Experiment 9: FIFO Page Replacement Algorithm

AIM: To study & implement the FIFO Page Replacement Algorithm

THEORY:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in. Page replacement becomes necessary when a page fault occurs and there are no free page frames in memory. However, another page fault would arise if the replaced page is referenced again. Hence it is important to replace a page that is not likely to be referenced in the immediate future. y. If no page frame is free, the virtual memory manager performs a page replacement operation to replace one of the pages existing in memory with the page whose reference caused the page fault. It is performed as follows:

The virtual memory manager uses a page replacement algorithm to select one of the pages currently in memory for replacement, accesses the page table entry of the selected page to mark it as "not present" in memory, and initiates a page-out operation for it if the modified bit of its page table entry indicates that it is a dirty page.

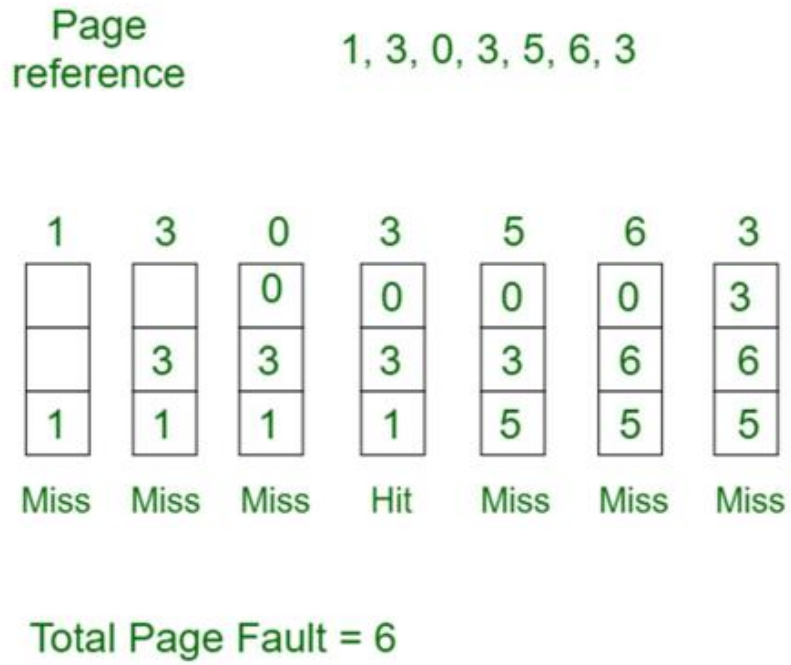
Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page.

Different page replacement algorithms suggest different ways to decide which page to replace.

The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms: 1. First In First Out (FIFO): This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames.Find the number of page faults



Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> 3 Page Faults. when 3 comes, it is already in memory so —> 0 Page Faults. Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 Page Fault. 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 Page Fault.

Finally, when 3 come it is not available so it replaces 0 1 page fault. Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example,

if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults. .

CODE:

```
#include<stdio.h>
int main()
{
   int incomingStream[] = {3, 0, 4, 3, 2, 1, 4, 6, 3, 0, 8, 9, 3,8,5};
   int pageFaults = 0;
   int frames = 3; //no. of frames
   int m, n, s, pages;
   pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
   //Reference string / size of incoming
   printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3\n");
   int temp[frames];
   for(m = 0; m < frames; m++)
   {
      temp[m] = -1;
   }
   //For entire reference string,
   for(m = 0; m < pages; m++)
```

```
    {
        s = 0;
        //evaluate for all pages ; set count s=0 (no.of hits) initially
        for(n = 0; n < frames; n++)
        {
            //If current page value of stream is present in temp,increment
hit (s) and decrease faults.
            if(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;
        if((pageFaults <= frames) && (s == 0))
        {
            temp[pageFaults - 1] = incomingStream[m];
        }
        //pagefaults are less than no.of frames and hits=0 then keep on
assigning m into temp array
        else if(s == 0)
        {
            temp[(pageFaults - 1) % frames] = incomingStream[m];
        }
        printf("%d\t\t\t", incomingStream[m]);
```

```
        for(n = 0; n < frames; n++)
        {
            if(temp[n] != -1)
                printf(" %d\t\t\t", temp[n]);
            else
                printf(" - \t\t\t");
        }
        printf("\n");
    }
    printf("Total Page Faults:\t%d\n", pageFaults);
    return 0;
}
```

Output:

```
/tmp/KjYeKCtaKB.o
Incoming     Frame 1     Frame 2     Frame 3
3            3           -           -
0            3           0           -
4            3           0           4
3            3           0           4
2            2           0           4
1            2           1           4
4            2           1           4
6            2           1           6
3            3           1           6
0            3           0           6
8            3           0           8
9            9           0           8
3            9           3           8
8            9           3           8
5            9           3           5
Total Page Faults:  12


=== Code Execution Successful ===
```

CONCLUSION:

Successfully Implemented Page Replacement policies for handling page faults.