



# 《操作系统》课程 实验报告

实验名称：实验四 页面淘汰算法模拟实现与比较

姓名：江家玮

学号：22281188

日期：2024.11.09

# 目录

1 实验目的 .....	1
2 实验内容 .....	1
3 实验代码分析 .....	1
3.1 代码主要组成部分 .....	1
3.2 代码详解 .....	1
3.2.1 包含的头文件 .....	1
3.2.2 定义常量和函数声明 .....	2
3.2.3 数据结构 .....	2
3.2.4 初始化函数 .....	3
3.2.5 随机生成访问序列 .....	3
3.2.6 显示当前状态 .....	4
3.2.7 页面存在性检查 .....	5
3.2.8 页面替换算法 .....	5
4 实验结果展示 .....	12
5 实验心得体会 .....	14

## 1 实验目的

理解并掌握主要页面淘汰算法的设计和实现要旨。

## 2 实验内容

利用标准 C 语言，编程设计与实现最佳淘汰算法、先进先出淘汰算法、最近最久未使用淘汰算法、简单 Clock 淘汰算法及改进型 Clock 淘汰算法，并随机发生页面访问序列开展有关算法的测试及性能比较。

## 3 实验代码分析

### 3.1 代码主要组成部分

该代码是一个页面置换算法的模拟程序，主要用于测试和比较不同的页面替换策略。代码包括以下主要组成部分：

- (1) 页面信息结构 (PAGE) 定义：保存页面访问序列和分配的最大页框数。
- (2) 内存页面结构 (MEM) 定义：用于管理每个页面的访问记录、访问位和修改位。
- (3) 页面初始化：设置页面的初始状态，包括最大页框数和访问序列长度。
- (4) 访问序列生成：随机生成页面访问序列，以供后续测试使用。
- (5) 页面替换算法实现：实现了多种页面替换算法，包括：
  - 最佳替换 (OPT)
  - 先进先出 (FIFO)
  - 最近最久未使用 (LRU)
  - 时钟 (CLOCK)，包括简单和改进版本。
- (6) 缺页率和运行时间计算：用于统计每种算法的缺页率和运行时间。
- (7) 主函数：循环执行不同的页面替换算法，输出各算法的平均缺页率和执行时间。

### 3.2 代码详解

这段代码实现了一种模拟的页面置换算法，包括多种替换策略（如 FIFO、OPT、LRU 和 Clock）。下面是代码的详细分析：

#### 3.2.1 包含的头文件

```
1. #include "windows.h"
2. #include <conio.h>
3. #include <stdlib.h>
4. #include <io.h>
5. #include <string.h>
6. #include <stdio.h>
7. #include <iostream>
8. #include <time.h>
9. using namespace std;
10. #define MAX 64
```

- windows.h: Windows 特有的 API。
- conio.h: 提供控制台输入输出函数。
- stdlib.h: 标准库，包含内存管理、随机数生成等函数。
- io.h: 提供文件处理功能。

- string.h: 字符串处理函数。
- stdio.h: 输入输出函数。
- iostream: C++标准输入输出流。
- time.h: 提供时间处理函数。

### 3.2.2 定义常量和函数声明

```

1. void FIFO();           // 先进先出算法
2. void OPT();           // 最佳置换算法
3. void LRU();           // 最近最久未使用算法
4. void CLOCK_pro(int choose); // 改进型Clock 置换算法
5. void Init();          // 初始化相关数据结构
6. void getRandomList(int m, int e, int s); // 随机生成访问序列
7. void showState();     // 显示当前状态及缺页情况
8. int isExist(int page); // 查找页面是否在内存
9. bool randBool();      // 随机生成0或1

```

- MAX: 定义页面的最大数量为64。
- 函数声明用于后续实现，分别对应不同的页面置换算法和辅助功能。

### 3.2.3 数据结构

```

1. struct PageInfo      // 页面信息结构
2. {
3.     int pages[MAX];   // 模拟的最大访问页面数
4.     int isVisited;    // 标志位, 0 表示无页面访问数据
5.     int page_missing_num; // 缺页中断次数
6.     int allocated_page_num; // 分配的页框数
7.     int visit_list_length; // 访问页面序列长度
8. } pInfo;
9.
10. struct MemInfo // 页表项信息
11. {
12.     int time; // 记录页框中数据的访问历史
13.     int isVisit; // 访问位
14.     int isModify; // 修改位
15.     int pages; // 页号, 16 位信息, 前6 位代表页号, 后10 位为偏移地址 0~2^16
16. } mInfo;
17.
18. MemInfo pageList[MAX]; // 分配的页框
19.
20. int page_loss_num = 0; // 缺页次数
21. int current_page;      // 页面访问指针
22. int replace_page;      // 页面替换指针
23. int is_loss_page;      // 缺页标志, 0 为不缺页, 1 为缺页

```

- PageInfo: 用于存储页面信息, 包括访问序列、缺页数、分配的页框数等。
- MemInfo: 表示页表项信息, 包括访问时间、访问位、修改位和页号。
- pageList: 数组, 用于存储当前在内存中的页面信息。

### 3.2.4 初始化函数

```
1. void Init()
2. {
3.     int i, pn;
4.     is_loss_page = 0;           // 缺页标志, 0 为不缺页, 1 为缺页
5.     pInfo.page_missing_num = 0; // 缺页次数
6.     pInfo.isVisited = 0;        // 标志位, 0 表示无页面访问数据
7.     printf("请输入分配的页框数: "); // 自定义分配的页框数
8.     scanf("%d", &pn);
9.     pInfo.allocated_page_num = pn;
10.    for (i = 0; i < MAX; i++) // 清空页面序列
11.    {
12.        pInfo.pages[i] = -1;
13.    }
14. }
```

- 功能: 初始化页面信息, 设定分配的页框数, 并清空页面序列。
- 用户输入: 通过 scanf 函数让用户输入分配的页框数 (pn), 并将其保存到 pInfo.allocated\_page\_num。
- 数组初始化: 使用-1 标记未分配的页面。

### 3.2.5 随机生成访问序列

```
1. void getRandomList(int m, int e, int s)
2. {
3.     int pl;
4.     Init(); // 初始化
5.     printf("请输入访问序列的长度: ");
6.     scanf("%d", &pl); // 随机生成访问序列的长度
7.     pInfo.visit_list_length = pl;
8.     srand((unsigned)time(NULL)); // 随机种子
9.     int p = rand() % MAX; // 在 0—MAX 范围内随机初始化工作集的起始位置 p
10.    int i, j=0;
11.    double t= rand() % 10 / 10.0; // 一个范围在 0 和 1 之间的值 t
12.    for (i = 0; i < s; i++) // 重复 s 次
13.    {
14.        if (j > pInfo.visit_list_length) // 不能超过访问序列的长度
15.            break;
16.        for (j = i * m; j < (i + 1) * m; j++) // 生成 m 个取值范围在 p~p + e 间的随机数
17.        {
18.            pInfo.pages[j]= (p + (rand() % e)) % MAX; // 并记录到页面访问序列串中;
19.        }
20.        double r = (rand() % 10) / 10.0; // 生成一个范围在 0-1 之间的随机数 r
21.        if (r < t) // 如果 r < t, 则为 p 生成一个新值
22.            p = rand() % MAX;
23.        else
24.        {
```

```

25.         p = (p + 1) % MAX; // 否则向后循环移位
26.     }
27. }
28. }

```

- 功能: 生成随机的页面访问序列。
- 参数:
  - m: 每次生成的页数。
  - e: 生成的页面范围。
  - s: 生成的重复次数。
- 随机数生成: 通过 rand()函数生成随机的页面编号, 确保页面在 0 到 MAX-1 之间。

### 3.2.6 显示当前状态

```

1. void showState(void)
2. {
3.     int i, n;
4.     if (current_page == 0)
5.     {
6.         printf("\n~~~~~页面访问序列~~~~~\n");
7.         for (i = 0; i < pInfo.visit_list_length; i++)
8.         {
9.             printf("%4d", pInfo.pages[i]);
10.            if ((i + 1) % 10 == 0) printf("\n"); // 每行显示10 个
11.        }
12.        printf("~~~~~\n");
13.    }
14.    printf("访问%3d -->内存空间[", pInfo.pages[current_page]);
15.    for (n = 0; n < pInfo.allocated_page_num; n++) // 页框信息
16.    {
17.        if (pageList[n].pages >= 0)
18.            printf("%3d", pageList[n].pages);
19.        else
20.            printf(" ");
21.    }
22.    printf(" ]");
23.    if (is_loss_page == 1) // 缺页标志, 0 为不缺页, 1 为缺页
24.    {
25.        printf(" --> 缺页中断 --> ");
26.        if (current_page == 0)
27.        {
28.            printf("置换率 = %3.1f ", (float)(pInfo.page_missing_num) * 100.00 );
29.        }
30.        else{
31.            printf("置换率 = %3.1f", (float)(pInfo.page_missing_num) * 100.00 / current_page);
32.        }
33.    }

```

```

34.     printf("\n");
35. }

```

- 功能: 显示当前访问的页面和内存中存储的页面。
- 输出: 显示当前访问的页面编号和内存中所有页面的状态, 以便观察替换效果。

### 3.2.7 页面存在性检查

```

1.  int isExist(int page)
2.  {
3.      int n;
4.      for (n = 0; n < pInfo.allocated_page_num; n++)
5.      {
6.          pageList[n].time++; // 访问历史加1
7.      }
8.      for (n = 0; n < pInfo.allocated_page_num; n++)
9.      {
10.         if (pageList[n].pages == page)
11.         {
12.             is_loss_page = 0; //is_loss_page 缺页标志, 0 为不缺页, 1 为缺页
13.             pageList[n].time = 0; //置访问历史为0
14.             if (randBool()) {
15.                 pageList[n].isVisit = 1;
16.                 pageList[n].isModify = 1;
17.             }
18.             else {
19.                 pageList[n].isVisit = 1;
20.             }
21.             return 1;
22.         }
23.     }
24.     is_loss_page = 1; // 页面不存在, 缺页
25.     return 0;
26. }

```

- 功能: 检查某个页面是否在内存中。
- 返回值: 返回 1 表示页面存在, 返回 0 表示页面缺失, 同时更新缺页标志 is\_loss\_page。

### 3.2.8 页面替换算法

#### 3.2.8.1 FIFO

```

1.  void FIFO(void)
2.  {
3.      int n, full, status;
4.      replace_page = 0; // 页面替换指针初始化为0
5.      page_loss_num = 0; // 缺页数初始化为0
6.      full = 0; // 是否装满所有的页框
7.      for (n = 0; n < pInfo.allocated_page_num; n++) // 清除页框信息
8.          pageList[n].pages = -1;
9.      is_loss_page = 0; // 缺页标志, 0 为不缺页, 1 为缺页

```

```

10.     for (current_page = 0; current_page < pInfo.visit_list_length; current_page++) // 执行算法
11.     {
12.         status = isExist(pInfo.pages[current_page]); // 查找页面是否在内存
13.         if (full < pInfo.allocated_page_num) // 开始时不计算缺页
14.         {
15.             if (status == 0) // 页不存在则装入页面
16.             {
17.                 pageList[replace_page].pages = pInfo.pages[current_page];
18.                 replace_page = (replace_page + 1) % pInfo.allocated_page_num;
19.                 full++;
20.             }
21.         }
22.         else // 正常缺页置换
23.         {
24.             if (status == 0) // 页不存在则置换页面
25.             {
26.                 pageList[replace_page].pages = pInfo.pages[current_page];
27.                 replace_page = (replace_page + 1) % pInfo.allocated_page_num;
28.                 pInfo.page_missing_num++; // 缺页次数加1
29.             }
30.         }
31.         Sleep(10);
32.         showState(); // 显示当前状态
33.     } // 置换算法循环结束
34.     _getch();
35.     return;
36. }

```

- 逻辑: 先进先出的策略, 即最早进入内存的页面最先被替换。
- 内存满: 使用 full 变量跟踪当前内存中已用的页框数量, 利用取模运算来循环替换。

### 3.2.8.2 OPT

```

1.     void OPT()
2.     {
3.         int n, full, status;
4.         replace_page = 0; // 页面替换指针初始化为0
5.         page_loss_num = 0; // 缺页数初始化为0
6.
7.         full = 0; // 是否装满所有的页框
8.         for (n = 0; n < pInfo.allocated_page_num; n++) // 清除页框信息
9.         {
10.            pageList[n].pages = -1;
11.        }
12.        is_loss_page = 0; // 缺页标志, 0 为不缺页, 1 为缺页
13.        for (current_page = 0; current_page < pInfo.visit_list_length; current_page++) // 执行算法
14.        {

```



```

15.         status = isExist(pInfo.pages[current_page]); // 查找页面是否在内存
16.         if (full < pInfo.allocated_page_num) // 开始时不计算缺页
17.         {
18.             if (status == 0) // 页不存在则装入页面
19.             {
20.                 pageList[replace_page].pages = pInfo.pages[current_page];
21.                 replace_page = (replace_page + 1) % pInfo.allocated_page_num;
22.                 full++;
23.             }
24.         }
25.         else // 正常缺页置换
26.         {
27.             if (status == 0) // 页不存在,则置换页面
28.             {
29.                 int min,max = 0 ; // 很大的数
30.                 for (int m = 0; m < pInfo.allocated_page_num ; m++)
31.                 {
32.                     min = 1000;
33.                     for (int n = current_page; n < pInfo.visit_list_length; n++)
34.                     {
35.                         if (pInfo.pages[n] == pageList[m].pages)
36.                         {
37.                             min = n;
38.                             break;
39.                         }
40.                     }
41.                     if (max < min)
42.                     {
43.                         max = min;
44.                         replace_page = m;
45.                     }
46.                 }
47.                 pageList[replace_page].pages = pInfo.pages[current_page];
48.                 replace_page = (replace_page + 1) % pInfo.allocated_page_num;
49.                 pInfo.page_missing_num++; // 缺页次数加1
50.             }
51.         }
52.         Sleep(10);
53.         showState(); // 显示当前状态
54.     } // 置换算法循环结束
55.     _getch();
56.     return;
57. }

```

- 逻辑: OPT 算法选择未来最长时间不再被访问的页面进行替换。
- 查找最远页面: 通过遍历后续访问列表, 找到最远使用的页面, 决定替换。

### 3.2.8.3 LRU

```

1.  void LRU(void)
2.  {
3.      int n, full, status, max;
4.      replace_page = 0;          // 页面替换指针
5.      page_loss_num = 0;        // 缺页次数初始化为0
6.
7.      full = 0;                  // 是否装满所有的页框
8.      for (n = 0; n < pInfo.allocated_page_num; n++)
9.      {
10.         pageList[n].pages = -1;    // 清除页框信息
11.         pageList[n].time = 0;      // 清除页框历史
12.     }
13.     is_loss_page = 0;            // 缺页标志, 0 为不缺页, 1 为缺页
14.     for (current_page = 0; current_page < pInfo.visit_list_length; current_page++) // 执行算法
15.     {
16.         status = isExist(pInfo.pages[current_page]); // 查找页面是否在内存
17.         if (full < pInfo.allocated_page_num) // 开始时不计算缺页
18.         {
19.             if (status == 0) // 页不存在则装入页面
20.             {
21.                 pageList[replace_page].pages = pInfo.pages[current_page]; // 把要调入的页面放入一个空
                // 的页框里
22.                 replace_page = (replace_page + 1) % pInfo.allocated_page_num;
23.                 full++;
24.             }
25.         }
26.         else // 正常缺页置换
27.         {
28.             if (status == 0) // 页不存在则置换页面
29.             {
30.                 max = 0;
31.                 for (n = 1; n < pInfo.allocated_page_num; n++)
32.                 {
33.                     if (pageList[n].time > pageList[max].time)
34.                     {
35.                         max = n;
36.                     }
37.                 }
38.                 replace_page = max;
39.                 pageList[replace_page].pages = pInfo.pages[current_page];
40.                 pageList[replace_page].time = 0;

```

```

41.         pInfo.page_missing_num++; // 缺页次数加1
42.     }
43. }
44.     Sleep(10);
45.     showState(); // 显示当前状态
46. } // 置换算法循环结束
47.     _getch();
48.     return;
49. }

```

- 逻辑: LRU 算法选择最近最少使用的页面进行替换。
- 使用时间: 通过比较时间戳, 找到最久未被访问的页面。

#### 3.2.8.4 CLOCK

```

1.     void CLOCK_pro(int choose)
2.     {
3.         int n, full, status;
4.         int num = -1;
5.         replace_page = 0; // 页面替换指针初始化为0
6.         page_loss_num = 0; // 缺页数初始化为0
7.
8.         full = 0; // 是否装满所有的页框
9.         for (n = 0; n < pInfo.allocated_page_num; n++) // 清除页框信息
10.            {
11.                pageList[n].pages = -1;
12.                pageList[n].isModify = 0;
13.                pageList[n].isVisit = 0;
14.                pageList[n].time = 0;
15.            }
16.         is_loss_page = 0; // 缺页标志, 0 为不缺页, 1 为缺页
17.         for (current_page = 0; current_page < pInfo.visit_list_length; current_page++) // 执行算法
18.            {
19.                status = isExist(pInfo.pages[current_page]); // 查找页面是否在内存
20.                if (full < pInfo.allocated_page_num) // 开始时不计算缺页
21.                {
22.                    if (status == 0) // 页不存在则装入页面
23.                    {
24.                        pageList[replace_page].pages = pInfo.pages[current_page];
25.                        replace_page = (replace_page + 1) % pInfo.allocated_page_num;
26.                        pageList[n].isVisit = 1;
27.                        full++;
28.                    }
29.                }
30.                else // 正常缺页置换
31.                {
32.                    if (status == 0) // 页不存在则置换页面

```

```

33.         {
34.             if(choose==1)
35.                 replace_page = replace_page_pro(++num); // 当choose =1 时采用改进的clock 算法
36.             else if(choose==0)
37.                 replace_page = replace_page_clock(++num); // 当choose =0 时采用基本的clock
                算法
38.                 pagelist[replace_page].pages = pInfo.pages[current_page];
39.                 pagelist[replace_page].isVisit = 1;
40.                 pInfo.page_missing_num++; // 缺页次数加1
41.             }
42.         }
43.         Sleep(10);
44.         showState(); // 显示当前状态
45.     } // 置换算法循环结束
46.     _getch();
47.     return;
48. }

```

- 逻辑: Clock 算法使用访问位来决定替换页面, 类似于环形缓冲区。
- 指针控制: 使用 **pointer** 指向当前检查的页面, 循环查找访问位为 0 的页面进行替换。

### 3.2.8.5 主函数

```

1.  int main()
2.  {
3.      char ch;
4.      system("cls");
5.      printf("*****页面置换算法*****\n");
6.      int m,e,s;
7.      printf("请输入工作集移动率(m):");
8.      scanf("%d",&m); //8
9.      printf("请输入工作集包含的页数(e):");
10.     scanf("%d",&e); //8
11.     printf("请输入重复次数(s):");
12.     scanf("%d",&s); // 10
13.     getRandomList(m, e, s); // 随机生成访问序列
14.     clock_t start, finish;
15.     double totaltime;
16.     while (true)
17.     {
18.         printf("=====MENU=====\\n");
19.         printf("1. 最佳淘汰算法(OPT)\\n");
20.         printf("2. 先进先出淘汰算法(FIFO)\\n");
21.         printf("3. 最近最久未使用淘汰算法(LRU)\\n");
22.         printf("4. 基本的 Clock 淘汰算法\\n");
23.         printf("5. 改进型的 Clock 淘汰算法\\n");
24.         printf("=====\\n");

```

```

25.     printf("请输入选择的算法(1/2/3/4/5): ");
26.     ch = (char)_getch();
27.     switch (ch) {
28.         case '1':
29.             printf("\n《-----1. 最佳淘汰算法(OPT)-----》\n");
30.             start = clock();
31.             OPT();
32.             finish = clock();
33.             totaltime = (double)(finish - start) / CLOCKS_PER_SEC;
34.             cout << "\n 运行总时长= " << totaltime << " 秒" << endl;
35.             break;
36.         case '2':
37.             printf("\n\n《-----2. 先进先出淘汰算法(FIFO)-----》\n");
38.             start = clock();
39.             FIFO();
40.             finish = clock();
41.             totaltime = (double)(finish - start) / CLOCKS_PER_SEC;
42.             cout << "\n 运行总时长= " << totaltime << " 秒" << endl;
43.             break;
44.         case '3':
45.             printf("\n\n《-----3. 最近最久未使用淘汰算法(LRU)-----》\n");
46.             start = clock();
47.             LRU();
48.             finish = clock();
49.             totaltime = (double)(finish - start) / CLOCKS_PER_SEC;
50.             cout << "\n 运行总时长= " << totaltime << " 秒" << endl;
51.             break;
52.         case '4':
53.             printf("\n\n《----- 4. Clock 淘汰算法-----》\n");
54.             start = clock();
55.             CLOCK_pro(0);
56.             finish = clock();
57.             totaltime = (double)(finish - start) / CLOCKS_PER_SEC;
58.             cout << "\n 运行总时长= " << totaltime << "秒" << endl;
59.             break;
60.         case '5':
61.             printf("\n\n《----- 5. 改进的 Clock 淘汰算法-----》\n");
62.             start = clock();
63.             CLOCK_pro(1);
64.             finish = clock();
65.             totaltime = (double)(finish - start) / CLOCKS_PER_SEC;
66.             cout << "\n 运行总时长= " << totaltime << "秒" << endl;
67.             break;
68.         default:

```

```

69.         return 0;
70.     }
71. }
72.     return 0;
73. }

```

- 功能: 主函数是程序的入口点, 提供用户选择不同页面替换算法的菜单。
- 循环输入: 使用 `_getch()` 获取用户输入, 选择相应的算法进行处理。
- 清屏: 使用 `system("cls")` 清理控制台输出, 确保每次显示都干净整洁。

## 4 实验结果展示

```

=====
请输入选择的算法(1/2/3/4/5):
《-----1. 最佳淘汰算法(OPT)-----》

=====页面访问序列=====
30 28 31 27 29 28 28 27 35 33
30 35 28 29 30 28 35 39 33 39
=====

访问 30 -->内存空间[ 30 ] --> 缺页中断 --> 置换率 = 3400.0
访问 28 -->内存空间[ 30 28 ] --> 缺页中断 --> 置换率 = 3400.0
访问 31 -->内存空间[ 30 28 31 ] --> 缺页中断 --> 置换率 = 1700.0
访问 27 -->内存空间[ 30 28 27 ] --> 缺页中断 --> 置换率 = 1166.7
访问 29 -->内存空间[ 29 28 27 ] --> 缺页中断 --> 置换率 = 900.0
访问 28 -->内存空间[ 29 28 27 ]
访问 28 -->内存空间[ 29 28 27 ]
访问 27 -->内存空间[ 29 28 27 ]
访问 35 -->内存空间[ 29 28 35 ] --> 缺页中断 --> 置换率 = 462.5
访问 33 -->内存空间[ 33 28 35 ] --> 缺页中断 --> 置换率 = 422.2
访问 30 -->内存空间[ 30 28 35 ] --> 缺页中断 --> 置换率 = 390.0
访问 35 -->内存空间[ 30 28 35 ]
访问 28 -->内存空间[ 30 28 35 ]
访问 29 -->内存空间[ 30 28 29 ] --> 缺页中断 --> 置换率 = 307.7
访问 30 -->内存空间[ 30 28 29 ]
访问 28 -->内存空间[ 30 28 29 ]
访问 35 -->内存空间[ 35 28 29 ] --> 缺页中断 --> 置换率 = 256.2
访问 39 -->内存空间[ 39 28 29 ] --> 缺页中断 --> 置换率 = 247.1
访问 33 -->内存空间[ 39 33 29 ] --> 缺页中断 --> 置换率 = 238.9
访问 39 -->内存空间[ 39 33 29 ]

运行总时长= 0.676 秒

```

图 1 最佳淘汰算法结果展示

```

《-----2. 先进先出淘汰算法(FIFO)-----》

=====页面访问序列=====
30 28 31 27 29 28 28 27 35 33
30 35 28 29 30 28 35 39 33 39
=====

访问 30 -->内存空间[ 30 ] --> 缺页中断 --> 置换率 = 4300.0
访问 28 -->内存空间[ 30 28 ] --> 缺页中断 --> 置换率 = 4300.0
访问 31 -->内存空间[ 30 28 31 ] --> 缺页中断 --> 置换率 = 2150.0
访问 27 -->内存空间[ 27 28 31 ] --> 缺页中断 --> 置换率 = 1466.7
访问 29 -->内存空间[ 27 29 31 ] --> 缺页中断 --> 置换率 = 1125.0
访问 28 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 920.0
访问 28 -->内存空间[ 27 29 28 ]
访问 27 -->内存空间[ 27 29 28 ]
访问 35 -->内存空间[ 35 29 28 ] --> 缺页中断 --> 置换率 = 587.5
访问 33 -->内存空间[ 35 33 28 ] --> 缺页中断 --> 置换率 = 533.3
访问 30 -->内存空间[ 35 33 30 ] --> 缺页中断 --> 置换率 = 490.0
访问 35 -->内存空间[ 35 33 30 ]
访问 28 -->内存空间[ 28 33 30 ] --> 缺页中断 --> 置换率 = 416.7
访问 29 -->内存空间[ 28 29 30 ] --> 缺页中断 --> 置换率 = 392.3
访问 30 -->内存空间[ 28 29 30 ]
访问 28 -->内存空间[ 28 29 30 ]
访问 35 -->内存空间[ 28 29 35 ] --> 缺页中断 --> 置换率 = 325.0
访问 39 -->内存空间[ 39 29 35 ] --> 缺页中断 --> 置换率 = 311.8
访问 33 -->内存空间[ 39 33 35 ] --> 缺页中断 --> 置换率 = 300.0
访问 39 -->内存空间[ 39 33 35 ]

运行总时长= 0.756 秒

```

图 2 先进先出淘汰算法结果展示

```

《-----3. 最近最久未使用淘汰算法(LRU)-----》

页面访问序列
30 28 31 27 29 28 28 27 35 33
30 35 28 29 30 28 35 39 33 39

访问 30 -->内存空间[ 30 ] --> 缺页中断 --> 置换率 = 0.0
访问 28 -->内存空间[ 30 28 ] --> 缺页中断 --> 置换率 = 0.0
访问 31 -->内存空间[ 30 28 31 ] --> 缺页中断 --> 置换率 = 0.0
访问 27 -->内存空间[ 27 28 31 ] --> 缺页中断 --> 置换率 = 33.3
访问 29 -->内存空间[ 27 29 31 ] --> 缺页中断 --> 置换率 = 50.0
访问 28 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 60.0
访问 27 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 50.0
访问 35 -->内存空间[ 27 35 28 ] --> 缺页中断 --> 置换率 = 55.6
访问 33 -->内存空间[ 27 35 33 ] --> 缺页中断 --> 置换率 = 60.0
访问 30 -->内存空间[ 30 35 33 ] --> 缺页中断 --> 置换率 = 58.3
访问 35 -->内存空间[ 30 35 28 ] --> 缺页中断 --> 置换率 = 61.5
访问 28 -->内存空间[ 29 35 28 ] --> 缺页中断 --> 置换率 = 64.3
访问 29 -->内存空间[ 29 30 28 ] --> 缺页中断 --> 置换率 = 62.5
访问 30 -->内存空间[ 35 30 28 ] --> 缺页中断 --> 置换率 = 64.7
访问 39 -->内存空间[ 35 39 28 ] --> 缺页中断 --> 置换率 = 66.7
访问 33 -->内存空间[ 35 39 33 ] --> 缺页中断 --> 置换率 = 66.7
访问 39 -->内存空间[ 35 39 33 ] --> 缺页中断 --> 置换率 = 66.7

运行总时长= 0.504 秒

```

图 3 最近最久未使用淘汰算法结果展示

```

《-----4. Clock 淘汰算法-----》

页面访问序列
30 28 31 27 29 28 28 27 35 33
30 35 28 29 30 28 35 39 33 39

访问 30 -->内存空间[ 30 ] --> 缺页中断 --> 置换率 = 1200.0
访问 28 -->内存空间[ 30 28 ] --> 缺页中断 --> 置换率 = 1200.0
访问 31 -->内存空间[ 30 28 31 ] --> 缺页中断 --> 置换率 = 600.0
访问 27 -->内存空间[ 27 28 31 ] --> 缺页中断 --> 置换率 = 433.3
访问 29 -->内存空间[ 27 29 31 ] --> 缺页中断 --> 置换率 = 350.0
访问 28 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 300.0
访问 27 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 200.0
访问 35 -->内存空间[ 35 29 28 ] --> 缺页中断 --> 置换率 = 188.9
访问 33 -->内存空间[ 35 33 28 ] --> 缺页中断 --> 置换率 = 180.0
访问 30 -->内存空间[ 35 33 30 ] --> 缺页中断 --> 置换率 = 158.3
访问 35 -->内存空间[ 28 33 30 ] --> 缺页中断 --> 置换率 = 153.8
访问 28 -->内存空间[ 28 29 30 ] --> 缺页中断 --> 置换率 = 131.2
访问 29 -->内存空间[ 28 29 30 ] --> 缺页中断 --> 置换率 = 129.4
访问 30 -->内存空间[ 28 29 30 ] --> 缺页中断 --> 置换率 = 127.8
访问 39 -->内存空间[ 39 29 35 ] --> 缺页中断 --> 置换率 = 127.8
访问 33 -->内存空间[ 39 33 35 ] --> 缺页中断 --> 置换率 = 127.8
访问 39 -->内存空间[ 39 33 35 ] --> 缺页中断 --> 置换率 = 127.8

运行总时长= 0.936秒

```

图 4 CLOCK 淘汰算法结果展示

```

《-----5. 改进的Clock 淘汰算法-----》

页面访问序列
30 28 31 27 29 28 28 27 35 33
30 35 28 29 30 28 35 39 33 39

访问 30 -->内存空间[ 30 ] --> 缺页中断 --> 置换率 = 2300.0
访问 28 -->内存空间[ 30 28 ] --> 缺页中断 --> 置换率 = 2300.0
访问 31 -->内存空间[ 30 28 31 ] --> 缺页中断 --> 置换率 = 1150.0
访问 27 -->内存空间[ 27 28 31 ] --> 缺页中断 --> 置换率 = 800.0
访问 29 -->内存空间[ 27 29 31 ] --> 缺页中断 --> 置换率 = 625.0
访问 28 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 520.0
访问 27 -->内存空间[ 27 29 28 ] --> 缺页中断 --> 置换率 = 337.5
访问 35 -->内存空间[ 35 29 28 ] --> 缺页中断 --> 置换率 = 311.1
访问 33 -->内存空间[ 35 33 28 ] --> 缺页中断 --> 置换率 = 290.0
访问 30 -->内存空间[ 35 33 30 ] --> 缺页中断 --> 置换率 = 250.0
访问 28 -->内存空间[ 28 33 30 ] --> 缺页中断 --> 置换率 = 238.5
访问 29 -->内存空间[ 28 29 30 ] --> 缺页中断 --> 置换率 = 200.0
访问 30 -->内存空间[ 35 29 30 ] --> 缺页中断 --> 置换率 = 194.1
访问 39 -->内存空间[ 35 39 30 ] --> 缺页中断 --> 置换率 = 188.9
访问 33 -->内存空间[ 35 39 33 ] --> 缺页中断 --> 置换率 = 188.9
访问 39 -->内存空间[ 35 39 33 ] --> 缺页中断 --> 置换率 = 188.9

运行总时长= 0.704秒

```

图 5 改进的 CLOCK 淘汰算法结果展示

## 5 实验心得体会

在这次页面替换算法的实验中，我深入理解了操作系统中内存管理的重要性及其复杂性。通过实现和比较多种页面替换算法，我不仅掌握了每种算法的基本原理，还认识到它们在实际应用中的优缺点。

首先，实验让我认识到页面替换的必要性。在现代操作系统中，由于物理内存有限，程序需要频繁地从硬盘调入和调出数据，页面替换算法则是解决这一问题的关键。通过不同的算法，我观察到在面对相同的访问序列时，缺页次数的差异显著。

其次，我体会到最佳淘汰算法（OPT）的理论最优性。虽然它能实现最低的缺页率，但在实际中难以实现，因为它需要预知未来的页面访问序列。这使我意识到，虽然理论模型提供了指导，但现实中的实现常常面临各种限制。

先进先出（FIFO）算法虽然简单易懂，但在某些情况下可能导致频繁的缺页，特别是在页面访问模式呈现局部性时。通过实验，我体会到了 FIFO 的劣势，例如“罄尽问题”现象的产生，即内存中长期驻留的页面并不一定是最常访问的。

在实现最近最久未使用（LRU）算法时，我感受到了它的复杂性，但其合理性让我十分赞同。LRU 考虑了页面的历史使用情况，使得最近未使用的页面更可能被替换，从而在大多数情况下提高了内存的利用率。

时钟算法的实现让我觉得它是一个很好的折中方案。通过简单的访问位管理，时钟算法在性能和实现复杂度之间找到了平衡。尤其是改进的时钟算法，进一步考虑了页面的修改位，使得替换决策更为高效。

最后，通过统计实验数据，我深刻理解了缺页率和运行时间之间的关系。在分析每种算法的表现时，我不仅关注缺页次数，还注意到了算法的运行时间。这让我明白，选择一个合适的页面替换算法，不仅要考虑其缺页率，还要综合考虑实现的复杂度和系统的实际需求。

总的来说，这次实验加深了我对操作系统中内存管理机制的理解，提升了我的编程能力和分析问题的思维方式。我相信这些经验将在我未来的学习和工作中发挥重要的作用。