Experiment No. 7

Aim: To study and implement the first-in-first out (FIFO) Page replacement Algorithm.

Objective: To understand and implement the (FIFO) first-in-first-out Algorithm. & analyze it's performance based on page faults.

Aim: To study and implement the first-in first out (FIFO) Page replacement Algorithm.

Objective: To understand and implement the FIFO (first in first out) Algorithm. and it analyze it's performance based on page faults.

About FIFO:

Memory Management & Paging:

Modern operating systems (OS) use virtual memory and divide memory into fixed sized pages. When a page that a process needs is not in the main memory, a page fault occures, and the OS uses a page replacement algorithm to decide which page to remove from memory.

First in first out (FIFO) Page replacement Algorithm:

The FIFO algorithm is one of the simplest page replacement algorithms. It maintains a queue of pages in the order they were added to the memory. When a page needs to be replaced, the oldest page (the one at the front of the queue) is removed.

Key features:

i) Simple to implement

ii) Uses a queue to track the order of page entries.

iii) May suffer from "Belady's anomaly", where increasing the number of frames can lead to more page faults.

Important Terms:

i) Page fault: Occurs when a page is not found in the frame

ii) Page Hit: Occurs when the page is already present in a frame.

iii) Frames: Slots in main memory to hold pages.

Algorithm:

i) Initialize an empty queue for page frames

ii) For each page in the reference string:

- If the page is in the queue (Page hit), do nothing.
- If the page is NOT in the queue (Page fault):
- If there is space in the queue, add the Page.
- If the queue is full, remove the front Page & insert the new page.

iii) Count the number of page faults.

Code Implementation:

```
#include <iostream>
#include <queue>
#include <unordered_set>

using namespace std;
void fifoPageReplacement (int pages[], int n,
int capacity)
        unordered_set<int> s;
        queue<int> indexQueue;

        int pageFaults = 0;
for(int i=0; i<n; i++){
  if(s.size() < capacity){
    if(s.~~size~~find(pages[i] == s.end()){
      s.insert(pages[i]);
      indexQueue.push(pages[i]);
      pagesfault++;
    }
}else{
    if(s.find(pages[i]) == s.end()){
      int val = indexQueue.front();
      indexQueue.pop();
      s.erase(val);
      s.insert(pages[i]);
      indexQueue.push(pages[i]);
```

```
            pageFault++;
        }
    }
}
cout<<"Total Page faults :"<<pageFaults<<
    endl;
}
int main(){
    int pages[] = {1,3,0,3,5,6};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 3;
    fifoPageReplacement(pages, n, capacity);
return 0;
}
```

Execution & O/P:

Input: Reference string : 1, 3, 0, 3, 5, 6
Number of frames : 3

Output: Total Page faults : 5

Conclusion: The FIFO Page replacement algo. is the simple & easy to implement but may not always provide the best performance. Thus this execution was implemented successfully.

Conclusion: The FIFO page replacement algo is the simple & easy to implement but may not always provide the best performance. Thus this experiment was executed & implimented successfully.

22/4/24