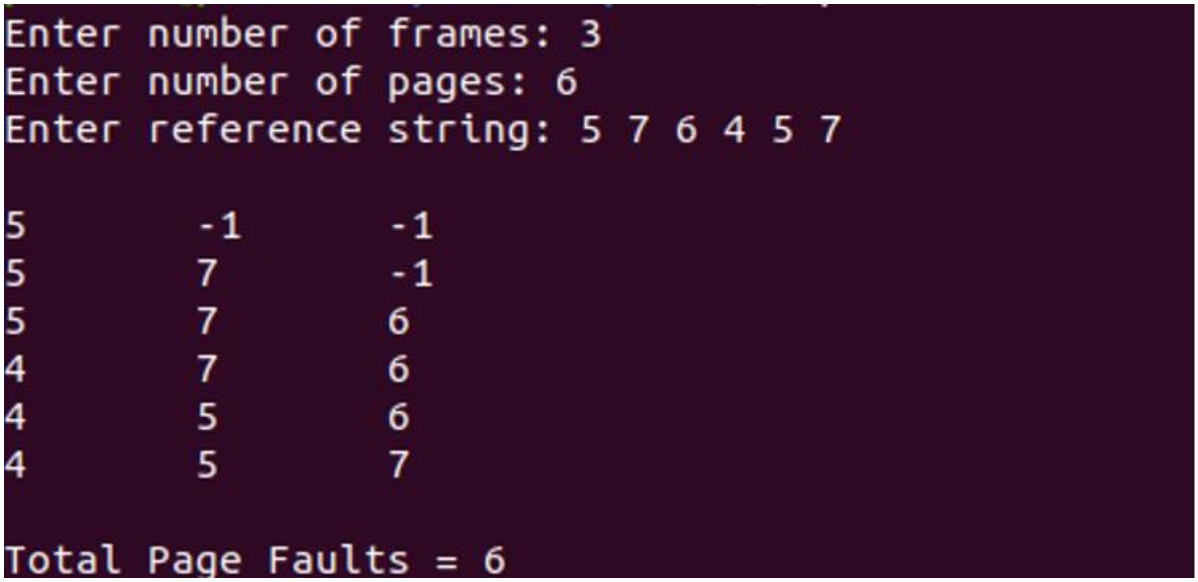
```
break;
}
}
}

if (f2 == 0)
{
p = LRU(time, nf); cn++;
fl++;
fr[p] = pg[i]; time[p] = cn;
}

printf("\n");

for (j = 0; j < nf; ++j)
{
printf("%d\t", fr[j]);
}
}
printf("\n\nTotal Page Faults = %d\n", fl);


return 0;
}
```
OUTPUT:



Comparing FIFO, Optimal & LRU Page Replacement Algorithms Algorithm:

1. Write code for FIFO Page Replacement in a function
2. Write code for Optimal Page Replacement in a function
3. Write code for LRU Page Replacement in a function
4. Execute all 3 in the main function.

Code:

```
#include <stdio.h>


void FIFO(int nf, int np, int pg[], int fr[]){ int i, j = 0, k, av, fl = 0;
for (i = 1; i <= np; i++)
{
printf("%d\t\t", pg[i]); av = 0;
for (k = 0; k < nf; k++) if (fr[k] == pg[i])
```

```
av = 1; if (av == 0)
{
fr[j] = pg[i];
j = (j + 1) % nf; fl++;
for (k = 0; k < nf; k++) printf("%d\t", fr[k]);
}
printf("\n");
}
printf("\n\nTotal Page Faults = %d\n", fl);
}

void optimal(int nf, int np, int pg[], int fr[]){
int tmp[10], f1, f2, f3, i, j, k, pos, max, fl = 0; for(i = 0; i <
np; ++i){

f1 = f2 = 0;
for(j = 0; j < nf; ++j){ if(fr[j] == pg[i]){
f1 = f2 = 1;
break;
}
}
if(f1 == 0){ for(j=0; j<nf; ++j){
if(fr[j]==-1){ fl++;
fr[j] = pg[i]; f2 = 1;
break;
```

```
}
}
} if(f2==0){ f3 = 0;
for(j=0; j<nf; j++){
tmp[j] = -1;

for(k=i + 1; k<np; k++){ if(fr[j] == pg[k]){
tmp[j] = k; break;
}
}
}
for(j = 0; j < nf; j++){ if(tmp[j] == -1){ pos = j;
f3 = 1;
break;
}

} if(f3==0){
max = tmp[0]; pos = 0;

for(j=1; j<nf; j++){ if(tmp[j] > max){ max = tmp[j];
pos = j;
}
}
}
fr[pos] = pg[i]; fl++;
```

```
}

printf("\n");

for(j=0; j<nf; j++){ printf("%d\t", fr[j]);
}
}
printf("\n\nTotal Page Faults = %d\n", fl);
}

int LRU(int time[], int n)
{
int i, mn = time[0], p = 0;

for (i = 1; i < n; ++i)
{
if (time[i] < mn)
{
mn = time[i]; p = i;

}
}
return p;
}
```

```
void LRUpr(int nf, int np, int pg[], int fr[]){ int i, j, cn = 0,
time[10], f1, f2, p, fl = 0; for (i = 0; i < np; ++i)
{
f1 = f2 = 0;

for (j = 0; j < nf; j++)
{
if (fr[j] == pg[i])
{
cn++; time[j] = cn; f1 = f2 = 1;
break;
}
}

if (f1 == 0)
{
for (j = 0; j < nf; j++)
{
if (fr[j] == -1)
{
cn++; fl++;
fr[j] = pg[i]; time[j] = cn; f2 = 1;
break;
}
```

```
}
}

if (f2 == 0)
{
p = LRU(time, nf); cn++;
fl++;
fr[p] = pg[i]; time[p] = cn;
}

printf("\n");

for (j = 0; j < nf; j++)
{
printf("%d\t", fr[j]);
}
}
printf("\n\nTotal Page Faults = %d\n", fl);
}

int main()
{
int nf, np, fr[10], pg[30], i; printf("Enter number of
frames: "); scanf("%d", &nf);
printf("Enter number of pages: "); scanf("%d", &np);
```

```
printf("Enter reference string: "); for (i = 0; i < np; ++i)
{
scanf("%d", &pg[i]);
}

for (i = 0; i < nf; ++i)
{
fr[i] = -1;
}
printf("\n\nFIFO PR\n"); FIFO(nf, np, pg, fr);
printf("\n\nOptimal PR"); optimal(nf, np, pg, fr);
printf("\n\nLRU PR"); LRUpr(nf, np, pg, fr); return 0;
}
```

OUTPUT:

```
E      number of frames: 3
E      number of pages: 6
Enter reference string: 5 4 1 5 2 4


FIFO PR
4               4       -1      -1
1               4       1       -1
5               4       1       5
2               2       1       5
4               2       4       5
0               2       4       0


Total Page Faults = 6


Optimal PR
2       4       5
2       4       5
2       1       5
2       1       5
2       1       5
4       1       5

Total Page Faults = 3


LRU PR
4       1       5
4       1       5
4       1       5
4       1       5
2       1       5
2       4       5
```

LAB 13

Aim: 1. FCFS Disk Scheduling

2. SSTF Disk Scheduling

FCFS Disk Scheduling

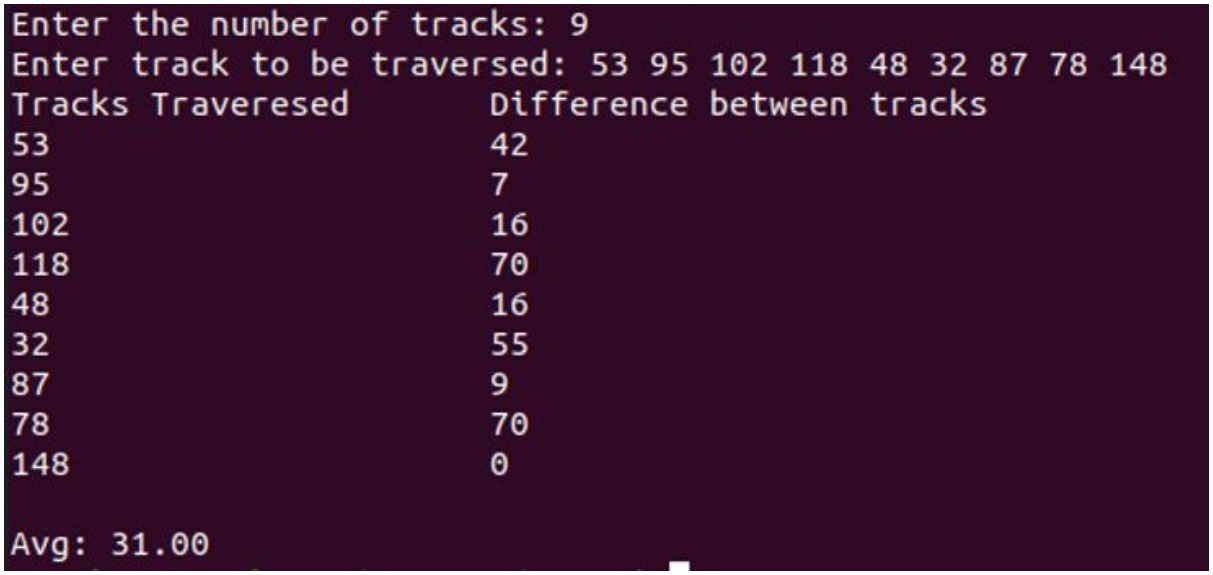
Code:

```
#include <stdio.h> #include <stdlib.h>

int main(){
int n, i, tot = 0; double av;
int t[n], tr[n];
printf("Enter the number of tracks: "); scanf("%d", &n);
printf("Enter track to be traversed: "); for(i=0; i<n; i++){
scanf("%d", &t[i]);
}
for(i=0; i<n-1; i++){
tr[i] = abs(t[i+1] - t[i]); tot += tr[i];
}
av = tot/n;
printf("Tracks Traveresed\tDifference between tracks\n");
for(i=0; i<n; i++){
printf("%d\t\t\t%d\n", t[i], tr[i]);
}
printf("\nAvg: %.2f\n", av);
```

}

OUTPUT:



SSTF Disk Scheduling

Code:

#include <stdio.h> #include <stdlib.h> #include <limits.h>

int main()

{

int i, n, tot=0, h, cn=0, d, j, m; printf("Enter the number of requests: "); scanf("%d",&n);

int r[n];

printf("Enter the requests sequence: "); for(i=0; i<n; i++){

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

}

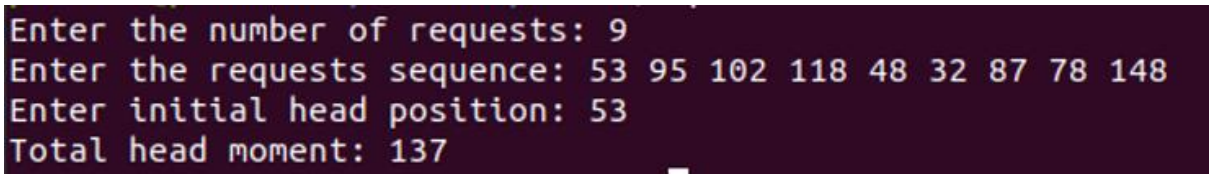
printf("Enter initial head position: "); scanf("%d",&h);

```
while(cn!=n)
{
m = INT_MAX;
for(i=0; i<n; i++)
{
d = abs(r[i] - h); if(d<m)
{
m = d; j = i;
}
}
tot += m; h = r[j];
r[j] = INT_MAX;
cn++;
}

printf("Total head moment: %d\n",tot); return 0;

}
```

OUTPUT:



LAB 14

Aim: 1. Scan Disk Scheduling

2. C-Scan Disk Scheduling

Scan Disk Scheduling

Code:

```
#include <stdio.h> #include <stdlib.h>

int main()
{
int i , j, n, tot=0, h, s, dir, tmp, pos; printf("Enter the number of requests: "); scanf("%d",&n);
int rq[n];
printf("Enter the requests sequence: "); for(i=0;i<n;i++){
scanf("%d",&rq[i]);
}
printf("Enter initial head position: "); scanf("%d",&h);
printf("Enter total disk size: "); scanf("%d",&s);
printf("Enter the head movement direction for high 1 and for low 0: "); scanf("%d",&dir);

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

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

if(rq[j]>rq[j+1])
{
tmp=rq[j]; rq[j]=rq[j+1]; rq[j+1]=tmp;
}

}
}

for(i=0; i<n; i++)
{
if(h<rq[i])
{
pos=i; break;
}
}

if(dir == 1)
{
for(i=pos; i<n; i++)
{
tot += abs(rq[i]-h); h=rq[i];
```

```
}
tot += abs(s-rq[i-1]-1); h = s-1;
for(i=pos-1; i>=0; i--)
{
tot += abs(rq[i]-h); h=rq[i];
}
}
else if(dir == 0)

{
for(i=pos-1; i>=0; i--)
{
tot += abs(rq[i]-h); h=rq[i];
}
tot += abs(rq[i+1]-0); h =0;
for(i=pos; i<n; i++)
{




}
}
else{

tot += abs(rq[i]-h); h=rq[i];
```

```
printf("Invalid Input!!");
}

printf("Total head movement: %d\n",tot);
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
Enter total disk size: 200
Enter the head movement direction for high 1 and for low 0: 1
Total head movement: 337
```

C-Scan Disk Scheduling

Code: #include<stdio.h> #include<stdlib.h> int main()

{

int i, j, n, tot=0, h, s, dir, tmp, pos; printf("Enter the number of requests: "); scanf("%d",&n);

int rq[n];

printf("Enter the requests sequence: "); for(i=0;i<n;i++){

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

}

printf("Enter initial head position: "); scanf("%d",&h);

printf("Enter total disk size: "); scanf("%d",&s);

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

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

}
}

for(i=0; i<n; i++)
{
if(h<rq[i])
{
pos = i; break;
}
}

if(dir == 1)
{
for(i=pos; i<n; i++)
{
tot += abs(rq[i]-h); h=rq[i];
```

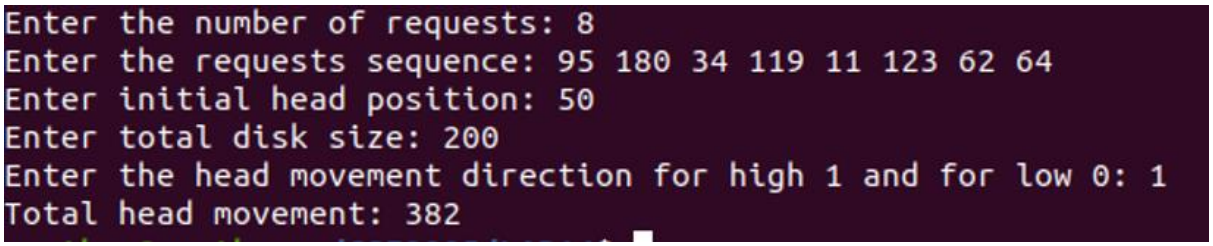
```
}
tot += abs(s-rq[i-1]-1) + abs(s-1); h=0;
for( i=0;i<pos;i++)
{
tot += abs(rq[i]-h); h=rq[i];

}
}
else if(dir == 0)
{
for(i=pos-1; i>=0; i--)
{
tot += abs(rq[i]-h); h=rq[i];
}
tot += abs(rq[i+1]-0) + abs(s-1); h = s-1;
for(i=n-1; i>=pos; i--)

{
tot += abs(rq[i]-h); h=rq[i];
}
}

printf("Total head movement: %d\n",tot);
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
Enter total disk size: 200
Enter the head movement direction for high 1 and for low 0: 1
Total head movement: 382