



**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA HỆ THỐNG THÔNG TIN**

**🙢 🕮 🙠**

**-Logo

Description automatically generated**

**THỰC HÀNH HỆ ĐIỀU HÀNH**

**Bài thực hành số 6**

**Họ và tên: Lê Anh Thư**

**MSSV: 20521985**

**Giảng viên: Nguyễn Thanh Nam**

**Phần I – Hướng dẫn thực hành:**

Sử dụng ngôn ngữ lập trình C viết chương trình mô phỏng các giải thuật thay thế trang đã nêu trong câu số 3 mục 6.3.3 với các yêu cầu sau:

- Giao diện ban đầu của chương trình được thể hiện như bên dưới:

**--- Page Replacement algorithm ---**

1. Default referenced sequence

2. Manual input sequence

Trong đó “Default referenced sequence” được sinh ra từ mã số sinh viên kèm với mã “007”, ví dụ sinh viên có mã số là 13520462 thì “Default referenced sequence” sẽ là: 1, 3, 5, 2, 0, 4, 6, 2, 0, 0, 7.

“Manual input sequence” là chuỗi được người dùng nhập vào.

- Sau khi lựa chọn chuỗi tham chiếu, chương trình chuyển sang giao diện cho phép người dùng nhập số khung trang:

**--- Page Replacement algorithm ---**

Input page frames:

- Sau khi lựa chọn khung trang, chương trình chuyển sang giao diện cho phép người dùng lựa chọn giải thuật thay thế trang:

**--- Page Replacement algorithm ---**

1. FIFO algorithm

2. OPT algorithm

3. LRU algorithm

- Sau khi lựa chọn giải thuật sử dụng, chương trình hiện kết quả như sau, ví dụ: Có chuỗi tham chiếu: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Có 3 khung trang và chọn giải thuật FIFO chương trình hiển thị output như sau:

**--- Page Replacement algorithm ---**

1 2 3 4 1 2 5 1 2 3 4 5

1 1 1 4 4 4 5 5 5 5 5 5

2 2 2 1 1 1 1 1 3 3 3

3 3 3 2 2 2 2 2 4 4

\* \* \* \* \* \* \* \* \*

Number of Page Fault: 9

triển khai trước phần code ở phía dưới, sau đó sẽ giải thích từng phần tương ứng với bài.

**Bài làm**

Giải thích sơ lược về code:

Tất cả các biến toàn cục và thư viện sẽ sử dụng được khai báo bên dưới như sau:

* defaulData[]: Là biến cho dãy số mặc định nếu người dùng chọn **“Default referenced sequence”** trong giao diện ban đầu

**Dãy số mặc định sẽ phụ thuộc vào MSSV với công thức: MSSV+007**

**MSSV = 20521985 => dãy số mặc định: 20521985007**

* n: là số lượng số trong số trong dãy số mặc định
* frameNum: là số frames
* input[]: là dãy số chúng ta dùng để xử lý
* p[]: Là dãy số hiện tại, tại mỗi đoạn
* hit: Là biến để kiểm tra xem số hiện tại đã có trong p[] hay không
* faultCount: Biến đếm số trang lỗi
* tg[][]: Mảng 2 chiều, dùng để lưu trữ toàn bộ quá trình mà thuật toán thực hiện
* pf[]: Mảng 1 chiều, dùng để lưu trữ lỗi.

Code triển khai:

* Hình ảnh:

A screenshot of a computer

Description automatically generated

* Source code:

#include <stdio.h>

char defaultData[] = "20521985007";

int frameNum;

int n = 11;

int faultCount = 0;

int input[100];

int p[50];

int pf[100];

int tg[100][100];

int hit = 0;

1. **Giao diện ban đầu của chương trình:**

Code triển khai:

* Hình ảnh:

Graphical user interface, text

Description automatically generated

* Source code:

int main(){

int choice = 0;

while(choice == 0){

printf("---Page Replacement Algorithm---\n");

printf("1. Default referenced sequence\n");

printf("2. Manual input sequence\n");

scanf("%d", &choice);

switch(choice){

case 1:

getDataDefault();

break;

case 2:

getDataManual();

break;

default:

printf("Invalid choose: Please type again!\n");

choice = 0;

break;

}

}

return 0;

}

* Giải thích code:

Trong hàm main, ta sử dụng vòng lặp while để người dùng lựa chọn:

* **Trường hợp 1**: Khi người dùng nhập vào “1”: Chương trình sẽ chọn dãy số mặc định tương ứng là: “mssv kèm với mã 007” **– “2, 0 , 5, 2, 1, 9, 8, 5, 0, 0, 7”**

**Hình ảnh minh họa:**

Text

Description automatically generated

* **Trường hợp 2**: Khi người dùng nhập vào “2”: Chương trình sẽ hiển thị những dòng lệnh tiếp theo để người dùng nhập vào.

**Hình ảnh minh họa:**

A screenshot of a computer

Description automatically generated with medium confidence

* **Trường hợp 3**: Khi người dùng nhập vào những số khác “1” và “2”: Chương trình sẽ in ra màn hình dòng “Invalid choose: Please type again!” và sẽ bắt đầu cho người dùng nhập lại.

**Hình ảnh minh họa:**

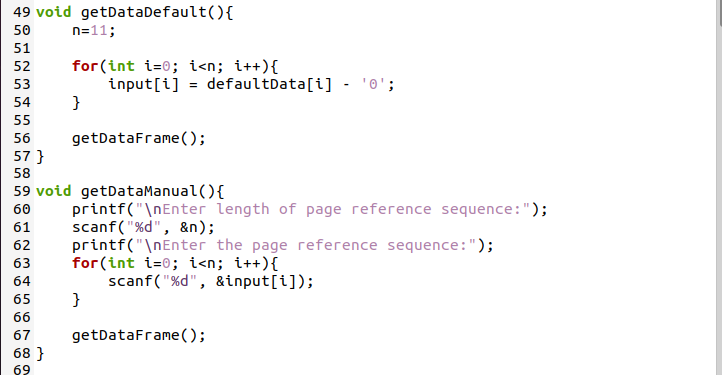
Text

Description automatically generated

Sau khi hoàn thành bước lựa chọn chương trình sẽ tiếp tục thực thi những hàm sau:

Code triển khai:

* Hình ảnh:



* Source code:

void getDataDefault(){

n=11;

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

input[i] = defaultData[i] - '0';

}

getDataFrame();

}

void getDataManual(){

printf("\nEnter length of page reference sequence:");

scanf("%d", &n);

printf("\nEnter the page reference sequence:");

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

scanf("%d", &input[i]);

}

getDataFrame();

}

Dựa vào input như ở trên:

* **Trường hợp 1**: Hàm getDataDefault() sẽ thực thi: Dùng dể biến chuỗi def mặc định mà mình đã gán sẵn vào mảng in[] để chúng ta thực hiện thuật toán
* **Trường hợp 2**: Hàm getData() sẽ thực thi: Dùng để nhập dãy số theo yêu cầu của người dùng
* Cuối mỗi hàm sẽ có getDataFrame() để thực hiện bước tiếp theo đó là nhập số frame cho thuật toán

\*Chọn số mặc định :

1.Default referenced sequence

2.Manual input sequence

\*Chọn dãy số theo yêu cầu:

2,0,5,2,1,9,8,5,0,0,7

\*Số khung trang là :

Enter no of frames: 3

A screenshot of a computer

Description automatically generated

1. **Triển khai khung trang:**

Code triển khai:

* Hình ảnh:

Text, letter

Description automatically generated

* Source code:

void getDataFrame(){

frameNum = 0;

while(frameNum < 1){

printf("\nEnter no of frames:");

scanf("%d", &frameNum);

}

getAlgorithm();

}

* Giải thích code:
  + Nhập vào số khung trang với biến nf: Nếu nf < 1, người dùng sẽ phải nhập lại cho đến khi nào đúng.
    - * Hàm getAlgorithm(): Để tiếp tục đến với giao diện chọn thuật toán
* Kết quả

A screenshot of a computer

Description automatically generated

1. **Giao diện chọn giải thuật thay thế trang:**

Code triển khai:

* Hình ảnh:

Text

Description automatically generated

* Source code:

void getAlgorithm(){

int choice = 0;

while(choice == 0){

printf("---Page Replacement algorithm---\n");

printf("1. FIFO algorithm\n");

printf("2. OPT algorithm\n");

printf("3. LRU algorithm\n");

scanf("%d", &choice);

switch(choice){

case 1:

FIFO();

break;

case 2:

OPT();

break;

case 3:

LRU();

break;

default:

printf("Invalid choose! Please type again.\n");

choice = 0;

break;

}

}

}

* Giải thích:

Trong hàm getAlgorithm(), ta sử dụng vòng lặp while để người dùng lựa chọn:

* **Trường hợp 1**: Khi người dùng nhập vào “1”: Chương trình sẽ thực thi thuật toán FIFO
* **Trường hợp 2**: Khi người dùng nhập vào “2”: Chương trình sẽ thực thi thuật toán OPT
* **Trường hợp 3**: Khi người dùng nhập vào “3”: Chương trình sẽ thực thi thuật toán LRU
* **Trường hợp 4:** Khi người dùng nhập những số khác “1”,”2”,”3”: Chương trình sẽ in ra màn hình dòng “Invalid choose: Please type again!” và sẽ thực hiện vòng lặp bắt đầu cho người dùng nhập lại.

Khi đã nhập xong nó sẽ chuyển đến với các hàm tương ứng, để thực hiện yêu cầu:

* **Thuật toán FIFO**

Ta sẽ dùng một mảng p[] như một mảng trung gian để thực hiện từng bước của quá trình thay bằng việc dùng thẳng mảng tg[][] để lưu trữ toàn bộ quá trình và xử lý trên đó.

Sau khi thao tác với mảng p[] xong, ta sẽ lưu nó vào biến tg[][] và kết thúc.

Code triển khai:

* Hình ảnh:

Graphical user interface, text, application

Description automatically generated

* Source code:

void FIFO(){

initialize();

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

if(isHit(input[i]) == 0){

int k;

for(k=0; k<frameNum - 1; k++){

p[k] = p[k+1];

}

p[k] = input[i];

for(int j=0; j<frameNum; j++){

tg[j][i] = p[j];

}

pf[i] = 1;

tg[frameNum][i] = 1;

faultCount++;

}

else{

pf[i] = 0;

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

}

}

}

printResult();

}

**Các hàm thành phần:**

* + Hàm initialize() (Hàm khởi tạo): Khởi tạo các giá trị ban đầu cho biến pgfaultcnt và p[]

Code triển khai:

* Hình ảnh:

A screenshot of a computer

Description automatically generated with medium confidence

* Source code:

void initialize(){

faultCount = 0;

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

p[i] = -1;

}

}

* + Hàm isHit() (Hàm kiểm tra): Kiểm tra xem biến in[i] đang xét hiện đã nằm trong hang chờ p[] hay chưa

Kết quả sẽ trả về “1” (khi xảy ra đụng độ) và “0” (khi không xảy ra đụng độ)

Code triển khai:

* Hình ảnh:

A screenshot of a computer

Description automatically generated

* Source code:

int isHit(int data){

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

if(p[i]==data){

hit = 1;

break;

}

}

return hit;

}

* + Nếu như biến in[i] ta thực hiện việc thay thế in[i] với trang đã tồn tại lâu nhất trong bộ nhớ, vì trình tự thêm vào lần lượt tại các vị trí [0], [1], [2] vậy nên mặc định rằng tại vị trí [0] sẽ luôn là vị trí mà trang tồn tại lâu nhất, ta đưa các vị trí cũ từ [2] về [1], [1] về [0] và thay số mới vào vị trí [2] (Ví dụ với frame = 3)
  + Sau khi thực hiện xong lưu quá trình hiện tại vào biến tg và tiếp tục xét
  + Cuối cùng là in ra kết quả
* Kết quả:

A screenshot of a computer

Description automatically generated

* **Thuật toán OPT**

Thuật toán OPT với các bước ban đầu sẽ tương tự như thuật toán FIFO, dựa trên cơ chế của thuật toán OPT, là **thay thế trang nhớ được tham chiếu trễ nhất trong tương lai**.

Dựa trên cơ chế đó, ta thêm vào biến near[] để xác định vị trí mà số sẽ có lần thay thế xa nhất để thay thế cho số hiện tại.

Đầu tiên, ta kiểm tra lần lượt với các biến hiện tại có trong p[] với các số trong tương lai sẽ xét. Sau đó xác định số tại vị trí mà sẽ thay thế xa nhất trong tương lai và thay thế nó với số hiện tại đang xét

Code triển khai:

* Hình ảnh:

:Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* Source code:

void OPT(){

initialize();

int near[50];

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

if(isHit(input[i]) == 0){

for(int j=0; j<frameNum; j++){

int pg = p[j];

int found = 0;

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

if(pg == input[k]){

near[j] = k;

found = 1;

break;

}

else{

found = 0;

}

}

if(found == 0){

near[j] = 9999;

}

}

int max = -9999;

int repIndex = 0;

for(int j=0; j<frameNum; j++){

if(near[j] > max){

max = near[j];

repIndex = j;

}

}

pf[i] = 1;

p[repIndex] = input[i];

faultCount++;

tg[frameNum][i] = 1;

}

else{

pf[i] = 0;

}

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

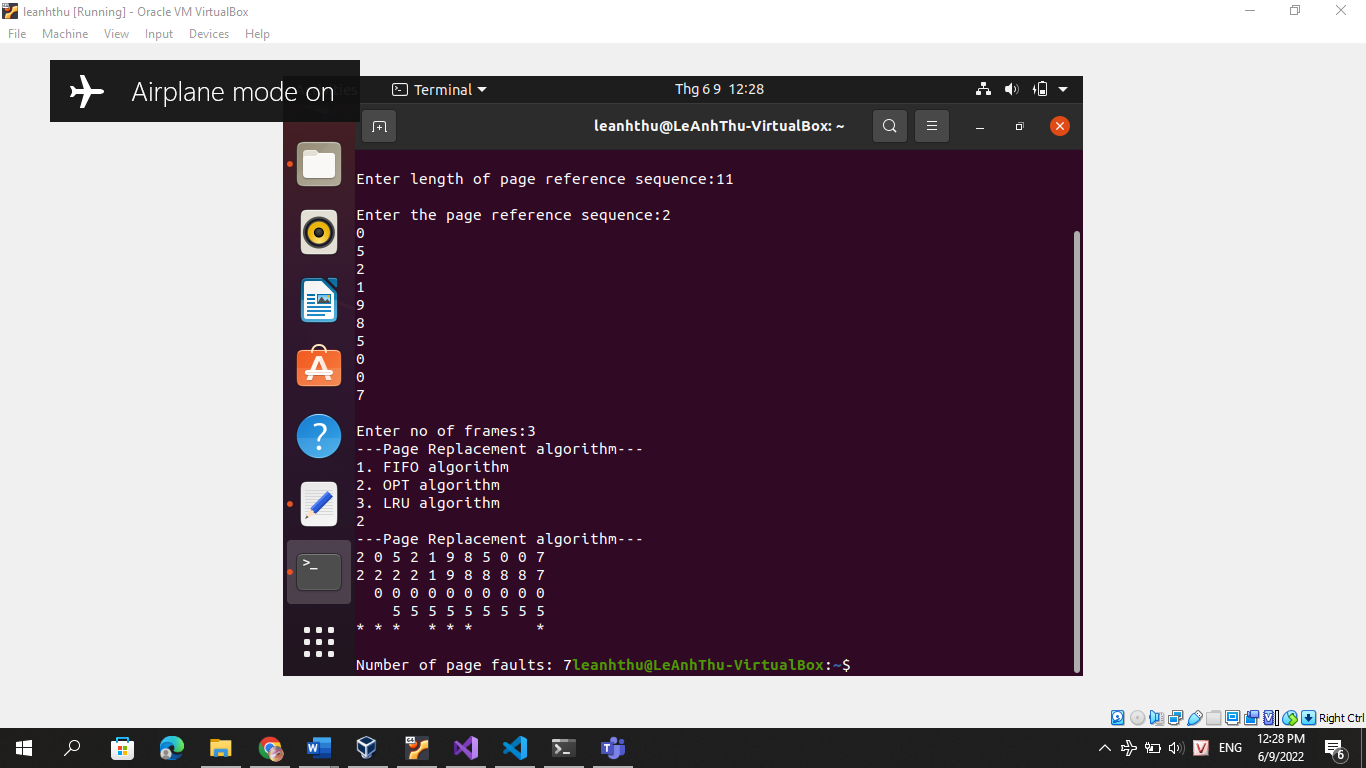
}

}

printResult();

}

* Kết quả:



* **Thuật toán LRU**

Thuật toán LRU cũng tương tự như thuật toán OPT, thuật toán LRU hoạt động dựa trên cơ chế thay thế trang nhớ được tham chiếu trễ nhất trong quá khứ. Vì vậy, theo cơ chế đã được định nghĩa biến near bây giờ sẽ thành biến least để chỉ trang nhớ được tham chiếu trễ nhất trong quá khứ.

Vì thuật toán LRU được xét trong quá khứ nên thay vì ta xét từ i cho đến n, thì giờ ta sẽ xét từ vị trí trở về 0.

Kết quả, ta sẽ tìm được trang nhớ tham chiếu trễ nhất và thay thế nó cho số hiện tại.

Code triển khai:

* Hình ảnh:

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* Source code:

void LRU(){

initialize();

int least[50];

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

if(isHit(input[i]) == 0){

for(int j=0; j<frameNum; j++){

int pg = p[j];

int found = 0;

for(int k=i-1; k>=0; k--){

if(pg == input[k]){

least[j] = k;

found = 1;

break;

}

else{

found = 0;

}

}

if(found == 0){

least[j] = -9999;

}

}

int min = 9999;

int repIndex = 0;

for(int j=0; j<frameNum; j++){

if(least[j] < min){

min = least[j];

repIndex = j;

}

}

pf[i] = 1;

p[repIndex] = input[i];

faultCount++;

tg[frameNum][i] = 1;

}

else{

pf[i] = 0;

}

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

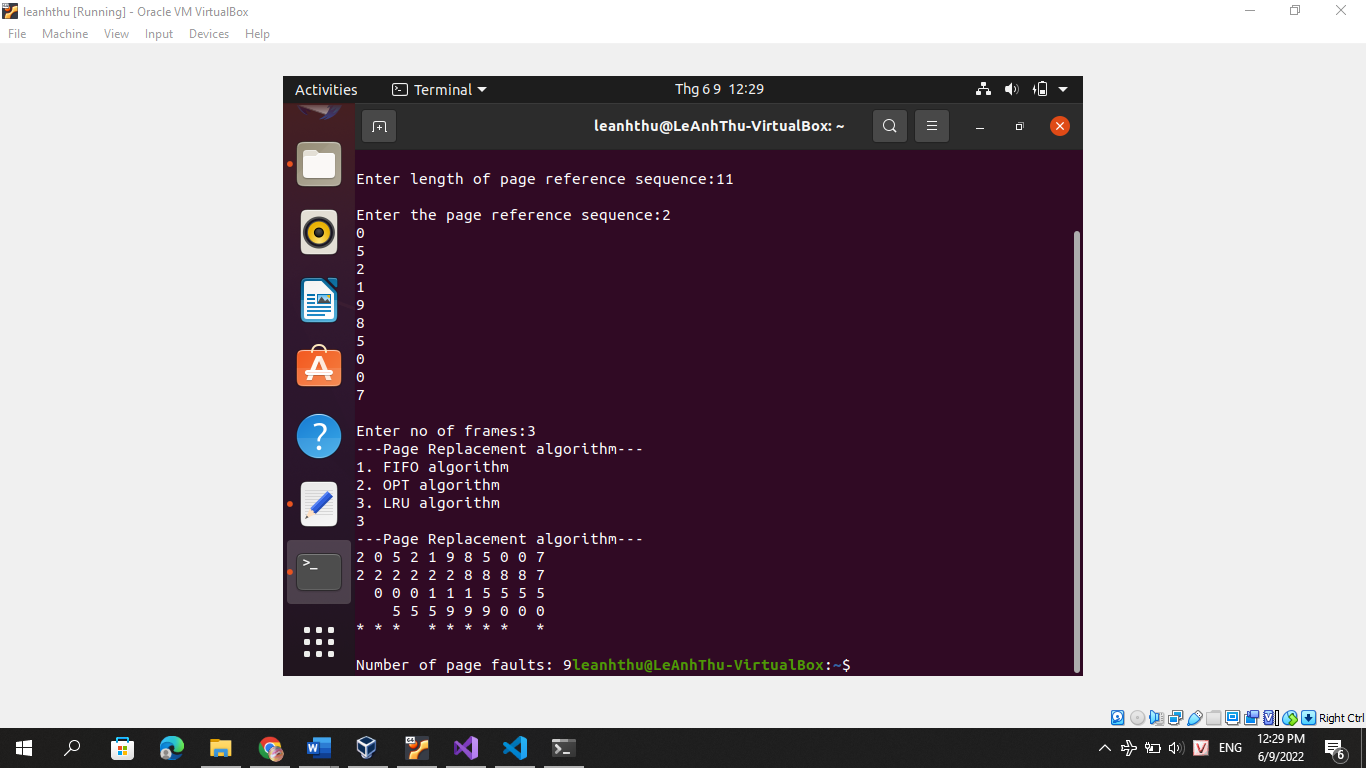
}

}

printResult();

}

* Kết quả:



**Source code đầy đủ:**

#include <stdio.h>

char defaultData[] = "123412512345";

int frameNum;

int n = 11;

int faultCount = 0;

int input[100];

int p[50];

int pf[100];

int tg[100][100];

void getDataDefault();

void getDataManual();

void getDataFrame();

void getAlgorithm();

void initialize();

void printResult();

int isHit(int data);

void FIFO();

void OPT();

void LRU();

int main(){

int choice = 0;

while(choice == 0){

printf("---Page Replacement Algorithm---\n");

printf("1. Default referenced sequence\n");

printf("2. Manual input sequence\n");

scanf("%d", &choice);

switch(choice){

case 1:

getDataDefault();

break;

case 2:

getDataManual();

break;

default:

printf("Invalid choose: Please type again!\n");

choice = 0;

break;

}

}

return 0;

}

void getDataDefault(){

n=11;

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

input[i] = defaultData[i] - '0';

}

getDataFrame();

}

void getDataManual(){

printf("\nEnter length of page reference sequence:");

scanf("%d", &n);

printf("\nEnter the page reference sequence:");

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

scanf("%d", &input[i]);

}

getDataFrame();

}

void getDataFrame(){

frameNum = 0;

while(frameNum < 1){

printf("\nEnter no of frames:");

scanf("%d", &frameNum);

}

getAlgorithm();

}

void getAlgorithm(){

int choice = 0;

while(choice == 0){

printf("---Page Replacement algorithm---\n");

printf("1. FIFO algorithm\n");

printf("2. OPT algorithm\n");

printf("3. LRU algorithm\n");

scanf("%d", &choice);

switch(choice){

case 1:

FIFO();

break;

case 2:

OPT();

break;

case 3:

LRU();

break;

default:

printf("Invalid choose! Please type again.\n");

choice = 0;

break;

}

}

}

void initialize(){

faultCount = 0;

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

p[i] = -1;

}

}

void printResult(){

printf("---Page Replacement algorithm---\n");

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

printf("%d ", input[i]);

}

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

printf("\n");

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

if(tg[i][k] == -1){

printf(" ");

}

else{

printf("%d ", tg[i][k]);

}

}

}

printf("\n");

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

if(pf[i] == 1){

printf("\* ");

}

else{

printf(" ");

}

}

printf("\n\nNumber of page faults: %d", faultCount);

}

int isHit(int data){

int hit = 0;

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

if(p[i]==data){

hit = 1;

break;

}

}

return hit;

}

void FIFO(){

initialize();

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

if(isHit(input[i]) == 0){

int k;

for(k=0; k<frameNum - 1; k++){

p[k] = p[k+1];

}

p[k] = input[i];

for(int j=0; j<frameNum; j++){

tg[j][i] = p[j];

}

pf[i] = 1;

tg[frameNum][i] = 1;

faultCount++;

}

else{

pf[i] = 0;

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

}

}

}

printResult();

}

void OPT(){

initialize();

int near[50];

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

if(isHit(input[i]) == 0){

for(int j=0; j<frameNum; j++){

int pg = p[j];

int found = 0;

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

if(pg == input[k]){

near[j] = k;

found = 1;

break;

}

else{

found = 0;

}

}

if(found == 0){

near[j] = 9999;

}

}

int max = -9999;

int repIndex = 0;

for(int j=0; j<frameNum; j++){

if(near[j] > max){

max = near[j];

repIndex = j;

}

}

pf[i] = 1;

p[repIndex] = input[i];

faultCount++;

tg[frameNum][i] = 1;

}

else{

pf[i] = 0;

}

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

}

}

printResult();

}

void LRU(){

initialize();

int least[50];

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

if(isHit(input[i]) == 0){

for(int j=0; j<frameNum; j++){

int pg = p[j];

int found = 0;

for(int k=i-1; k>=0; k--){

if(pg == input[k]){

least[j] = k;

found = 1;

break;

}

else{

found = 0;

}

}

if(found == 0){

least[j] = -9999;

}

}

int min = 9999;

int repIndex = 0;

for(int j=0; j<frameNum; j++){

if(least[j] < min){

min = least[j];

repIndex = j;

}

}

pf[i] = 1;

p[repIndex] = input[i];

faultCount++;

tg[frameNum][i] = 1;

}

else{

pf[i] = 0;

}

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

}

}

printResult();

}

**PHẦN** **I I: Bài tập ôn tập**

**1. Nghịch lý Belady là gì? Sử dụng chương trình đã viết trên để chứng minh nghịch lý này.**

**2. Nhận xét về mức độ hiệu quả và tính khả thi của các giải thuật FIFO, OPT, LRU.**

**❖ Giải thuật nào là bất khả thi nhất? Vì sao?**

**❖ Giải thuật nào là phức tạp nhất? Vì sao?**

**Bài làm**

**Câu 1: Nghịch lý Belady là gì? Sử dụng chương trình đã viết trên để chứng minh nghịch lý này.**

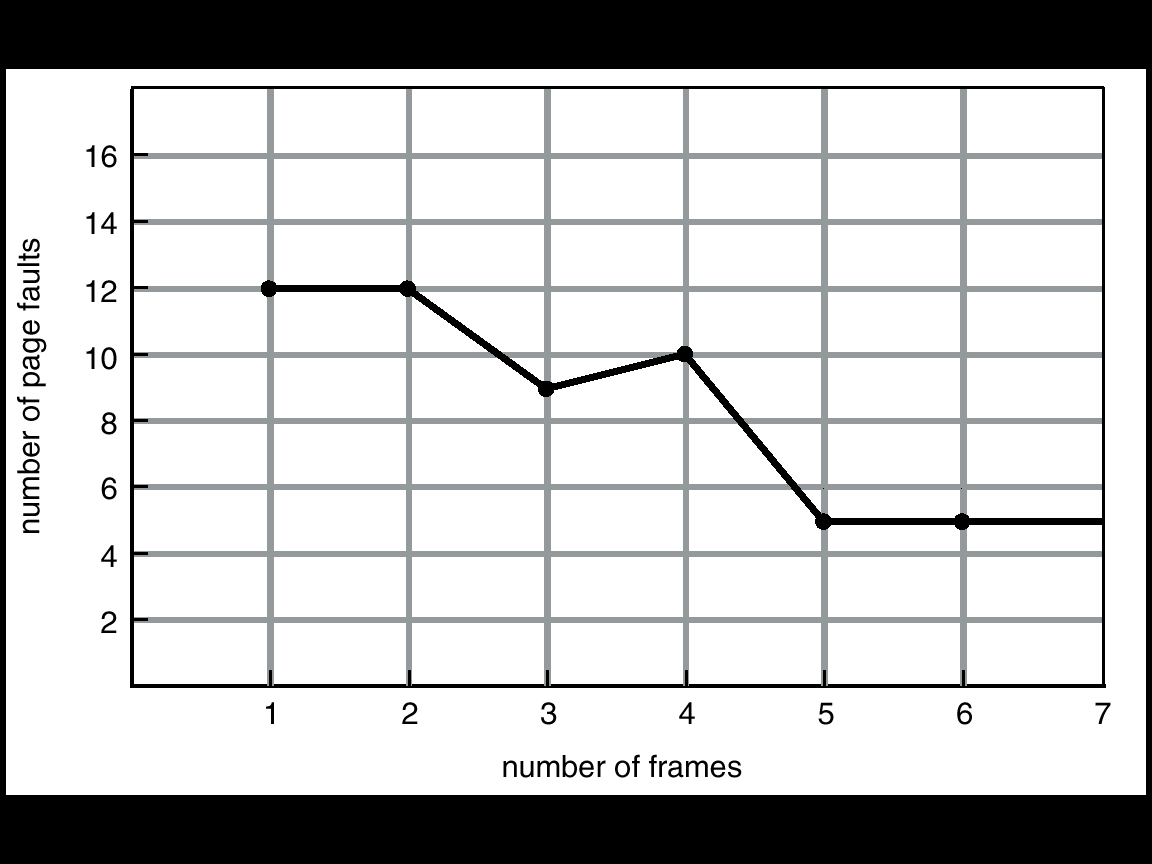
Thông thường, số frame tăng thì số page faults nên giảm. Tuy nhiên, với FIFO số frame tăng thì số page faults có thể cũng tăng theo. Điều nghịch lý này còn gọi là nghịch lý Belady.

Ví dụ chi tiết về nghịch lý Belady:



Chúng ta chú ý rằng số lượng lỗi cho 4 khung (10) là lớn hơn số lượng lỗi cho 3 khung (9). Hầu hết các kết quả không mong đợi này được gọi là sự nghịch lý Belady; đối với một số giải thuật thay thế trang, tỉ lệ lỗi trang có thể tăng khi số lượng khung được cấp phát tăng. Chúng ta sẽ mong muốn rằng cho nhiều bộ nhớ hơn tới một quá trình sẽ cải tiến năng lực của nó. Sự không bình thường này được gọi là nghịch lý Belady.

Sơ đồ minh họa:



Như sơ đồ trên, bất thường (anomaly) Belady khi số page fault tăng mặc dù quá trình đã được cấp nhiều frame hơn.

**Sử dụng chương trình đã viết trên để chứng minh nghịch lý này:**

Để chứng minh cho nghịch lý Belady, ta tái sử dụng ở phần FIFO thực hành ở phần trên để minh họa nên không trình bày lại phần triển khai mà chỉ thực hiện phần demo để kiểm chứng:

* **Thuật toán FIFO**

Code triển khai:

* Hình ảnh:

Graphical user interface, text, application

Description automatically generated

* Source code:

void FIFO(){

initialize();

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

if(isHit(input[i]) == 0){

int k;

for(k=0; k<frameNum - 1; k++){

p[k] = p[k+1];

}

p[k] = input[i];

for(int j=0; j<frameNum; j++){

tg[j][i] = p[j];

}

pf[i] = 1;

tg[frameNum][i] = 1;

faultCount++;

}

else{

pf[i] = 0;

for(int k=0; k<frameNum; k++){

tg[k][i] = p[k];

}

}

}

printResult();

}

* Kết quả:

Text

Description automatically generated

* Với frame = 3

Kết quả: 9 lỗi

Text

Description automatically generated

* Đối với frame = 4

Kết quả: 10 lỗi

Như đã chứng minh, mặc dù số lượng frame tăng theo logic sẽ giúp ta giảm được tình trạng lỗi trang nhưng trên thực tế, dựa vào kết quả nhận được số lượng lỗi trang tăng và đó là kết quả minh chứng cho nghịch lý Belady.

**Câu 2: Nhận xét về mức độ hiệu quả và tính khả thi của các giải thuật FIFO, OPT, LRU.**

**❖ Giải thuật nào là bất khả thi nhất? Vì sao?**

Giải thuật bất khả thi nhất là OPT. Vì giải thuật OPT dựa trên cơ sở lý thuyết là thay thế trang nhớ được tham chiếu trễ nhất trong tương lai. Trong thực tế, thì chúng ta không thể biết rằng trang nào sẽ được kích hoạt trong tương lai, điều này dẫn đến việc không thể so sánh giữa các trang. Điều này dẫn đến kết quả là giải thuật OPT trờ thành giải thuật bất khả thi nhất.

**❖ Giải thuật nào là phức tạp nhất? Vì sao?**

Nhìn chung LRU sẽ được coi là phức tạp hơn bởi lẽ OPT không thể cài đặt được trong thực tế. Nhưng nếu so trên thực nghiệm và cơ sở lý thuyết thì độ phức tạp của cả 2 giải thuật là như nhau.

Giải thuật phức tạp nhất là 2 giải thuật: OPT và LRU, cả 2 giải thuật này đều tương tự nhau bởi vì dựa trên nguyên lý, OPT là thay thế trang nhớ được tham chiếu trễ nhất trong tương lai, LRU là thay thế trang nhớ được tham chiếu trễ nhất trong quá khứ. Cả 2 giải thuật phải đối chiếu trên vòng lặp rất nhiều lần và độ phức tạp lớn hơn FIFO rất nhiều.

***Cảm ơn thầy đã hướng dẫn thực hành chúng em !***