OS Lab ASSIGNMENT-11

Name: Ashmit Thawait

Roll No: 102203790

Group: CO-17

Q. Simulate and compare the performance of three-page replacement algorithms – FIFO (First-In-First-Out), LRU (Least Recently Used), and Optimal – in a virtual memory management system.

Simulation and Analysis -

Students will run the implemented algorithms on the same set of page references and analyze their performance.

- Create a set of sample page reference sequences.
- Run each of the three algorithms on the same page reference sequence.
- Record the number of page faults and the pages replaced for each algorithm.
- Analyze and compare the results for the different algorithms.

Answer:

Output:

```
ashmit@ashmit-ubuntu:~$ cd Desktop/ashmit
ashmit@ashmit-ubuntu:~/Desktop/ashmit$ g++ assignment11.cpp
ashmit@ashmit-ubuntu:~/Desktop/ashmit$ ./a.out
FIFO Page Faults: 9
LRU Page Faults: 10
Optimal Page Faults: 7
ashmit@ashmit-ubuntu:~/Desktop/ashmit$
```

Code:

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

using namespace std;

// Function to simulate FIFO page replacement algorithm
void FIFO(const vector<int>& pages, int capacity) {
   queue<int> fifoQueue;
   unordered_set<int> pageSet;
   int pageFaults = 0;

   for (int page : pages) {
      if (pageSet.find(page) == pageSet.end()) {
         if (fifoQueue.size() == capacity) {
            int removedPage = fifoQueue.front();
            fifoQueue.pop();
            pageSet.erase(removedPage);
         }

         fifoQueue.push(page);
         pageSet.insert(page);
         pageFaults++;
      }
   }

   cout << "FIFO Page Faults: " << pageFaults << endl;
}

// Function to simulate LRU page replacement algorithm
void LRU(const vector<int>& pages, int capacity) {
   unordered_set<int> pageSet;
   vector<int> pageOrder;
   int pageFaults = 0;

   for (int page : pages) {
      if (pageSet.find(page) == pageSet.end()) {
         if (pageOrder.size() == capacity) {
            int leastRecentlyUsed = pageOrder.front();
            pageOrder.erase(remove(pageOrder.begin(), pageOrder.end(), leastRecentlyUsed),
pageOrder.end());
            pageSet.erase(leastRecentlyUsed);
         }

         pageOrder.push_back(page);
         pageSet.insert(page);
```

```
            pageFaults++;
        } else {
            // If the page is already in the set, move it to the front to mark it as recently used
            pageOrder.erase(remove(pageOrder.begin(), pageOrder.end(), page), pageOrder.end());
            pageOrder.push_back(page);
        }
    }

    cout << "LRU Page Faults: " << pageFaults << endl;
}

// Function to simulate Optimal page replacement algorithm
void Optimal(const vector<int>& pages, int capacity) {
    unordered_set<int> pageSet;
    vector<int> futurePages(pages.begin(), pages.end());
    int pageFaults = 0;

    for (int i = 0; i < pages.size(); ++i) {
        if (pageSet.find(pages[i]) == pageSet.end()) {
            if (pageSet.size() == capacity) {
                int farthestIndex = -1, farthestPage = -1;

                for (int page : pageSet) {
                    auto found = find(futurePages.begin() + i, futurePages.end(), page);
                    if (found == futurePages.end()) {
                        farthestPage = page;
                        break;
                    }
                    else {
                        int index = distance(futurePages.begin(), found);
                        if (index > farthestIndex) {
                            farthestIndex = index;
                            farthestPage = page;
                        }
                    }
                }

                pageSet.erase(farthestPage);
            }

            pageSet.insert(pages[i]);
            pageFaults++;
        }
    }

    cout << "Optimal Page Faults: " << pageFaults << endl;
}

int main() {
    // Generate a sample page reference sequence
    vector<int> pageReferences = {1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5};
```

```
    // Set the page frame capacity
    int capacity = 3;

    // Run each algorithm on the same page reference sequence
    FIFO(pageReferences, capacity);
    LRU(pageReferences, capacity);
    Optimal(pageReferences, capacity);

    return 0;
}
```