Name – Krishnaprasad Awala

Enrollment no – MITU21BTITD006

Roll no – D2215004

Lab 4

Code :

#include <stdio.h>

int main()

{

int incomingStream[] = {4 , 1 , 2 , 4 , 5};

int pageFaults = 0;

int frames = 3;

int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

printf(" Incoming \ t Frame 1 \ t Frame 2 \ t Frame 3 ");

int temp[ frames ];

for(m = 0; m < frames; m++)

{

temp[m] = -1;

}

for(m = 0; m < pages; m++)

{

s = 0;

for(n = 0; n < frames; n++)

{

if(incomingStream[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = incomingStream[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = incomingStream[m];

}

printf("\n");

printf("%d\t\t\t",incomingStream[m]);

for(n = 0; n < frames; n++)

{

if(temp[n] != -1)

printf(" %d\t\t\t", temp[n]);

else

printf(" - \t\t\t");

}

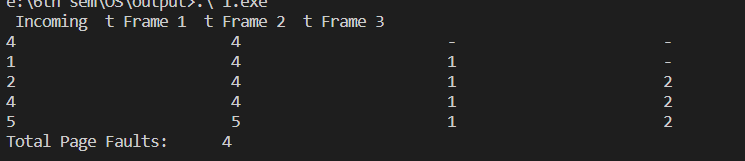
}

printf("\nTotal Page Faults:\t%d\n", pageFaults);

return 0;

}

Output :



LRU :

Code :

//C++ implementation of above algorithm

#include<bits/stdc++.h>

using namespace std;

// Function to find page faults using indexes

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 least recently used indexes

// of pages.

unordered\_map<int, 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())

{

s.insert(pages[i]);

// increment page fault

page\_faults++;

}

// Store the recently used index of

// each page

indexes[pages[i]] = i;

}

// If the set is full then need to perform lru

// i.e. remove the least recently used page

// and insert the current page

else

{

// Check if current page is not already

// present in the set

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

{

// Find the least recently used pages

// that is present in the set

int lru = INT\_MAX, val;

for (auto it=s.begin(); it!=s.end(); it++)

{

if (indexes[\*it] < lru)

{

lru = indexes[\*it];

val = \*it;

}

}

// Remove the indexes page

s.erase(val);

// insert the current page

s.insert(pages[i]);

// Increment page faults

page\_faults++;

}

// Update the current page index

indexes[pages[i]] = i;

}

}

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 :



Optimal Page:

#include <bits/stdc++.h>

using namespace std;

int predict(int page[], vector<int>& fr, int pn, int index) {

// Store the index of pages which are going

// to be used recently in future

int res = -1, farthest = index;

for (int i = 0; i < fr.size(); i++) {

int j;

for (j = index; j < pn; j++) {

if (fr[i] == page[j]) {

if (j > farthest) {

farthest = j;

res = i;

}

break;

}

}

// Return the page which are

// are never referenced in future,

if (j == pn)

return i;

}

// If all of the frames were not in future,

// return any of them, we return 0. Otherwise

// we return res.

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

}

bool search(int key, vector<int>& fr) {

for (int i = 0; i < fr.size(); i++)

if (fr[i] == key)

return true;

return false;

}

void opr(int page[], int pn, int fn) {

vector<int> fr;

int hit = 0;

for (int i = 0; i < pn; i++) {

// Page found in a frame : HIT

if (search(page[i], fr)) {

hit++;

continue;

}

//If a page not found in a frame : MISS

// check if there is space available in frames.

if (fr.size() < fn)

fr.push\_back(page[i]);

// Find the page to be replaced.

else {

int j = predict(page, fr, pn, i + 1);

fr[j] = page[i];

}

}

cout << "Hits = " << hit << endl;

cout << "Misses = " << pn - hit << endl;

}

// main Function

int main() {

int page[] = { 1, 7, 8, 3, 0, 2, 0, 3, 5, 4, 0, 6, 1 };

int pn = sizeof(page) / sizeof(page[0]);

int fn = 3;

opr(page, pn, fn);

return 0;

}

Output:

