

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**







**HỆ ĐIỀU HÀNH**

**LỚP: IT007.O18.1**

**BÁO CÁO BÀI TẬP THỰC HÀNH 6**

**Họ tên: Trần Đình Khánh Đăng**

**MSSV: 22520195**



**Phần 6.3**

1. **Task name 1: Giải thuật FIFO**

**Lưu đồ thuật toán**

Ảnh có chứa văn bản, biểu đồ, Phông chữ, ảnh chụp màn hình

Mô tả được tạo tự động

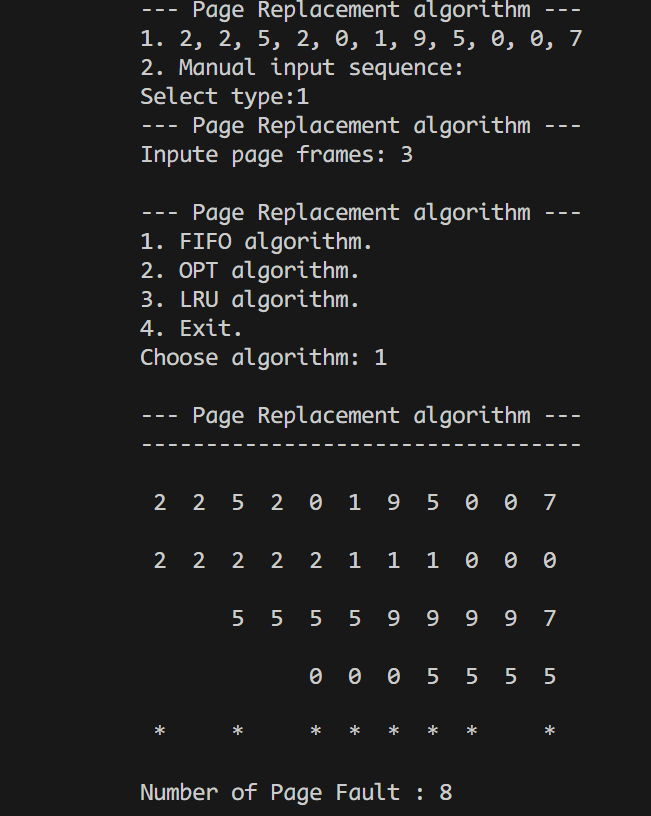
Lưu đồ thuật toán FIFO

**Sơ đồ trình bày**

Xét chuỗi truy xuất bộ nhớ sau: 2, 2, 5, 2, 0, 1, 9, 5, 0, 0, 7. Có bao nhiêu lỗi trang xảy ra khi sử dụng thuật toán thay thế FIFO, giả sử có 3 khung trang?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 5 | 2 | 0 | 1 | 9 | 5 | 0 | 0 | 7 |
| 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | 0 |
| - | - | 5 | 5 | 5 | 5 | 9 | 9 | 9 | 9 | 7 |
| - | - | - | - | 0 | 0 | 0 | 5 | 5 | 5 | 5 |
| \* |  | \* |  | \* | \* | \* | \* | \* |  | \* |

**Kết quả khi chạy chương trình:**

****

1. **Task name 2: Giải thuật LRU**

**Lưu đồ thuật toán**

Ảnh có chứa văn bản, biểu đồ, Kế hoạch, Bản vẽ kỹ thuật

Mô tả được tạo tự động

Lưu đồ hàm tìm ra trang được gói sớm nhất trong quá khứ

Ảnh có chứa văn bản, biểu đồ, Kế hoạch, Phông chữ

Mô tả được tạo tự động

Lưu đồ thuật toán LRU

**Sơ đồ trình bày**

Xét chuỗi truy xuất bộ nhớ sau: 2, 2, 5, 2, 0, 1, 9, 5, 0, 0, 7. Có bao nhiêu lỗi trang xảy ra khi sử dụng thuật toán thay thế LRU, giả sử có 3 khung trang?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 5 | 2 | 0 | 1 | 9 | 5 | 0 | 0 | 7 |
| 2 | 2 | 2 | 2 | 2 | 2 | 9 | 9 | 9 | 9 | 7 |
| - | - | 5 | 5 | 5 | 1 | 1 | 1 | 0 | 0 | 0 |
| - | - | - | - | 0 | 0 | 0 | 5 | 5 | 5 | 5 |
| \* |  | \* |  | \* | \* | \* | \* | \* |  | \* |

**Kết quả khi chạy chương trình:**

**Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, thiết kế

Mô tả được tạo tự động**

1. **Task name 3: Giải thuật OPT**

**Lưu đồ thuật toán**

Ảnh có chứa văn bản, biểu đồ, Bản vẽ kỹ thuật, Kế hoạch

Mô tả được tạo tự động

Lưu đồ hàm tìm ra vị trí cần được thay thế

Ảnh có chứa văn bản, biểu đồ, Phông chữ, Song song

Mô tả được tạo tự động

Lưu đồ thuật toán OPT

**Sơ đồ trình bày**

Xét chuỗi truy xuất bộ nhớ sau: 2, 2, 5, 2, 0, 1, 9, 5, 0, 0, 7. Có bao nhiêu lỗi trang xảy ra khi sử dụng thuật toán thay thế OPT, giả sử có 3 khung trang?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 5 | 2 | 0 | 1 | 9 | 5 | 0 | 0 | 7 |
| 2 | 2 | 2 | 2 | 2 | 1 | 9 | 9 | 9 | 9 | 7 |
| - | - | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| - | - | - | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| \* |  | \* |  | \* | \* | \* |  |  |  | \* |

**Kết quả khi chạy chương trình:**

**Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, thiết kế

Mô tả được tạo tự động**

**Chương trình:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

void Print(int \*seq, int \*\*table, char\* fault, int n,int f)

{

    int count = 0;

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

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

    for (int i = 0; i < n; i++)

        printf("%2d ", seq[i]);

    printf("\n\n\t");

    for (int i = 0; i < f; i++)

    {

        for (int j = 0; j < n; j++)

            if (table[i][j] != -1)

                printf("%2d ", table[i][j]);

            else

                printf(" ");

        printf("\n\n\t");

    }

    for (int i = 0; i < n; i++)

    {

        printf(" %c ", fault[i]);

        if (fault[i] == '\*')

            count++;

    }

    printf("\n\n\tNumber of Page Fault : %d\n\n", count);

}

int in\_MMR(int\*\* table, int f, int i, int page)

{

    for (int j = 0; j < f; j++)

        if (table[j][i] == page || table[j][i]==-1)

            return j;

    return -1;

}

void FIFO(int\* seq, int \*\*table, char \*fault, int n, int f)

{

    int first = 0;

    table[0][0] = seq[0];

    fault[0] = '\*';

    for (int i = 1; i < n; i++)

    {

        for (int j = 0; j < f; j++)

            table[j][i] = table[j][i - 1];

        int id = in\_MMR(table, f, i, seq[i]);

        if (id != -1)

        {

            if (table[id][i] == -1) {

                table[id][i] = seq[i];

                fault[i] = '\*';

            } else fault[i] = ' ';

        }

        else

        {

            fault[i] = '\*';

            table[first][i] = seq[i];

            first = (first + 1) % f;

        }

    }

}

void OPT(int\* seq, int\*\* table, char\* fault, int n, int f)

{

    int\* next,k;

    next = (int\*)malloc(f \* sizeof(int));

    table[0][0] = seq[0];

    fault[0] = '\*';

    for (k = 1; k < n; k++)

        if (seq[k] == seq[0])

            break;

    next[0] = k;

    for (int i = 1; i < n; i++)

    {

        int j;

        for (j = 0; j < f; j++)

            table[j][i] = table[j][i - 1];

        int id = in\_MMR(table, f, i, seq[i]);

        if (id == -1)

        {

            int choose = 0;

            for (j = 1; j < f; j++)

                if (next[choose] < next[j])

                    choose = j;

            table[choose][i] = seq[i];

            for (j = i + 1; j < n; j++)

                if (seq[j] == seq[i])

                    break;

            next[choose] = j;

            fault[i] = '\*';

        }

        else

        {

            if (table[id][i] == -1) {

                table[id][i] = seq[i];

                fault[i] = '\*';

            }

            else fault[i] = ' ';

            j = i + 1;

            for (j = i + 1; j < n; j++)

                if (seq[j] == seq[i])

                    break;

            next[id] = j;

        }

    }

}

void LRU(int\* seq, int\*\* table, char\* fault, int n, int f)

{

    int\* last;

    last = (int\*)malloc(f \* sizeof(int));

    table[0][0] = seq[0];

    fault[0] = '\*';

    last[0] = 0;

    for (int i = 1; i < n; i++)

    {

        for (int j = 0; j < f; j++)

            table[j][i] = table[j][i - 1];

        int id = in\_MMR(table, f, i, seq[i]);

        if (id == -1)

        {

            fault[i] = '\*';

            int min\_ref = 0;

            for (int j = 1; j < f; j++)

                if (last[j] < last[min\_ref])

                    min\_ref = j;

            last[min\_ref] = i;

            table[min\_ref][i] = seq[i];

        }

        else

        {

            if (table[id][i] == -1) {

                table[id][i] = seq[i];

                fault[i] = '\*';

            } else fault[i] = ' ';

            last[id] = i;

        }

    }

}

int main()

{

    int\* DRS, \*MIS, n, i, f, algo, type;

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

    printf("\t1. 2, 2, 5, 2, 0, 1, 9, 5, 0, 0, 7\n");

    printf("\t2. Manual input sequence: \t\n");

    printf("\tSelect type:");

    scanf("%d", &type);

    if (type == 2) {

        printf("\tInput length of sequences: ");

        scanf("%d", &n);

        MIS = (int\*)malloc(n \* sizeof(int));

        printf("\tInpute sequences: ");

        for (i = 0; i < n; i++)

            scanf("%d", &MIS[i]);

    }

    else {

        n = 11;

        MIS = (int\*)malloc(n \* sizeof(int));

        MIS[0] = 2;

        MIS[1] = 2;

        MIS[2] = 5;

        MIS[3] = 2;

        MIS[4] = 0;

        MIS[5] = 1;

        MIS[6] = 9;

        MIS[7] = 5;

        MIS[8] = 0;

        MIS[9] = 0;

        MIS[10] = 7;

    }

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

    printf("\tInpute page frames: ");

    scanf("%d", &f);

    char\* fault = (char\*)malloc((n + 1) \* sizeof(char));

    int\*\* table = (int\*\*)malloc(f \* sizeof(int\*));

    for (i = 0; i < f; i++) {

        table[i] = (int\*)malloc(n \* sizeof(int));

        for (int j = 0; j < n; j++)

            table[i][j] = -1;

    }

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

    printf("\t1. FIFO algorithm.\n");

    printf("\t2. OPT algorithm.\n");

    printf("\t3. LRU algorithm.\n");

    printf("\t4. Exit.\n");

    printf("\tChoose algorithm: ");

    scanf("%d", &algo);

    switch (algo)

    {

    case 1:

        FIFO(MIS, table, fault, n, f);

        break;

    case 2:

        OPT(MIS, table, fault, n, f);

        break;

    case 3:

        LRU(MIS, table, fault, n, f);

        break;

    default:

        return 0;

    }

    Print(MIS, table, fault, n, f);

    return 0;

}

**Phần 6.4**

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

* **Nghịch lý Belady:** là hiện tượng tăng số lượng frame nhưng số lỗi tăng chứ không giảm.
* **Ví dụ:**
  + n = 12.
  + MIS[] = [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]
  + Thuật toán FIFO.

3 trang có 9 lỗi 4 trang có 10 lỗi

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, thiết kế

Mô tả được tạo tự độngẢnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, thiết kế

Mô tả được tạo tự động

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

* **Hiệu quả:**
* LRU thường được xem là giải thuật hiệu quả với hiệu suất tốt và sự gần gũi với thực tế.
* OPT có hiệu suất tối ưu nhất trong lý thuyết nhưng không thực tế.
* FIFO có thể đơn giản và dễ triển khai, nhưng không hiệu quả trong môi trường truy cập bộ nhớ đa dạng.
* **Tính khả thi:**
* LRU thường được sử dụng rộng rãi trong các hệ thống thực tế nhờ tính khả thi và hiệu suất tốt.
* FIFO thích hợp cho các hệ thống đơn giản hoặc khi không có yêu cầu hiệu suất cao.
* OPT không khả thi do yêu cầu thông tin về tương lai mà thường không có sẵn.