Experiment No.9
Memory Management: Virtual Memory a Write a program in C demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU, Optimal
Date of Performance:
Date of Submission:
Marks:
Sign:

Aim: Memory Management: Virtual Memory

Objective:

To study and implement page replacement policy FIFO, LRU, OPTIMAL

Theory:

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.

First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the OS maintains a queue that keeps track of all the pages in memory, with the oldest page at the front and the most recent page at the back.

When there is a need for page replacement, the FIFO algorithm, swaps out the page at the front of the queue, that is the page which has been in the memory for the longest time.

Least Recently Used (LRU)

Least Recently Used page replacement algorithm keeps track of page usage over a short period of time. It works on the idea that the pages that have been most heavily used in the past are most likely to be used heavily in the future too.

In LRU, whenever page replacement happens, the page which has not been used for the longest amount of time is replaced.

Optimal Page Replacement

Optimal Page Replacement algorithm is the best page replacement algorithm as it gives the least number of page faults. It is also known as OPT, clairvoyant replacement algorithm, or Belady's optimal page replacement policy.

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future, i.e., the pages in the memory which are going to be referred farthest in the future are replaced.

This algorithm was introduced long back and is difficult to implement because it requires future knowledge of the program behavior. However, it is possible to implement optimal page replacement on the second run by using the page reference information collected on the first run.

Program:

```
#include<stdio.h>
   int main()
   {
      int incomingStream[] = {4 , 1 , 2 , 4 , 5};
      int pageFaults = 0;
      int frames = 3;
      int m, n, s, pages;
      pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
      printf(" Incoming \ t Frame 1 \ t Frame 2 \ t Frame 3 ");
      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(incomingStream[m] == temp[n])
            {
```

```
            s++;
            pageFaults--;
        }
    }
    pageFaults++;
    if((pageFaults <= frames) && (s == 0))
    {
        temp[m] = incomingStream[m];
    }
    else if(s == 0)
    {
        temp[(pageFaults - 1) % frames] = incomingStream[m];
    }
    printf("\n");
    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("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;  }
```

Output:

```
 Incoming  t Frame 1  t Frame 2  t Frame 3
4             4            -            -
1             4            1            -
2             4            1            2
4             4            1            2
5             5            1            2
Total Page Faults:  4
```

Conclusion:

Why do we need page replacement strategies ?

Page replacement strategies are essential in virtual memory management systems to efficiently utilize limited physical memory resources while providing the illusion of a larger memory space to user applications. The primary reason for employing page replacement strategies stems from the concept of virtual memory, which allows the operating system to use a portion of the disk as an extension of physical memory. However, physical memory is finite, and there are often more processes and data than can fit into it simultaneously.

Page replacement strategies come into play when a new page needs to be loaded into physical memory but there is no free space available. In such cases, the operating system must decide which page currently residing in physical memory to evict or replace with the new page.

Additionally, page replacement strategies play a crucial role in supporting multitasking environments, where multiple processes compete for limited physical memory resources. Without efficient page replacement mechanisms, system performance could degrade significantly due to excessive paging activity, leading to increased response times, thrashing, and overall degradation of system performance.

In summary, page replacement strategies are indispensable components of virtual memory management systems, enabling efficient utilization of physical memory resources, supporting multitasking environments, and maintaining acceptable system performance levels. They are vital for achieving the delicate balance between maximizing memory usage and minimizing the overhead associated with managing memory.