Name : Ruturaj Sandip Sutar **Roll No : 59** **Div : B**

Batch : 2 **PRN:-12310720**

Implement page replacement algorithms

1. FIFO

Code:-

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

void fifoPageReplacement(int pages[], int n, int capacity) {
    unordered_set<int> s;
    queue<int> indexes;
    int pageFaults = 0;

    for (int i = 0; i < n; i++) {
        if (s.size() < capacity) {
            if (s.find(pages[i]) == s.end()) {
                s.insert(pages[i]);
                indexes.push(pages[i]);
                pageFaults++;
            }
        } 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]);
                pageFaults++;
            }
        }
    }
    cout << "Total Page Faults (FIFO): " << pageFaults << endl;
}
```

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

Output:-

```
Total Page Faults (FIFO): 11
```

2. Optimal

Code:-

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int predict(int pages[], vector<int>& frame, int n, int index) {
    int res = -1, farthest = index;
    for (int i = 0; i < frame.size(); i++) {
        int j;
        for (j = index; j < n; j++) {
            if (frame[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }
        if (j == n) return i;
    }
    return (res == -1) ? 0 : res;
}

void optimalPageReplacement(int pages[], int n, int capacity) {
    vector<int> frame;
    int pageFaults = 0;

    for (int i = 0; i < n; i++) {
```

```
        if (find(frame.begin(), frame.end(), pages[i]) == frame.end()) {
            if (frame.size() < capacity) {
                frame.push_back(pages[i]);
            } else {
                int j = predict(pages, frame, n, i + 1);
                frame[j] = pages[i];
            }
            pageFaults++;
        }
    }
    cout << "Total Page Faults (Optimal): " << pageFaults << endl;
}

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

Output:-

```
Total Page Faults (Optimal): 9
```

3. LRU

Code:-

```
#include <iostream>
#include <unordered_map>
#include <list>
using namespace std;

void lruPageReplacement(int pages[], int n, int capacity) {
    unordered_map<int, list<int>::iterator> indexes;
    list<int> pageList;
    int pageFaults = 0;

    for (int i = 0; i < n; i++) {
        if (indexes.find(pages[i]) == indexes.end()) {
            if (pageList.size() == capacity) {
                int last = pageList.back();
```

```
            pageList.pop_back();
            indexes.erase(last);
        }
        pageList.push_front(pages[i]);
        indexes[pages[i]] = pageList.begin();
        pageFaults++;
    } else {
        pageList.erase(indexes[pages[i]]);
        pageList.push_front(pages[i]);
        indexes[pages[i]] = pageList.begin();
    }
  }
  cout << "Total Page Faults (LRU): " << pageFaults << endl;
}

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

Output:-

```
Total Page Faults (LRU): 12
```

4. Clock

Code:-

```
#include <iostream>
#include <vector>
using namespace std;

void clockPageReplacement(int pages[], int n, int capacity) {
  vector<int> frames(capacity, -1);
  vector<bool> secondChance(capacity, false);
  int pointer = 0, pageFaults = 0;

  for (int i = 0; i < n; i++) {
    int page = pages[i];
    bool found = false;
```

```
        for (int j = 0; j < capacity; j++) {
            if (frames[j] == page) {
                secondChance[j] = true;
                found = true;
                break;
            }
        }

        if (!found) {
            while (secondChance[pointer]) {
                secondChance[pointer] = false;
                pointer = (pointer + 1) % capacity;
            }
            frames[pointer] = page;
            secondChance[pointer] = true;
            pointer = (pointer + 1) % capacity;
            pageFaults++;
        }
    }
    cout << "Total Page Faults (Clock): " << pageFaults << endl;
}

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

Output:-

```
Total Page Faults (Clock): 10
```