**EXPERIMENT 8**

**OBJECTIVE**:-

Write a program to find page fault for a given string of pages (size 20).

Assume the memory frames is an array of size n. Page fault algorithms to be

used: 1.FIFO 2.LRU with counters or stack. What are the effect of increasing

the number of memory frames?

**CODE**:-

//------------------------------------------------------------------------------

//

// Desc: page\_replacement.c

//

// Date: 29/3/2018

//

// Author: VU DUY DU - 2k16/CO/364

//

//------------------------------------------------------------------------------

#include<iostream>

#include<vector>

#include<queue>

#include<ctime>

struct page\_slot{

int page;

int age;//aging counter

page\_slot(int page)

:page(page),age(0){}

};

int page\_faults\_FIFO(int \*pages, int n, int capacity){

int fault\_num = 0;

std::queue<int> fifo\_queue;

std::vector<int> to\_access\_element\_in\_queue;

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

//std::cout<<pages[i]<<" - ";

bool page\_fault = true;

for(size\_t j = 0; j<fifo\_queue.size(); j++){

if(pages[i] == to\_access\_element\_in\_queue[j]){

page\_fault = false;

//std::cout<<"page hit\n";

break;

}

}

if(page\_fault){

//std::cout<<"page fault\n";

fault\_num++;

if((int)fifo\_queue.size()<capacity){

fifo\_queue.push(pages[i]);

to\_access\_element\_in\_queue.push\_back(pages[i]);

}else{

fifo\_queue.push(pages[i]);

int p = fifo\_queue.front();

fifo\_queue.pop();

for(size\_t j = 0; j<fifo\_queue.size(); j++)

if(p == to\_access\_element\_in\_queue[j])

to\_access\_element\_in\_queue[j] = pages[i];

}

}

}

return fault\_num;

}

int page\_faults\_LRU(int \*pages, int n, int capacity){

int fault\_num = 0;

std::vector<page\_slot> page\_holder;

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

//std::cout<<pages[i]<<" - ";

if((int)page\_holder.size()<capacity){

bool page\_fault = true;

for(size\_t j = 0; j<page\_holder.size();j++){

if(pages[i]==page\_holder[j].page){

page\_fault = false;

page\_holder[j].age = 0;

//std::cout<<"page hit\n";

break;

}

}

if(page\_fault){

page\_holder.push\_back(page\_slot(pages[i]));

fault\_num++;

//std::cout<<"page fault\n";

}

}else{

bool page\_fault = true;

int oldest = 0, oldest\_index = 0;

for(size\_t j = 0; j<page\_holder.size();j++){

if(pages[i]==page\_holder[j].page){

page\_fault = false;

page\_holder[j].age = 0;

//std::cout<<"page hit\n";

break;

}

if(oldest<page\_holder[j].age){

oldest = page\_holder[j].age;

oldest\_index = j;

}

}

if(page\_fault){//this is where page fault occurs.

//replace the current page with the oldest page

page\_holder[oldest\_index].page = pages[i];

page\_holder[oldest\_index].age = 0;

fault\_num++;

//std::cout<<"page fault\n";

}

}

//aging all the pages inside the page holder

for(size\_t j = 0; j<page\_holder.size();j++)

page\_holder[j].age++;

}

return fault\_num;

}

int main(){

std::cout<<"Hello FIFO and LRU\n";

int n = 20;

int pages[20];

//generate random pages reference

srand(time(0));

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

pages[i] = (rand()%10000)%8; //0-8

std::cout<<pages[i]<<", ";

}

int capacity = 2;

for(;capacity<10; capacity++)

std::cout<<"\nFIFO: "

<<page\_faults\_FIFO(pages,n,capacity)

<<"\nLRU: "

<<page\_faults\_LRU(pages,n,capacity)<<"\n";

return 0;

}

**OUTPUT:-**

Hello FIFO and LRU

3, 6, 6, 5, 2, 0, 1, 7, 3, 4, 0, 5, 4, 2, 0, 1, 7, 7, 7, 7,

FIFO: 16

LRU: 16

FIFO: 14

LRU: 15

FIFO: 14

LRU: 14

FIFO: 14

LRU: 13

FIFO: 14

LRU: 13

FIFO: 8

LRU: 8

FIFO: 8

LRU: 8

FIFO: 8

LRU: 8

**DISCUSSION:-**

In the program, the two functions implementing FIFO and LRU algorithms are run 8 times, as the capacity increase from 2 – 10. Since the number of pages is fixed (in this particular case is 8), so when the capacity increases, the page fault decreases,

*Source code of this experiment can be found here:*

*https://github.com/Sieunguoimay/OSLab-4thsem-2018/tree/master/Exp8*