**OPERATING SYSTEMS**

LAB EXPERIMENT - 9

Submitted To: Prashant Giridhar Shambharkar Sir.

Submitted By: Manav Agrawal (2K19/CO/216)

---------------------------------------------------------------------

Aim:

Write a C program to simulate page replacement algorithms a) LFU b) Optimal

Introduction:

Page Replacement Algorithms:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

The following are the page replacement algorithms which we will be implementing:

### Least Frequently Used (LFU):

In LFU Page Replacement method, the page with the minimum count is selected for replacement with the page that needs to enter into the system. LFU is a cache algorithm which is used to manage computer’s memory. A counter is assigned to every block of memory that is loaded into the cache memory. However, the LFU technique is hardly implemented these days but this algorithm is normally combined with other algorithms which make it a hybrid algorithm, and then it is implemented. LFU algorithm is sometimes also combined with LRU replacement algorithm, and then implemented.

### Optimal:

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

*Example-* Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults

0 is already there so —> 0 Page fault.

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>1 Page fault.

0 is already there so —> 0 Page fault..

4 will takes place of 1 —> 1 Page Fault.

Now for the further page reference string —> 0 Page fault because they are already available in the memory.

Algorithms:

1. **Least Frequently Used (LFU):**

In LFU Page Replacement method, the page with the minimum count is selected for replacement with the page that needs to enter into the system.

LFU is a cache algorithm which is used to manage computer’s memory. A counter is assigned to every block of memory that is loaded into the cache memory.

ALGORITHM:

1 - In the current stack at any iteration choose that element

for replacement which has the smallest count in the incoming

page stream.

2 - Check the old page as well as the frequency of that Page.

1. If the frequency of the page is larger than the old page do

not remove it.

2. If all the old pages are having the same frequency then take

the last i.e FIFO method for that and remove that page.

1. **Optimal:**

In this algorithm, OS replaces the page that will not be used for the longest period of time in future.

*Example :*

**Input :** Number of frames, fn = 3

Reference String, pg[] = {7, 0, 1, 2,

0, 3, 0, 4, 2, 3, 0, 3, 2, 1,

2, 0, 1, 7, 0, 1};

**Output :** No. of hits = 11

No. of misses = 9

The main idea is for every reference we do following :

1 - If the referred page is already present, increment hit

count.

2 - If not present, find a page that is never referenced in Future.

1. If such a page exists, replace this page with a new page.

2. If no such page exists, find a page that is referenced farthest

in future.

3 - Replace this page with a new page

Implementation:

**A) Least Frequently Used (LFU):**

#include<stdio.h>

int main()

{

int total\_frames, total\_pages, hit = 0;

int pages[25], frame[10], arr[25], time[25];

int m, n, page, flag, k, minimum\_time, temp;

printf("Enter Total Number of Pages:\t");

scanf("%d", &total\_pages);

printf("Enter Total Number of Frames:\t");

scanf("%d", &total\_frames);

for(m = 0; m < total\_frames; m++)

{

frame[m] = -1;

}

for(m = 0; m < 25; m++)

{

arr[m] = 0;

}

printf("Enter Values of Reference String\n");

for(m = 0; m < total\_pages; m++)

{

printf("Enter Value No.[%d]:\t", m + 1);

scanf("%d", &pages[m]);

}

printf("\n");

for(m = 0; m < total\_pages; m++)

{

arr[pages[m]]++;

time[pages[m]] = m;

flag = 1;

k = frame[0];

for(n = 0; n < total\_frames; n++)

{

if(frame[n] == -1 || frame[n] == pages[m])

{

if(frame[n] != -1)

{

hit++;

}

flag = 0;

frame[n] = pages[m];

break;

}

if(arr[k] > arr[frame[n]])

{

k = frame[n];

}

}

if(flag)

{

minimum\_time = 25;

for(n = 0; n < total\_frames; n++)

{

if(arr[frame[n]] == arr[k] && time[frame[n]] < minimum\_time)

{

temp = n;

minimum\_time = time[frame[n]];

}

}

arr[frame[temp]] = 0;

frame[temp] = pages[m];

}

for(n = 0; n < total\_frames; n++)

{

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

}

printf("\n");

}

printf("Page Hit:\t%d\n", hit);

return 0;

}

**B) Optimal:**

#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;

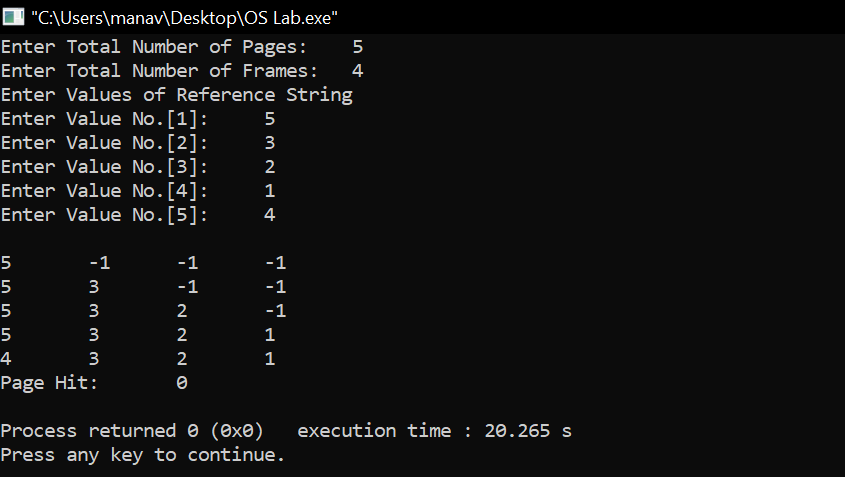
optimalPage(pg, pn, fn);

return 0;

}

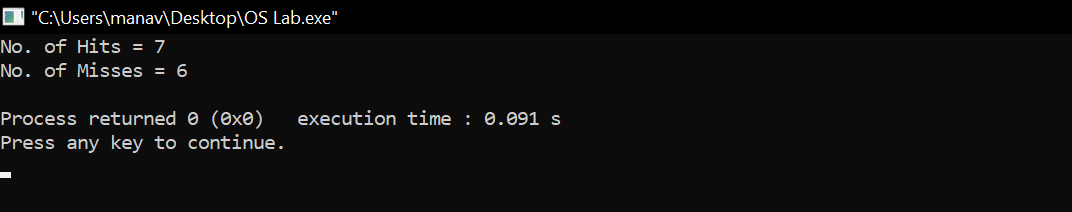
Output:

**A) Least Frequently Used (LFU):**

****

**B) Optimal:**

Input - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

****

Learning From The Experiment:

While the LFU method may seem like an intuitive approach to memory management it is not without faults. Consider an item in memory which is referenced repeatedly for a short period of time and is not accessed again for an extended period of time. Due to how rapidly it was just accessed its counter has increased drastically even though it will not be used again for a decent amount of time. This leaves other blocks which may actually be used more frequently susceptible to purging simply because they were accessed through a different method.

In Optimal Page replacement, complexity is less and it is easy to implement. Assistance needed is low i.e Data Structure used are easy and light. Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

***THANK YOU!***