****

**操作系统**

**课程设计报告**

题目;页面置换算法

专业班级：计科1505

姓名：陶婉莹

学号：0902150521

完成时间：2018年4月18日

目录

**[第一章 概述 3](#_Toc14782)**

[1.1 设计题目 3](#_Toc860)

[1.2 设计目的 3](#_Toc21244)

[1.3 设计要求 3](#_Toc62)

**[第二章 程序设计 4](#_Toc18628)**

[2.1功能设计 4](#_Toc20542)

[2.2运行流程 5](#_Toc24795)

[2.3详细算法 8](#_Toc7000)

**[第三章 运行结果 13](#_Toc17048)**

[3.1初始化界面 13](#_Toc13733)

[3.2功能选择界面 14](#_Toc6984)

[3.3运行结果界面 15](#_Toc623)

[3.4动态演示界面 16](#_Toc18294)

**[第四章 设计心得 17](#_Toc219)**

**第一章 概述**

1.1 设计题目

页面置换算法

1.2 设计目的

深入掌握内存调度算法的概念与案例和实现方法

1.3 设计要求

编写程序实现

1. 先进先出页面置换算法（FIFO）
2. 最近最久未使用页面置换算法（LRU）
3. 最佳置换页面置换算法（OPT）

设计一个虚拟存储区和内存工作区，编程序演示以上三种算法的具体实现过程，并计算访问命中率，演示页面置换的三种算法，通过随机数产生一个指令序列，将指令序列转换为页地址流，计算并输出各种算法在不同内存容量下的命中率。

**第二章 程序设计**

2.1功能设计

1. 产生随机序列功能

随机生成1-128之间的整数，作为指令序列号，同时将随机生成的数字除以10取余作为该指令的页地址，随机数的生成以当前时钟做种子，保证每次生成的随机性。

1. 算法运行功能
2. 根据先进先出算法进行页面置换
3. 根据最近最久未使用算法进行页面置换
4. 根据最佳置换页面算法进行页面置换
5. 结果分析功能
6. 计算先进先出算法命中率
7. 计算最近最久未使用算法命中率
8. 计算最佳置换页面算法命中率
9. 分析出最优算法
10. 演示效果功能
11. 手动运行页面置换算法，一次运行一步
12. 自动运行页面置换算法，系统每个时间间隔自动运行一步

2.2运行流程

1. 程序流程图

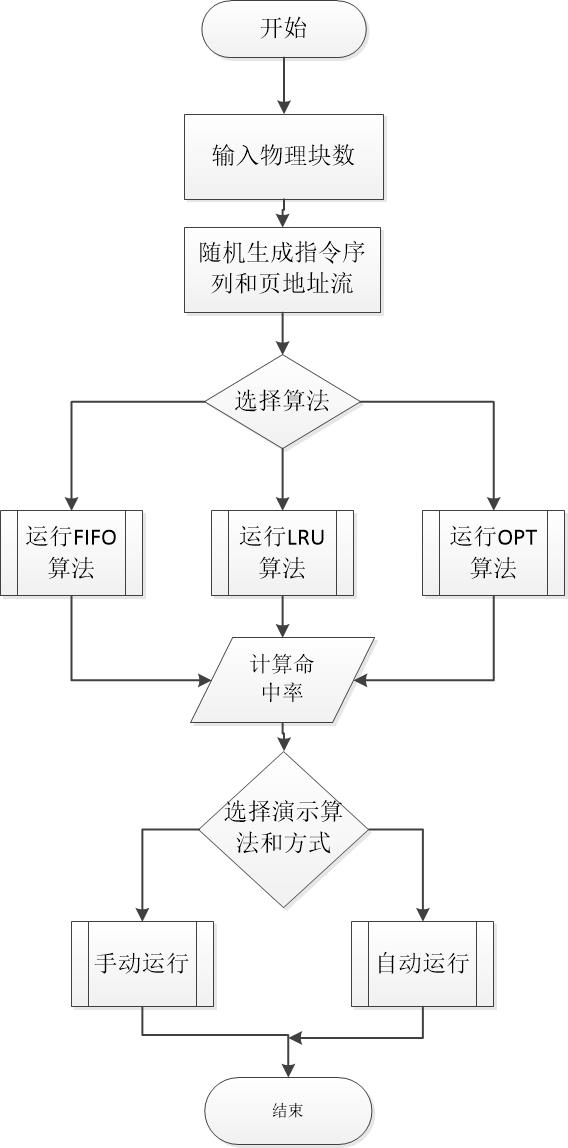


图2.1 程序流程图

1. 算法流程图

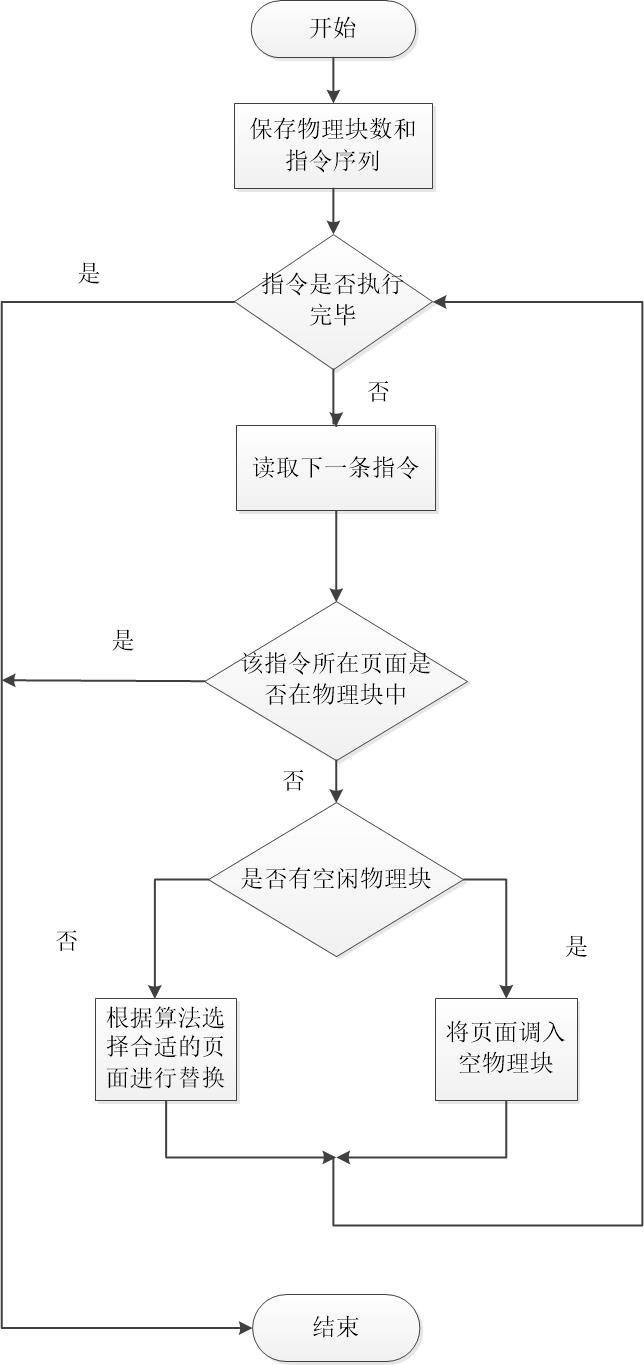


图2.2 算法流程图

2.3详细算法

1. FIFO算法

// FIFO算法

**if** (comboBox.getSelectedItem().equals("FIFO")) {

**for** (**int** a = 0; a < block; a++)

Memory[a] = -1; // 初始化为-1

**for** (**int** q = 0; q < 128; q++) {

flag = **false**;

pos++;

**int** temp = seq[pos];// 中间变量暂时存放当前指令所在页地址

// 查看是否已经存在内存中

**for** (**int** i = 0; i < block; i++) {

**if** (Memory[i] == temp) {// 如果命中，命中数+1，跳出循环

FIFOhit++;

flag = **true**;

**break**;

}

}

// 如果未命中，考虑是否要替换

**if** (!flag) {

**if** (memorynum < block) {// 如果内存没满，则不用替换

Memory[memorynum] = temp;

memorynum++;

} **else** {// 如果内存已满，选择最早调入的页面进行替换

Memory[replacepos] = temp;// Memory[replacepos]为被替换的元素，第一次满即替换memory[0]

changecolor[replacepos][pos + 1] = 1;// 标记待变颜色的表格框

replacepos = (replacepos + 1) % block;// 按照block大小来循环替换

}

}

**for** (**int** i = 0; i < block; i++)

data1[i][pos + 1] = Memory[i];

}

// 显示命中率

FIFOrate.setText(String.*valueOf*((**double**) FIFOhit / 128));

**int** a = 0;

**for** (**int** i = 0; i < block; i++) {

a = i + 1;

data1[i][0] = "第" + a + "块";

}

DefaultTableModel tableModel1 = **new** DefaultTableModel(data1, tableTitle); // 表格模型对象

JTable Result1 = **new** JTable(tableModel1);

**int** columnCount1 = Result1.getColumnCount();

Result1.getColumnModel().getColumn(0).setPreferredWidth(80);

**for** (**int** i = 1; i < columnCount1; i++) {

TableColumn tableColumn1 = Result1.getColumnModel().getColumn(i);

tableColumn1.setPreferredWidth(80);

}

Result1.setRowHeight(40);// 指定每一行的行高40

Result1.setAutoResizeMode(JTable.***AUTO\_RESIZE\_OFF***);

JScrollPane jspResult1 = **new** JScrollPane(Result1);

jspResult1.setBounds(10, 10, 1018, 320);

jspResult1.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);

jspResult1.setVerticalScrollBarPolicy(JScrollPane.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***);

TableCellRenderer tcr1 = **new** ColorTableCellRenderer(block, changecolor);

Result1.setDefaultRenderer(Object.**class**, tcr1);

JFrame FIFO = **new** JFrame("FIFO算法结果 命中率：" + (**double**) FIFOhit / 128);

FIFO.getContentPane().setLayout(**null**);

FIFO.setBounds(920, 10, 1048, 380);

FIFO.setDefaultCloseOperation(2);

FIFO.getContentPane().add(jspResult1);

FIFO.setVisible(**true**);

}

1. LRU算法

// FIFO算法

**if** (comboBox.getSelectedItem().equals("FIFO")) {

**for** (**int** a = 0; a < block; a++)

Memory[a] = -1; // 初始化为-1

**for** (**int** q = 0; q < 128; q++) {

flag = **false**;

pos++;

**int** temp = seq[pos];// 中间变量暂时存放当前指令所在页地址

// 查看是否已经存在内存中

**for** (**int** i = 0; i < block; i++) {

**if** (Memory[i] == temp) {// 如果命中，命中数+1，跳出循环

FIFOhit++;

flag = **true**;

**break**;

}

}

// 如果未命中，考虑是否要替换

**if** (!flag) {

**if** (memorynum < block) {// 如果内存没满，则不用替换

Memory[memorynum] = temp;

memorynum++;

} **else** {// 如果内存已满，选择最早调入的页面进行替换

Memory[replacepos] = temp;// Memory[replacepos]为被替换的元素，第一次满即替换memory[0]

changecolor[replacepos][pos + 1] = 1;// 标记待变颜色的表格框

replacepos = (replacepos + 1) % block;// 按照block大小来循环替换

}

}

**for** (**int** i = 0; i < block; i++)

data1[i][pos + 1] = Memory[i];

}

// 显示命中率

FIFOrate.setText(String.*valueOf*((**double**) FIFOhit / 128));

**int** a = 0;

**for** (**int** i = 0; i < block; i++) {

a = i + 1;

data1[i][0] = "第" + a + "块";

}

DefaultTableModel tableModel1 = **new** DefaultTableModel(data1, tableTitle); // 表格模型对象

JTable Result1 = **new** JTable(tableModel1);

**int** columnCount1 = Result1.getColumnCount();

Result1.getColumnModel().getColumn(0).setPreferredWidth(80);

**for** (**int** i = 1; i < columnCount1; i++) {

TableColumn tableColumn1 = Result1.getColumnModel().getColumn(i);

tableColumn1.setPreferredWidth(80);

}

Result1.setRowHeight(40);// 指定每一行的行高40

Result1.setAutoResizeMode(JTable.***AUTO\_RESIZE\_OFF***);

JScrollPane jspResult1 = **new** JScrollPane(Result1);

jspResult1.setBounds(10, 10, 1018, 320);

jspResult1.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);

jspResult1.setVerticalScrollBarPolicy(JScrollPane.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***);

TableCellRenderer tcr1 = **new** ColorTableCellRenderer(block, changecolor);

Result1.setDefaultRenderer(Object.**class**, tcr1);

JFrame FIFO = **new** JFrame("FIFO算法结果 命中率：" + (**double**) FIFOhit / 128);

FIFO.getContentPane().setLayout(**null**);

FIFO.setBounds(920, 10, 1048, 380);

FIFO.setDefaultCloseOperation(2);

FIFO.getContentPane().add(jspResult1);

FIFO.setVisible(**true**);

}

1. OPT算法

// FIFO算法

**if** (comboBox.getSelectedItem().equals("FIFO")) {

**for** (**int** a = 0; a < block; a++)

Memory[a] = -1; // 初始化为-1

**for** (**int** q = 0; q < 128; q++) {

flag = **false**;

pos++;

**int** temp = seq[pos];// 中间变量暂时存放当前指令所在页地址

// 查看是否已经存在内存中

**for** (**int** i = 0; i < block; i++) {

**if** (Memory[i] == temp) {// 如果命中，命中数+1，跳出循环

FIFOhit++;

flag = **true**;

**break**;

}

}

// 如果未命中，考虑是否要替换

**if** (!flag) {

**if** (memorynum < block) {// 如果内存没满，则不用替换

Memory[memorynum] = temp;

memorynum++;

} **else** {// 如果内存已满，选择最早调入的页面进行替换

Memory[replacepos] = temp;// Memory[replacepos]为被替换的元素，第一次满即替换memory[0]

changecolor[replacepos][pos + 1] = 1;// 标记待变颜色的表格框

replacepos = (replacepos + 1) % block;// 按照block大小来循环替换

}

}

**for** (**int** i = 0; i < block; i++)

data1[i][pos + 1] = Memory[i];

}

// 显示命中率

FIFOrate.setText(String.*valueOf*((**double**) FIFOhit / 128));

**int** a = 0;

**for** (**int** i = 0; i < block; i++) {

a = i + 1;

data1[i][0] = "第" + a + "块";

}

DefaultTableModel tableModel1 = **new** DefaultTableModel(data1, tableTitle); // 表格模型对象

JTable Result1 = **new** JTable(tableModel1);

**int** columnCount1 = Result1.getColumnCount();

Result1.getColumnModel().getColumn(0).setPreferredWidth(80);

**for** (**int** i = 1; i < columnCount1; i++) {

TableColumn tableColumn1 = Result1.getColumnModel().getColumn(i);

tableColumn1.setPreferredWidth(80);

}

Result1.setRowHeight(40);// 指定每一行的行高40

Result1.setAutoResizeMode(JTable.***AUTO\_RESIZE\_OFF***);

JScrollPane jspResult1 = **new** JScrollPane(Result1);

jspResult1.setBounds(10, 10, 1018, 320);

jspResult1.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);

jspResult1.setVerticalScrollBarPolicy(JScrollPane.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***);

TableCellRenderer tcr1 = **new** ColorTableCellRenderer(block, changecolor);

Result1.setDefaultRenderer(Object.**class**, tcr1);

JFrame FIFO = **new** JFrame("FIFO算法结果 命中率：" + (**double**) FIFOhit / 128);

FIFO.getContentPane().setLayout(**null**);

FIFO.setBounds(920, 10, 1048, 380);

FIFO.setDefaultCloseOperation(2);

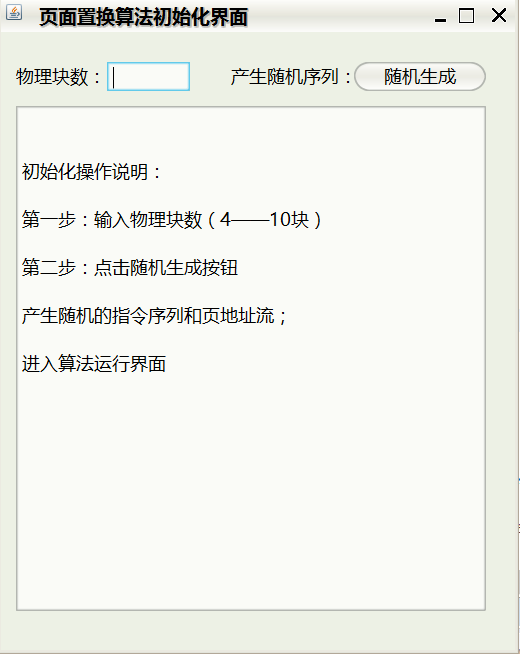
FIFO.getContentPane().add(jspResult1);

FIFO.setVisible(**true**);

}

1. **运行结果**

3.1初始化界面



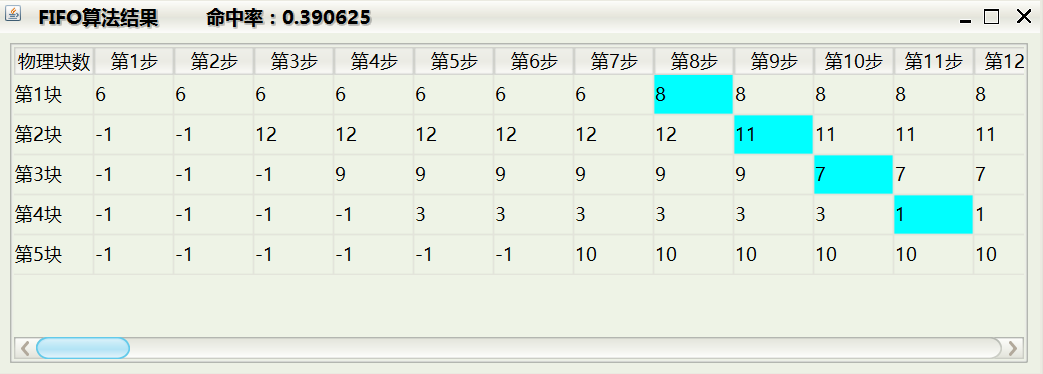


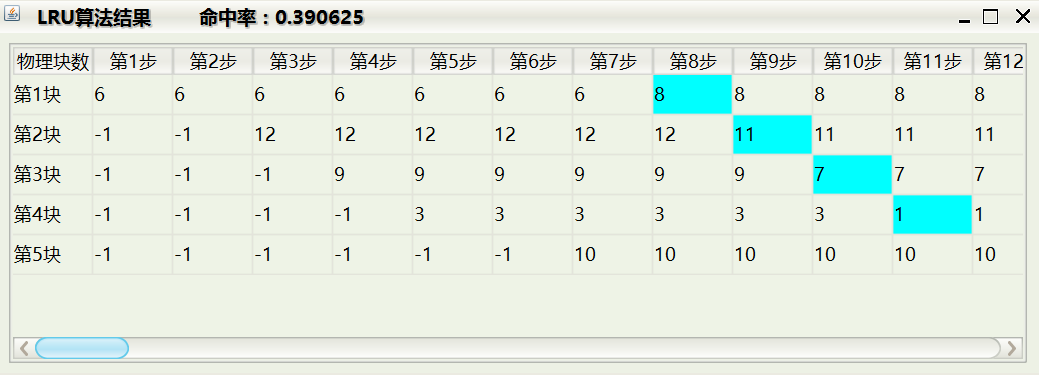
3.2功能选择界面

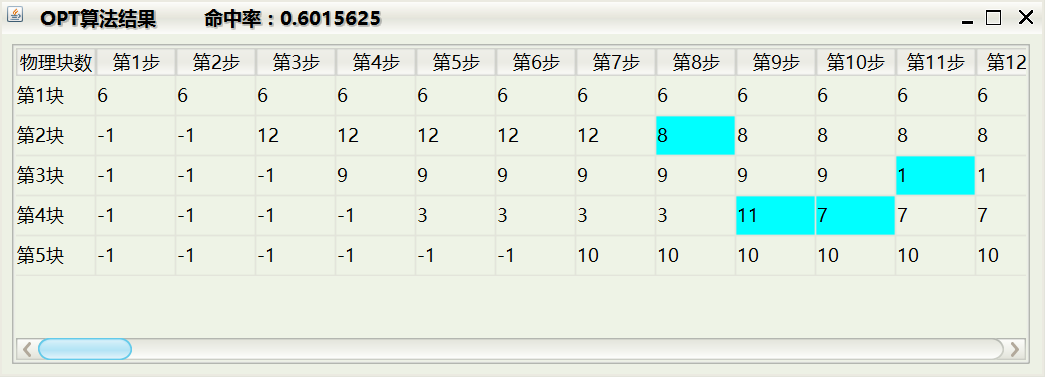


3.3运行结果界面

测试样例一



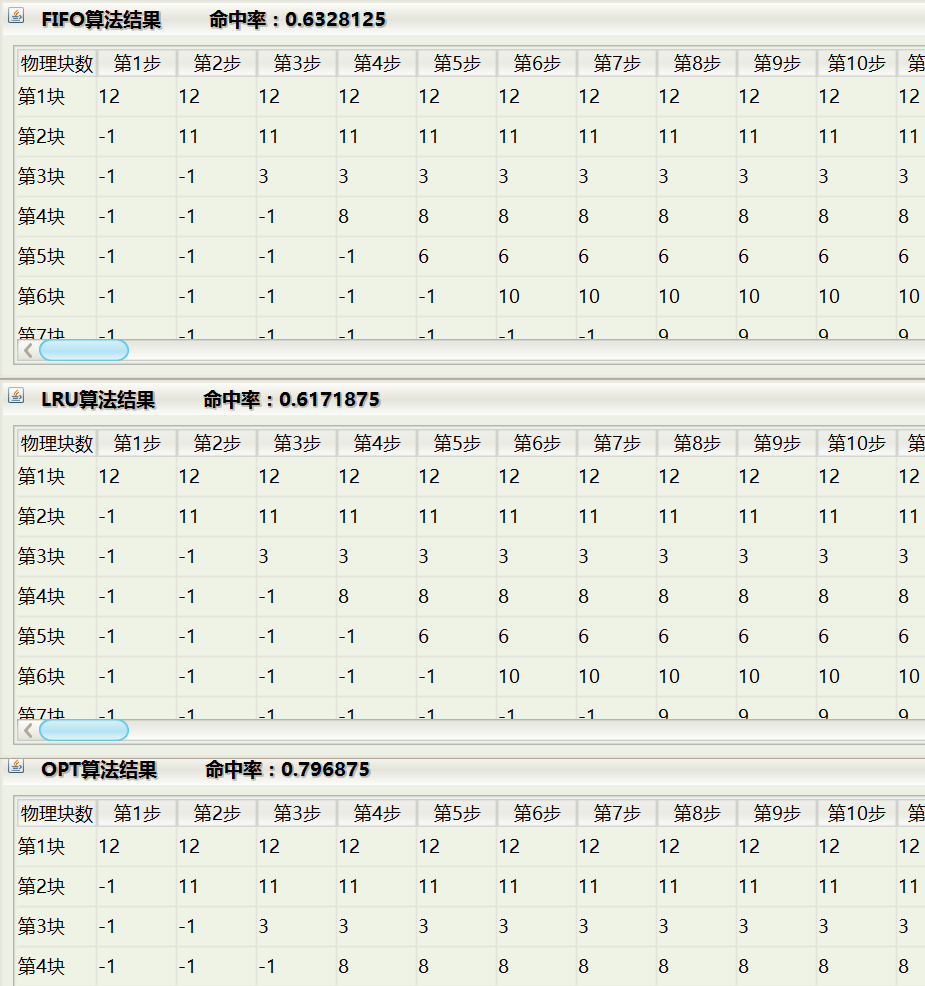




测试样例二



测试样例三



3.4动态演示界面





**第四章 设计心得**

这次的课程设计题目是页面置换算法，包括FIFO算法、LRU算法和OPT算法，因为之前上操作系统原理课的时候着重学习了这些算法，所以实现的时候并没有什么难度，比较顺利。

在完成设计要求原有的基本功能之后，我在程序加入了动态演示功能，这个功能分为手动运行和自动运行，其中自动运行使用到了线程，因为之前编程的时候并没有怎么用过线程，所以在实现的过程中遇到了一些问题，经过查询相关资料和翻阅书籍后逐一修改了程序中存在的bug，最终达到了预期的效果。比较满意。

当然此次课程设计还存在一些不足，比如在对算法运行结果分析的时候，不能只依靠一次运行结果计算出的命中率来得出哪个是最优算法，必须经过多次运行结果分析比较，才能相对地得出每个算法各自的优点以及适用场合。当然，就命中率来说，OPT算法自然是最优的，其他算法的性能也应按照OPT算法的性能来衡量。感谢老师在这次课程设计中对我提出的意见和建议，我从中学习到了更多有关操作系统页面置换算法的原理，加深了理解，自身各方面能力也有所提高。今后我一定会更加努力学习，提升编程能力。