Ex. No.: 11a)
Date:

FIFO PAGE REPLACEMENT

Aim:

To find out the number of page faults that occur using First-in First-out (FIFO) page replacement technique.

Algorithm:

1. Declare the size with respect to page length
2. Check the need of replacement from the page to memory
3. Check the need of replacement from old page to new page in memory 4.
Form a queue to hold all pages
5. Insert the page require memory into the queue
6. Check for bad replacement and page fault
7. Get the number of processes to be inserted
8. Display the values

Program Code:

```
   def fifo():
    ref_str = [int(input(f"Enter [{i+1}]: ")) for i in range(int(input("Enter the size
of reference string: ")))]
    frames = int(input("Enter page frame size: "))
    mem, ptr, faults = [], 0, 0

    for page in ref_str:
        if page not in mem:
            faults += 1
            if len(mem) < frames:
                mem.append(page)
            else:
                mem[ptr] = page
                ptr = (ptr + 1) % frames
            print(f"{page} -> {' '.join(map(str, mem))}")
        else:
            print(f"{page} -> No Page Fault")

    print(f"Total page faults: {faults}")

fifo()
```

Sample Output:

```
[root@localhost student]# python fifo.py
    Enter the size of reference string: 20
    Enter [ 1] : 7
    Enter [ 2] : 0
```

```
Enter [ 3] : 1
Enter [ 4] : 2
Enter [ 5] : 0
Enter [ 6] : 3
Enter [ 7] : 0
Enter [ 8] : 4
Enter [ 9] : 2
Enter [10] : 3
Enter [11] : 0
Enter [12] : 3
Enter [13] : 2
Enter [14] : 1
Enter [15] : 2
Enter [16] : 0
Enter [17] : 1
Enter [18] : 7
Enter [19] : 0
Enter [20] : 1

Enter page frame size : 3

7 -> 7 - -
0 -> 7 0 -
1 -> 7 0 1
2 -> 2 0 1
0 -> No Page Fault
3 -> 2 3 1
0 -> 2 3 0
4 -> 4 3 0
2 -> 4 2 0
3 -> 4 2 3
0 -> 0 2 3
3 -> No Page Fault
2 -> No Page Fault
1 -> 0 1 3
2 -> 0 1 2
0 -> No Page Fault
1 -> No Page Fault
7 -> 7 1 2
0 -> 7 0 2 1 -> 7 0 1
 Total page faults: 15.
[root@localhost student]#
```

Result :

The number of **page faults** was accurately determined using the **FIFO page replacement technique**, highlighting the importance of **page replacement strategies** in **virtual memory management**.

Ex. No.: 11b)
Date:

LRU

Aim:
To write a c program to implement LRU page replacement algorithm.

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 counter value
7: Stack them according the selection.
8: Display the values
9: Stop the process

Program Code:

```
#include <stdio.h>

int main() {
    int f, p, fau = 0, i, j, lru;
    printf("Enter frames & pages: ");
    scanf("%d %d", &f, &p);

    int ref[p], mem[f], cnt[f];
    printf("Enter ref string: ");
    for(i = 0; i < p; i++) scanf("%d", &ref[i]);

    for(i = 0; i < f; i++) mem[i] = -1, cnt[i] = 0;
    for(i = 0; i < p; i++) {
        for(j = 0; j < f; j++)
            if(mem[j] == ref[i]) { cnt[j] = i+1; break; }
        if(j == f) {
            fau++;
            for(lru = j = 0; j < f; j++)
                if(cnt[j] < cnt[lru]) lru = j;
            mem[lru] = ref[i];
            cnt[lru] = i+1;
        }
        for(j = 0; j < f; j++) printf("%d ", mem[j]);
        printf("\n");
    }
    printf("Total Page Faults = %d\n", fau);
}
```

Sample Output :

Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3
5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4

Result:

The number of **page faults** was accurately determined using the **LRU page replacement technique**, highlighting the importance of **page replacement strategies** in **virtual memory management**

Ex. No.: 11c)
Date: **Optimal**

Aim: To write a c program to implement Optimal page replacement algorithm.

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 frequently used page by counter value
7. Stack them according the selection.

8. Display the values
9. Stop the process

PROGRAM:

```c
#include <stdio.h>
int main() {
    int f,p,i,j,k,fa=0;
    printf("Enter frames & pages: ");
    scanf("%d%d",&f,&p);
    int r[p],m[f];
    printf("Ref string: ");
    for(i=0;i<p;i++) scanf("%d",&r[i]);
    for(i=0;i<f;i++) m[i]=-1;
    for(i=0;i<p;i++) {
        for(j=0;j<f;j++) if(m[j]==r[i]) break;
        if(j==f) {
            fa++;
            int farthest=i+1,idx=0;
            for(j=0;j<f;j++) {
                for(k=i+1;k<p;k++) if(m[j]==r[k]) break;
                if(k==p) {idx=j;break;}
                if(k>farthest) farthest=k,idx=j;
            }
            m[idx]=r[i];
        }
        for(j=0;j<f;j++) printf("%d ",m[j]);
        printf("\n");
    }
    printf("Faults: %d\n",fa);
    return 0;
}
```

Output:

```
Output

Page    Frames
7   7 - -
0   7 0 -
1   7 0 1
2   2 0 1
0   2 0 1
3   2 0 3
0   2 0 3
4   2 4 3
2   2 4 3
3   2 4 3

Total Page Faults = 6
```

Result:

An **LRU (Least Recently Used)** page replacement algorithm was implemented using **C**, successfully demonstrating the use of **temporal locality** for **efficient memory utilization**.