OPERATING SYSTEMS

Lab 12 : Page Replacement Algorithms

Name – Vinay Santosh Menon
Registration Number - 20BAI1103

TASK:

To write a C/C++ code for FIFO, LRU and Optimal

FIRST IN FIRST OUT – (FIFO)

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, 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.

CODE

```
#include<iostream>
#include<unordered_set>
#include<algorithm>
#include<queue>
using namespace std;

int pageFaults(int pages[], int n, int capacity)
{
    unordered_set<int> s;
    queue<int> indexes;
    int page_faults = 0;
    for (int i=0; i<n; i++)
    {
        if (s.size() < capacity)
        {
            if (s.find(pages[i])==s.end())
            {
                s.insert(pages[i]);
                page_faults++;
                indexes.push(pages[i]);
            }
        }
        else
        {
            if (s.find(pages[i]) == s.end())
            {
                int val = indexes.front();
                indexes.pop();
                s.erase(val);
                s.insert(pages[i]);
                indexes.push(pages[i]);
                page_faults++;
            }
        }
    }
```

```
    return page_faults;
}

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;
}
```

RESULT

```
PS C:\Users\vinay\OneDrive\Documents\code\OS>
7
```

LEAST RECENTLY USED

In operating systems that use paging for memory management, page replacement algorithm is needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.
In **L**east **R**ecently **U**sed (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely

CODE

```
#include<bits/stdc++.h>
using namespace std;

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

```
    unordered_set<int> s;

    unordered_map<int, int> indexes;

    int page_faults = 0;
    for (int i=0; i<n; i++)
    {
        if (s.size() < capacity)
        {
            if (s.find(pages[i])==s.end())
            {
                s.insert(pages[i]);


                page_faults++;
            }
            indexes[pages[i]] = i;
        }

        else
        {
            if (s.find(pages[i]) == s.end())
            {

                int lru = INT_MAX, val;
                for (auto it=s.begin(); it!=s.end(); it++)
                {
                    if (indexes[*it] < lru)
                    {
                        lru = indexes[*it];
                        val = *it;
                    }
                }

                s.erase(val);
                s.insert(pages[i]);
                page_faults++;
            }
            indexes[pages[i]] = i;
        }
    }
    return page_faults;
}

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;
    }
```

RESULT

```
PS C:\Users\vinay\OneDrive\Documents\code\OS\.vscode>
U }
6
```

OPTIMAL

CODE

```
#include <bits/stdc++.h>
using namespace std;

bool search(int key, vector<int>& fr)
{
    for (int i = 0; i < fr.size(); i++)
        if (fr[i] == key)
            return true;
    return false;
}

int predict(int pg[], vector<int>& fr, int pn, int index)
{
    int res = -1, farthest = index;
    for (int i = 0; i < fr.size(); i++) {
        int j;
        for (j = index; j < pn; j++) {
            if (fr[i] == pg[j]) {
                if (j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }

        if (j == pn)
            return i;
    }
```

```
    return (res == -1) ? 0 : res;
}

void optimalPage(int pg[], int pn, int fn)
{
    vector<int> fr;

    int hit = 0;
    for (int i = 0; i < pn; i++) {
        if (search(pg[i], fr)) {
            hit++;
            continue;
        }
        if (fr.size() < fn)
            fr.push_back(pg[i]);

        else {
            int j = predict(pg, fr, pn, i + 1);
            fr[j] = pg[i];
        }
    }
    cout << "No. of hits = " << hit << endl;
    cout << "No. of misses = " << pn - hit << endl;
}

int main()
{
    int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
    int pn = sizeof(pg) / sizeof(pg[0]);
    int fn = 4;
    optimalPage(pg, pn, fn);
    return 0;
}
```

RESULT

```
PS C:\Users\vinay\OneDrive\Documents\code\OS\.vscode
) { .\optimal }
No. of hits = 7
No. of misses = 6
```