EXPERIMENT NO. 7

AIM: Write a program to demonstrate Page Replacement policies i.e. FIFO, LRU for handling page faults.

RESOURCES REQUIRED:

H/W Requirements: P-IV and above, Ram 128 MB, Printer, Internet Connection.

S/W Requirements: Python Compiler.

THEORY:

1.FIFO Algorithm:

The simplest page-replacement algorithm is a FIFO algorithm. The first-in, first-out (FIFO) page replacement algorithm is a low-overhead algorithm that requires little book-keeping on the part of the operating system. The idea is obvious from the name – the operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the oldest arrival in front. When a page needs to be replaced, the page at the front of the queue (the oldest page) is selected. While FIFO is cheap and intuitive, it performs poorly in practical application. Thus, it is rarely used in its unmodified form. This algorithm experiences Bélády's anomaly.

FIFO page replacement algorithm is used by the VAX/VMS operating system, with some modifications. Partial second chance is provided by skipping a limited number of entries with valid translation table references and additionally, pages are displaced from process working set to a system wide pool from which they can be recovered if not already re-used.

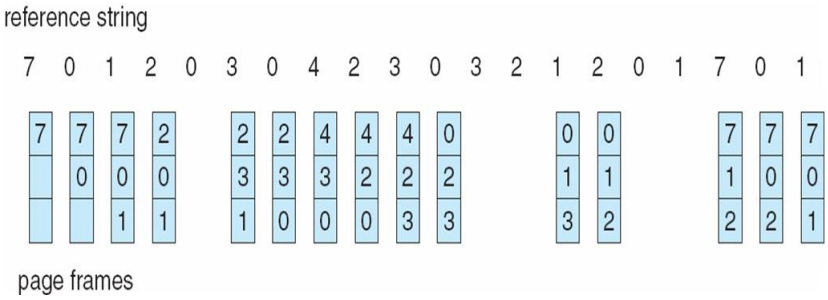
Treats page frames allocated to a process as a circular buffer:

When the buffer is full, the oldest page is replaced. Hence first-in, first-out:

A frequently used page is often the oldest, so it will be repeatedly paged out by FIFO.

Simple to implement, requires only a pointer that circles through the page frames of the process.

Consider the Example:



2.LRU Algorithm:

The least recently used page (LRU) replacement algorithm, though similar in name to NRU, differs in the fact that LRU keeps track of page usage over a short period, while NRU just looks at the usage in the last clock interval. LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too. While LRU can provide near-optimal performance in theory (almost as good as Adaptive Replacement Cache), it is rather expensive to implement in practice. There are a few implementation methods for this algorithm that try to reduce the cost yet keep as much of the performance as possible.

The most expensive method is the linked list method, which uses a linked list containing all the pages in memory. At the back of this list is the least recently used page, and at the front is the most recently used page. The cost of this implementation lies in the fact that items in the list must be moved about every memory reference, which is a very time-consuming process.

Because of implementation costs, one may consider algorithms (like those that follow) that are like LRU, but which offer cheaper implementations.

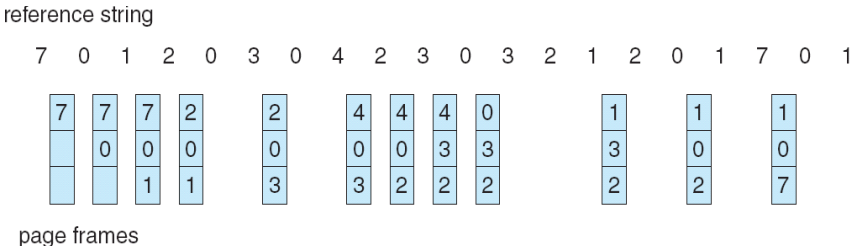
One important advantage of the LRU algorithm is that it is amenable to full statistical analysis. It has been proven, for example, that LRU can never result in more than N-times more page faults than OPT algorithm, where N is proportional to the number of pages in the managed pool.

On the other hand, LRU's weakness is that its performance tends to degenerate under many quite common reference patterns. For example, if there are N pages in the LRU pool, an application executing a loop over array of N + 1 pages will cause a page fault on each access. As loops over large arrays are common, much effort has been put into modifying LRU to work better in such situations. Many of the proposed LRU modifications try to detect looping reference patterns and to switch into suitable replacement algorithm, like Most Recently Used (MRU).

LRU Page Replacement Algorithm:

- Replaces the page that has not been referenced for the longest time:

By the principle of locality, this should be the page least likely to be referenced soon performs nearly as well as the optimal policy.



CONCLUSION: Hence, we have implemented a program on FIFO and LRU page replacement policies for handling page faults.

CODE:

1. FIFO:

```
class FIFO:
    def __init__(self,p,fs):
        self.p = p
        self.fs = fs
        self.n = len(self.p)

    def search(self,key,f):
        for i in range(self.fs):
            if f[i] == key:
                return True
        return False

    def fifo(self):
        f = [-1 for i in range(self.fs)]
        hit = 0
        k = 0
```

```
        for i in range(self.n):
            if(self.search(self.p[i],f)):
                hit += 1
                print(str(self.p[i])+"\t\t  No")
                continue
            if k<self.fs:
                f[k] = self.p[i]
                k += 1

            if k == self.fs:
                k =0
            print(str(self.p[i])+"\t"+str(f[0])+" "+str(f[1])+" "+str(f[2])+"\t  Yes")

        print("Number of hits = "+str(hit))
        print("Number of miss = "+str(self.n-hit))

if __name__ == '__main__':
    print("55_Adnan Shaikh")
    x = input("Enter elements of reference string use comma: ")
    p = [int(i.replace(" ","")) for i in x.split(',')]
    fs= int(input("Enter the frame size: "))

    fif = FIFO(p,fs)
    print("PAGE\tFRAME\tPAGE FAULT")
    fif.fifo()
```

2. LRU:

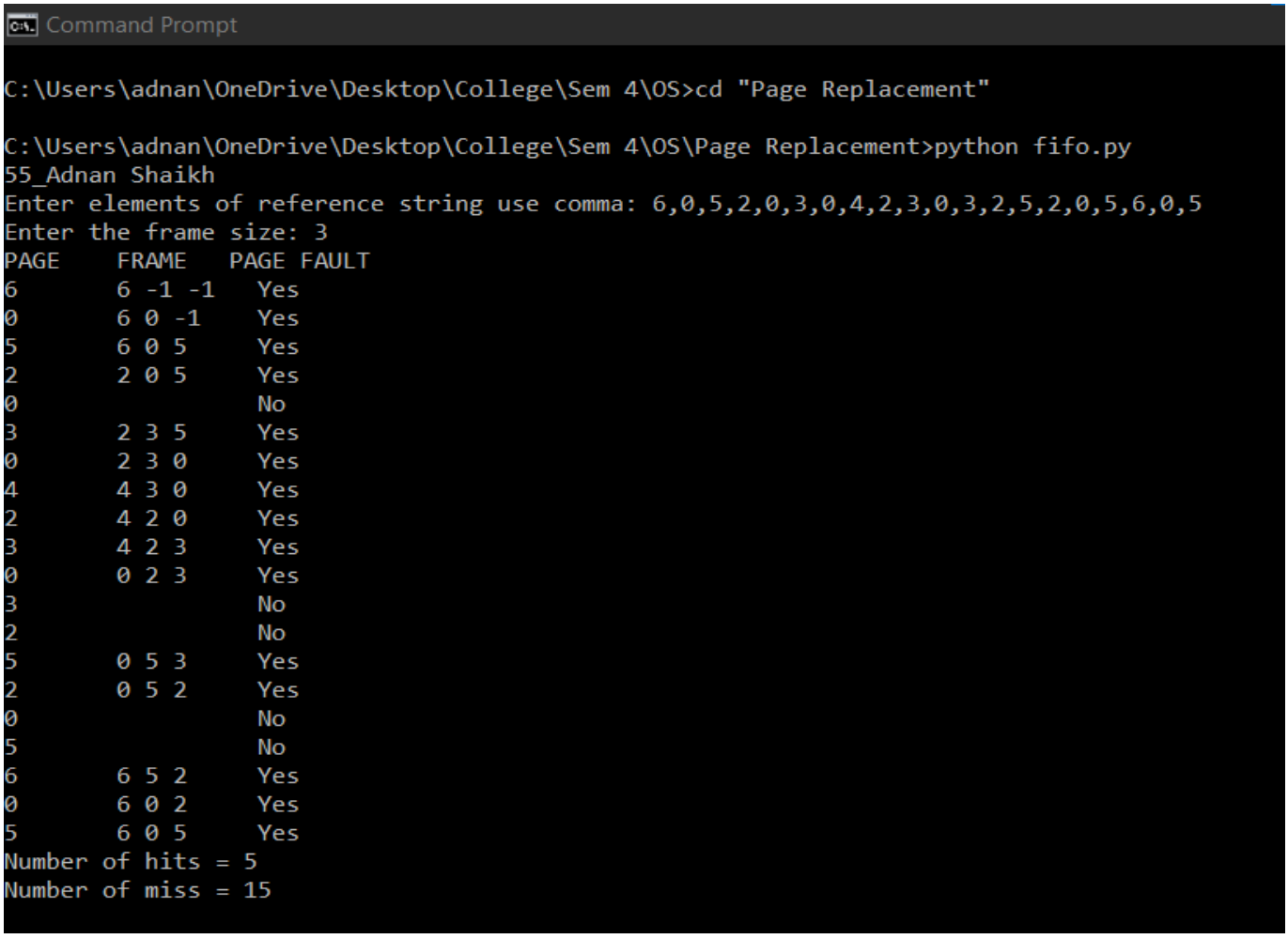
```
def lru():
    print("Enter the number of frames: ",end="")
    capacity = int(input())
    f,st,fault,pf = [],[],0,'No'
    print("Enter the reference string: ",end="")
    s = list(map(int,input().strip().split()))
    print("\nString|Frame →\t",end='')
    for i in range(capacity):
        print(i,end=' ')
    print("Fault\n   ↓\n")
    for i in s:
        if i not in f:
            if len(f)<capacity:
                f.append(i)
                st.append(len(f)-1)
            else:
                ind = st.pop(0)
                f[ind] = i
                st.append(ind)
            pf = 'Yes'
            fault += 1
        else:
            st.append(st.pop(st.index(f.index(i))))
            pf = 'No'
        print("   %d\t\t"%i,end='')
        for x in f:
```

```
            print(x,end=' ')
        for x in range(capacity-len(f)):
            print(' ',end=' ')
        print(" %s"%pf)
    print("\nTotal Requests: %d\nTotal Page Hit: %d\nTotal Page Faults: %d\nFault Rate: %0.2f%%"%(len(s),len(s)-fault,fault,(fault/len(s))*100))


if __name__ == "__main__":
    print("55_Adnan Shaikh")
    lru()
```

OUTPUT:

1. FIFO:



2. LRU:

```
Command Prompt

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Page Replacement>python lru.py
55_Adnan Shaikh
Enter the number of frames: 3
Enter the reference string: 6 0 5 2 0 3 0 4 2 3 0 3 2 5 2 0 5 6 0 5

String|Frame →  0 1 2 Fault
   ↓

   6            6      Yes
   0            6 0    Yes
   5            6 0 5  Yes
   2            2 0 5  Yes
   0            2 0 5  No
   3            2 0 3  Yes
   0            2 0 3  No
   4            4 0 3  Yes
   2            4 0 2  Yes
   3            4 3 2  Yes
   0            0 3 2  Yes
   3            0 3 2  No
   2            0 3 2  No
   5            5 3 2  Yes
   2            5 3 2  No
   0            5 0 2  Yes
   5            5 0 2  No
   6            5 0 6  Yes
   0            5 0 6  No
   5            5 0 6  No

Total Requests: 20
Total Page Hit: 8
Total Page Faults: 12
Fault Rate: 60.00%
```