Ex. No.: 11a)

Date: 15/04/25

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:

```
#include <stdio.h>
#define MAX 100
int main() {


int refStr[MAX], frames[MAX];


   int n, frameSize, i, j, k;


   int pageFaults = 0, pointer = 0, found;


   printf("Enter the size of reference string: ");


   scanf("%d", &n);


   for (i = 0; i < n; i++) {
    printf("Enter [ %2d] : ", i + 1);

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

}



// Input number of frames

printf("Enter page frame size : ");

scanf("%d", &frameSize);



// Initialize frames to -1

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

    frames[i] = -1;


// Process reference string

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

    found = 0;



    // Check if page is already in frames
```

```
for (j = 0; j < frameSize; j++) {

    if (frames[j] == refStr[i]) {

        found = 1;

        break;

    }

}



if (!found) {

    // Page fault, replace oldest (FIFO)

    frames[pointer] = refStr[i];
    pointer = (pointer + 1) % frameSize;

    pageFaults++;



    // Print frame content

    printf("%d -> ", refStr[i]);

    for (k = 0; k < frameSize; k++) {

        if (frames[k] != -1)
```

```
                    printf("%d ", frames[k]);

                else

                    printf("- ");

            }

            printf("\n");
} else {
            printf("%d -> No Page Fault\n", refStr[i]);

        }

    }

    // Final result

    printf("Total page faults: %d\n", pageFaults);

return 0;

}
```

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 :

Thus the algorithm is executed successfully.