实验报告三 内存管理

一、 实验目的

- 理解内存页面调度的机理。
- 掌握几种理论页面值换算法的实现方法
- 通过实验比较各种调度算法的优劣。

二、 相关知识

- 指针、结构体。
- 操作系统相关内存交换知识。
- 页面算法是虚拟内存管理实现的关键，通过本次实验理解内存页面调度的机制，在模拟实现 FIFO、LRU、OPT 算法的基础上，比较各种置换算法的效率和优劣，从而了解虚拟储存的实现过程。

三、 实验内容

随机给出一个页面执行序列，如：1,5,3,4,2,1,3,4,5,7,9,……。要求计算以下几种置换算法的缺页数、缺页率和命中率。

- 最佳置换算法 OPT（Optimal）
- 先进先出算法 FIFO（First In First Out）
- 最近最少使用算法 LRU（Least Recently Used）

四、 实验环境

- PC + Linux Red Hat 操作系统 + GCC
- 或 Windows xp + VC

五、 实验中遇到的主要问题及其解决方式

- 如何存储在内存中的页面

  开辟一个固定长度的数组，用于存储在内存中的页面。

- 如何替换页表中的页面

  FIFO 算法：用一个指针指向下一个放置页面的内存地址，依次移动指针，当指针达到数组尾部后，重新指向数组开头。

  LRU 算法：往远处查看历史执行页面，找到最久没被使用的页面，然后替换。

  OPT 算法：往远处查看即将被执行的页面，找到最久最不可能被使用的页面，然后替换。

- 实验结果

```
cjx@cjx:~/桌面/lab3$ gcc main.c FIFO.c OPT.c LRU.c tools.c -o main
cjx@cjx:~/桌面/lab3$ ./main
随机产生页面执行序列:
1 3 1 7 9 1 9 2 1 7 5 6 9 1 7 1 1 1 4 2 1 7 9 2 6 4 3 9 9 9 1 9 2 1 4 9 2 1 1 2
7 4 5 5 4 2 5 4 2 9

页面执行序列长度: 50
内存分配大小: 5

------
OPT
缺页号: -1--3--7--9--2--5--6--4--6--3--1--5-
缺页数: 12
缺页率: 0.240000
命中率:  0.760000

------
FIFO
缺页号: -1--3--7--9--2--5--6--1--7--4--2--9--6--3--1--4--2--7--5--9-
缺页数: 20
缺页率: 0.400000
命中率:  0.600000

------
LRU
缺页号: -1--3--7--9--2--5--6--9--4--2--6--4--3--1--2--7--5--9-
缺页数: 18
缺页率: 0.360000
命中率:  0.640000
```

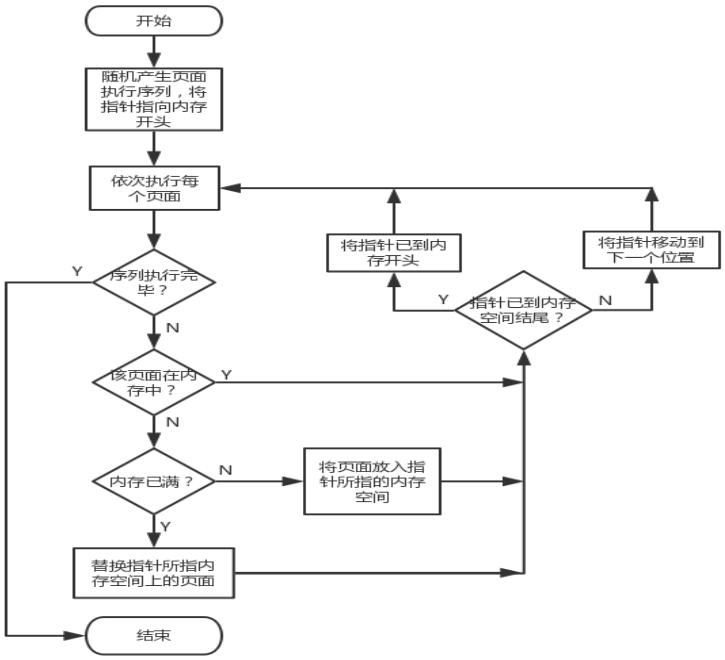
六、 源代码和流程图

- **main 函数**

```
#include<stdio.h>
#include "exchange.h"
#include "tools.h"
int main()
{
    int ResidentPages[RPN];
    int pages[PAGESNUM];

    randomPages(pages, PAGESNUM);//随机产生页面执行序列

    printf("随机产生页面执行序列：\n");
    print(pages, PAGESNUM);//打印页面执行序列
    printf("\n 页面执行序列长度：%d\n 内存分配大小：%d\n",PAGESNUM,RPN);

    printf("\n------\nOPT\n");
    init(ResidentPages, RPN);
    OPT(ResidentPages, RPN, pages, PAGESNUM);//OPT 算法

    printf("\n------\nFIFO\n");
    init(ResidentPages, RPN);
    FIFO(ResidentPages, RPN, pages, PAGESNUM);//FIFO 算法

    printf("\n------\nLRU\n");
    init(ResidentPages, RPN);
    LRU(ResidentPages, RPN, pages, PAGESNUM);//LRU 算法
    return 0;
}
```

✧ **FIFO 算法**

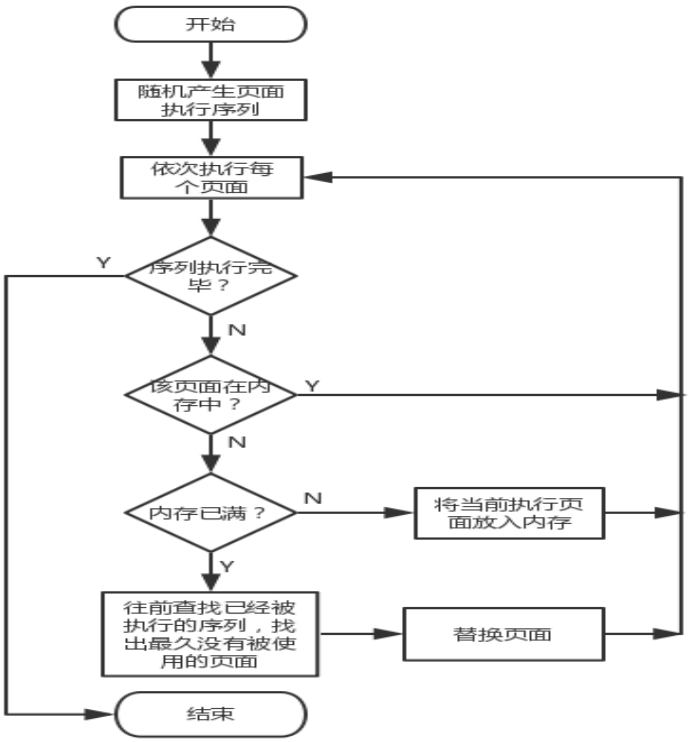
```
void FIFO(int residentPages[], int RPN1, int pages[], int PAGESNUM1)//先进先出算法
{
    int errNum = 0;
    int p = 0;//堆栈指针
    printf("缺页号：");
    for (int i = 0; i<PAGESNUM1; i++) {
        if (iInResidentPages(residentPages, RPN1, pages[i], &errNum) == -1) {
            printf("-%d-", pages[i]);
            residentPages[p] = pages[i];
            p++;
            if (p == RPN) {//指针到堆栈尾巴后又回到开头
                p = 0;
            }
        }
    }
    testError(errNum, PAGESNUM);//计算缺页数、缺页率和命中率
}
```



✧ **LRU 算法**

```
#include<stdio.h>
#include "tools.h"

void LRU(int residentPages[], int RPN1, int pages[], int PAGESNUM1)//最近最少使用
{
    int errNum = 0;
    int willUse[RPN];
    int pn;

    printf("缺页号：");
    for (int i = 0; i<PAGESNUM1; i++) {
        if (iInResidentPages(residentPages, RPN1, pages[i], &errNum) == -1) {
            printf("-%d-", pages[i]);
            init(willUse, RPN1);

            pn = RPN1;
            //往远查看历史执行页面，找出内存中最久最不可能使用的页面
            for (int j = i - 1; j >= 0; j--) {
                if (pn == 1) {
                    break;
                }
                for (int k = 0; k<RPN1; k++) {
                    if (residentPages[k] == pages[j]) {
                        if (willUse[k] == 0) {
                            willUse[k] = 1;
                            pn--;
                            break;
                        }
```

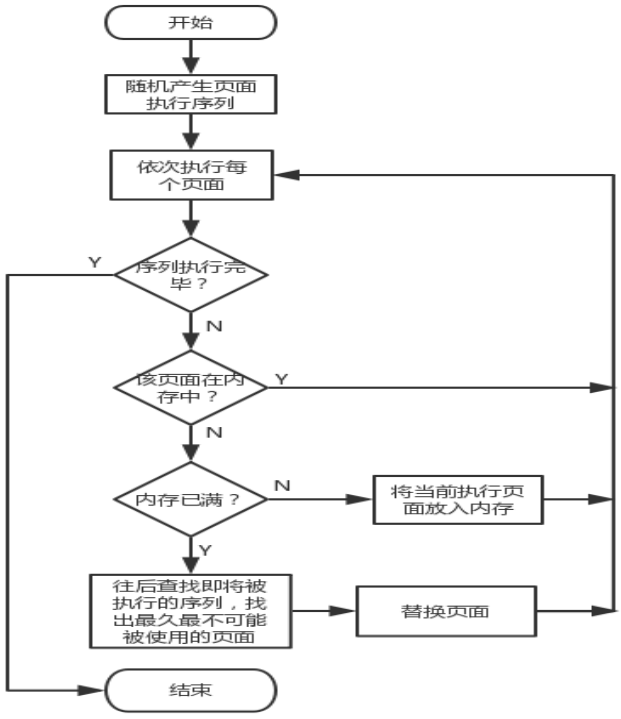
```
                    }
                }
            }

            //替换
            for (int t = 0; t<RPN1; t++) {
                if (willUse[t] == 0) {
                    residentPages[t] = pages[i];
                    break;
                }
            }
        }
    }

    testError(errNum, PAGESNUM);//计算缺页数、缺页率和命中率
}
```



✧ **OPT 算法**

```
#include<stdio.h>
#include "tools.h"

void OPT(int residentPages[], int RPN1, int pages[], int PAGESNUM1)//最佳置换算法
{
    int errNum = 0;
    int willUse[RPN];
    int pn;

    printf("缺页号：");
    for (int i = 0; i<PAGESNUM1; i++) {
        if (iInResidentPages(residentPages, RPN, pages[i], &errNum) == -1) {

            printf("-%d-", pages[i]);


            for (int j = 0; j<RPN1; j++) {
                willUse[j] = 0;
            }

            pn = RPN1;

            //往远查看即将被执行的页面，找出最久最不可能被使用到的页面
            for (int j = i + 1; j<PAGESNUM1; j++) {
                if (pn == 1) {
                    break;
                }
                for (int k = 0; k<RPN1; k++) {
                    if (residentPages[k] == pages[j]) {
```

```
                        willUse[k] = 1;
                        pn--;
                    }
                }
            }

            //替换
            for (int t = 0; t<RPN1; t++) {
                if (willUse[t] == 0) {
                    residentPages[t] = pages[i];
                    break;
                }
            }
        }
    }
    testError(errNum, PAGESNUM); //计算缺页数、缺页率和命中率
}
```



七、 实验总结

通过这次实验，对各种置换算法有了更加深入的理解。每种算法都有各自的优缺点。OPT 算法虽然命中率最高，但是在实际应用中无法实现，因为实际应用中页面的执行无法预知。