Mawlana Bhashani Science and Technology University

Lab-Report

Report No: 11

Course code: ICT-3110

Course title: Operating Systems Lab

Date of Performance:

Date of Submission: 13/9/2020

Submitted by

Name: Mahfuza Talukdar

ID:IT-18009

3rd year 1st semester

Session: 2017-2018

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Experiment no : 11

Experiment name : Implementation of FIFO page replacement algorithm.

Theory :

In a computer operating system that uses paging for virtual memory management, **page replacement algorithms** decide which memory pages to page out, sometimes called swap out, or write to disk, when a page of memory needs to be allocated. Page replacement happens when a requested page is not in memory (page fault) and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.

When the page that was selected for replacement and paged out is referenced again it has to be paged in (read in from disk), and this involves waiting for I/O completion. This determines the *quality* of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm. A page replacement algorithm looks at the limited information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimize the total number of page misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself.

The page replacing problem is a typical online problem from the competitive analysis perspective in the sense that the optimal deterministic algorithm is known.

Corresponding Code :

```
// C++ implementation of FIFO page replacement

// in Operating Systems.

#include<bits/stdc++.h>

using namespace std;


// Function to find page faults using FIFO

int pageFaults(int pages[], int n, int capacity)

{

        // To represent set of current pages. We use

        // an unordered_set so that we quickly check

        // if a page is present in set or not
```

```
unordered_set<int> s;

// To store the pages in FIFO manner
queue<int> indexes;

// Start from initial page
int page_faults = 0;
for (int i=0; i<n; i++)
{
        // Check if the set can hold more pages
        if (s.size() < capacity)
        {
                // Insert it into set if not present
                // already which represents page fault
                if (s.find(pages[i])==s.end())
                {
                        // Insert the current page into the set
                        s.insert(pages[i]);

                        // increment page fault
                        page_faults++;

                        // Push the current page into the queue
                        indexes.push(pages[i]);
                }
        }
```

```
// If the set is full then need to perform FIFO
// i.e. remove the first page of the queue from
// set and queue both and insert the current page
else
{
        // Check if current page is not already
        // present in the set
        if (s.find(pages[i]) == s.end())
        {
                // Store the first page in the
                // queue to be used to find and
                // erase the page from the set
                int val = indexes.front();

                // Pop the first page from the queue
                indexes.pop();

                // Remove the indexes page from the set
                s.erase(val);

                // insert the current page in the set
                s.insert(pages[i]);

                // push the current page into
                // the queue
                indexes.push(pages[i]);
```

```
                        // Increment page faults
                        page_faults++;
                    }
                }
            }

        return page_faults;
}

// Driver code
int main()
{
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,
                        2, 3, 0, 3, 2};
        int n = sizeof(pages)/sizeof(pages[0]);
        int capacity = 4;
        cout << pageFaults(pages, n, capacity);
        return 0;
}
```

Output :

```
7
Process returned 0 (0x0)   execution time : 0.114 s
Press any key to continue.
```

Discussion :

From this lab, we have learnt that how to implement FIFO page replacement algorithm using C language. We can solve any problem of this algorithm in future.