Experiment No. 9

Aim: Write a program in C demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU.

Theory:

FIFO(First In First Out):

- The simplest page-replacement algorithm and work on the basis of first in first out (FIFO). It throws out the pages in the order in which they were brought in.
- The time is associated with each page when it was brought into main memory.
- This algorithm always chooses oldest page for replacement.
- Since replacement is FIFO, a queue can be maintained to hold all the pages in main memory.
- This algorithm doesn't care about which pages are accessed frequently and which are not. However, it is used in windows 2000.

LRU(Least Recently Used):

- The time of page's last use is associated with each page.
- When a page must be replaced, LRU chooses that page that was used farthest back in the past.
- LRU is a good approximation to the optimal algorithm.
- This algorithm looks backward in time while optimal replacement algorithm looks forward in time.
- This policy suggests that replace a page whose last usage is farthest from current time.
- This algorithm can be implemented with some hardware support and is considered to be a good solution for page replacement.

- This algorithm does not suffer through Belady's anomaly.

FIFO Algorithm:

Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

1. Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
b) Simultaneously maintain the pages in the queue to perform FIFO.
c) Increment page fault

ii) Else

If current page is present in set, do nothing.

Else

a) Remove the first page from the queue as it was the first to be entered in the memory
b) Replace the first page in the queue with the current page in the string.
c) Store current page in the queue.
d) Increment page faults.

2. Return page faults.

LRU Algorithm:

Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

1. Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
b) Simultaneously maintain the recent occurred index of each page in a map called indexes.
c) Increment page fault

ii) Else

If current page is present in set, do nothing.

Else

a) Find the page in the set that was least recently used. We find it using index array. We basically need to replace the page with minimum index.
b) Replace the found page with current page.
c) Increment page faults.
d) Update index of current page.

2. Return page faults

Program:

FIFO:

```
#include<stdio.h>
int main()
{
 int reference_string[10], page_faults = 0, m, n, s, pages, frames;
 printf("\nEnter Total Number of Pages:\t");
 scanf("%d", &pages);
 printf("\nEnter values of Reference String:\n");
 for(m = 0; m < pages; m++)
 {

 scanf("%d", &reference_string[m]);
 }
 printf("\nEnter Total Number of Frames:\t");
 {
 scanf("%d", &frames);
 }
 int temp[frames];
 for(m = 0; m < frames; m++)
 {
 temp[m] = -1;
 }
 for(m = 0; m < pages; m++)
 {
 s = 0;
 for(n = 0; n < frames; n++)
 {
 if(reference_string[m] == temp[n]) {
 s++;
 page_faults--;
 }
 }
 page_faults++;
 if((page_faults <= frames) && (s == 0))
 {
 temp[m] = reference_string[m];
 }
 else if(s == 0)
 {
 temp[(page_faults - 1) % frames] = reference_string[m];
 }
 printf("\n");
 for(n = 0; n < frames; n++)
```

```
 {
 printf("%d\t", temp[n]);
 }
 }
 printf("\nTotal Page Faults:\t%d\n",
 page_faults); return 0;
}
```

LRU:

```
#include<stdio.h>
int findLRU(int time[], int n){
int i, minimum = time[0], pos =
0; for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum =
time[i]; pos = i;
}
}
return pos;
}
int main()
{
int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2,
i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
for(i = 0; i < no_of_pages; ++i){
scanf("%d", &pages[i]);
}
for(i = 0; i < no_of_frames; ++i){
frames[i] = -1;
}
for(i = 0; i < no_of_pages;
++i){ flag1 = flag2 = 0;
for(j = 0; j < no_of_frames; ++j){
if(frames[j] == pages[i]){
counter++;
time[j]          =
counter;  flag1  =
flag2     =     1;
break;}
}
if(flag1 == 0){
for(j = 0; j < no_of_frames; ++j){
if(frames[j] == -1){
counter++;
faults++;
frames[j] = pages[i];
time[j] = counter;
flag2 = 1;
break;
}
}
}
```

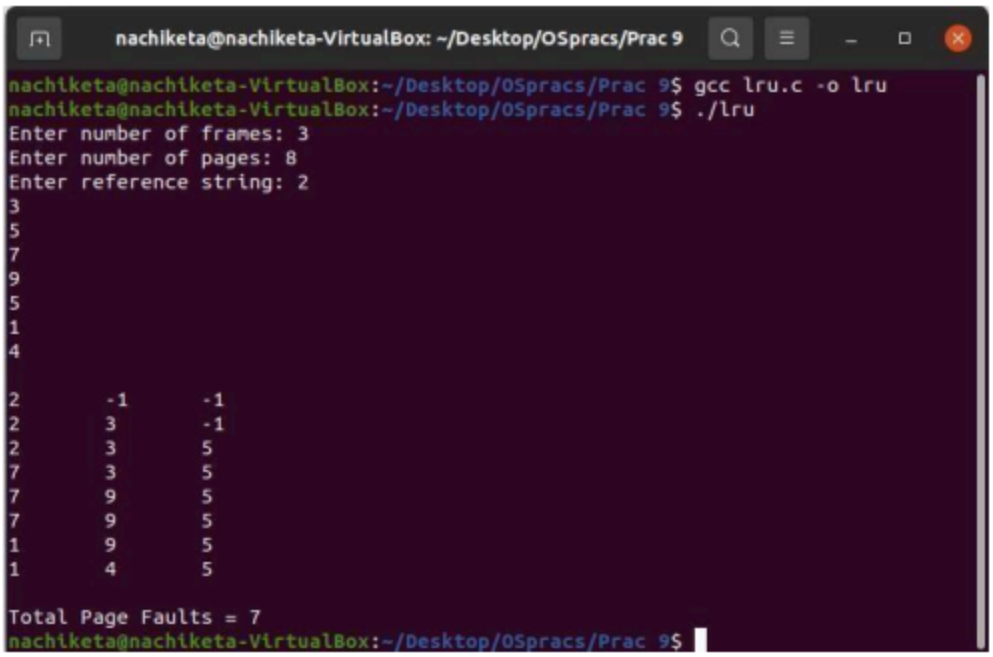
```
if(flag2 == 0){
pos = findLRU(time, no_of_frames);
counter++;
faults++;
frames[pos] =
pages[i]; time[pos] =
counter;
}
printf("\n");
for(j = 0; j < no_of_frames; ++j){
printf("%d\t", frames[j]);}
}
printf("\n\nTotal Page Faults = %d", faults);
printf("\n");
return 0;
}
```

Output:

FIFO:

```
nachiketa@nachiketa-VirtualBox: ~/Desktop/OSpracs/Prac 9
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac 9$ gcc fifo.c -o fifo
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac 9$ ./fifo

Enter Total Number of Pages:    8

Enter values of Reference String:
2
3
5
7
9
5
1
4

Enter Total Number of Frames:   3

2       -1      -1
2       3       -1
2       3       5
7       3       5
7       9       5
7       9       5
7       9       1
4       9       1
Total Page Faults:      7
nachiketa@nachiketa-VirtualBox:~/Desktop/OSpracs/Prac 9$
```

LRU:



Outcome:

Implemented various Page Replacement Algorithm and evaluated their performance.