

实 验 报 告

课程名称 操作系统 (B)

开课学期 2023-2024 学年第一学期 指导教师 喻学才

实 验 室 启天 B118

班 级 2020 软件工程 3 班

学 号

姓 名

成绩： (五级)

实验课程 评分表标准

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 全勤、学习态度 端正 、 实验认 真、积极回答问 题、操作过程正 确，结果准确， 实验报告 内容 规范 | 偶有缺勤、实验 认真、回答问题 较积极、操作过 程正确，结果准 确，实验报告内 容规范 | 旷课 2 次以内、 偶有迟到、实验 认真、回答问题 较好、操作过程 基本正确，结果 基本准确，实验 报告 内容较规 范 | 旷课 2 次以上、 学习态度一般、 基本 能回答 出 问题、操作过程 较正确，结果基 本准确，实验报 告 内容基本规 范 | 经常旷课，实验 过程不认真、问 题回答不积极、 实验报告不符 合要求或未交 |
| 优秀 (90-100) | 良好 (80-89) | 中 (70-79) | 及格 (60-69) | 不及格(<59) |
| 实验一 |  |  |  |  |  |
| 实验二 |  |  |  |  |  |
| 实验三 |  |  |  |  |  |
| 实验成绩总评 (五级制) | |  | | | |

说明：1 ．每次实验结束，学生完成一份实验报告，课程结束后汇总,加封面装订成册存档； 2 ．各系 (部) 可在以上五项栏目的基础上，可根据实验课程和实验项目的具体需要，统一 设计和调整项目内容，但封面格式应统一；3 ．对于设计性实验，只要求说明实验的目的要 求、提出可供实验的基本条件和注意事项，实验方案和步骤的设置、仪器的安排等，可由学 生自己设计；4.可根据实验数量自行添加行数。打印到封面背面

桂林航天工业学院学生实验报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 操作系统 (B) | | | 实验项目名称 | | 存储器管理 (2 学时) | | |
| 开课教学单位及实验室 | | 计算机科学与工程学院 | | | | 实验日期 | | 2023 年 12月 11日 |
| 学生姓名 |  | | 学号 | |  | 专业班级 | | 2020 软件工程 3 班 |
| 指导教师 | 喻学才 | | | | 实验成绩 | |  | |
| 一、实验目的  1. 能够描述存储管理策略，掌握分页存储管理的过程、原理和虚拟存储的实现方式。  2. 能够设计适当的数据结构编程实现虚拟存储管理的页面置换算法，并结合具体应用分析算法性能。  二、实验内容及要求  设计一个请求页式存储管理方案，用高级语言编写模拟程序实现具体过程，要求采用先进先出算法和最近最久未使用算法实现页面置换，并对算法性能进行分析。  要求实现下列页面置换算法：   1. 先进先出算法（FIFO）：淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面给予淘汰。 2. 最近最久未使用算法（LRU）：淘汰最近最久未被使用的页面。   程序中用户可选择置换算法，先输入所有页面号，为系统分配物理块，依次安装FIFO或LRU算法进行置换。   1. 实验源代码及说明 2. 实验结果及分析   **实验代码如下：**  package caozuoxiotn.demo5;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Scanner;  public class CuCunGuanLli {  static void FIFO(ArrayList<Integer> frame, ArrayList<Integer> page) {  System.out.println("============先进先出置换算法============");  // 框和页面长度  int n\_f = frame.size();  int n\_p = page.size();  // 缺页  int n\_lack = n\_f;  // 判断块:初始每个块对应的出现次数  // 因为在预装入之后才会有相应的判断  // 使用我将判断的状态直接设置成预装入之后 即为 3 2 1  ArrayList<Integer> judge = new ArrayList<Integer>(n\_f);  for (int i = 0; i < n\_f; i++) {  judge.add(3 - i);  }  for (int i = 0; i < n\_p; i++) {  System.out.print(page.get(i) + "===");  if (i < n\_f) {  // 预装入  frame.set(i, page.get(i));  System.out.println(frame);  } else {  // 每个页面存在次数加1  for (int j = 0; j < n\_f; j++) {  judge.set(j, judge.get(j) + 1);  }  if (frame.contains(page.get(i))) {  // 页面已经存在在物理块中  System.out.println("页面已经存在于物理块");  } else {  // 根据存在最久的（即judge对应最大的）替换  int index\_max = judge.indexOf(Collections.max(judge));  int rep\_page = frame.get(index\_max);  frame.set(index\_max, page.get(i));  // 将新换进的存在状态设置为1  judge.set(index\_max, 1);  System.out.print(frame);  System.out.println(" 替换掉了页面：" + rep\_page);  n\_lack = n\_lack + 1;  }  }  }  float p\_lack = 100 \* (float) n\_lack / n\_p;  System.out.println("===================================");  System.out.printf("缺页次数：%d\n", n\_lack);  System.out.printf("缺页率： %.2f%%\n", p\_lack);  System.out.println("===================================");  }  // lru 最近最久未使用算法  static void LRU(ArrayList<Integer> frame, ArrayList<Integer> page) {  System.out.println("===========最近最久未使用算法===========");  // 框和页面长度  int n\_f = frame.size();  int n\_p = page.size();  // 缺页  int n\_lack = n\_f;  // 和fifo类似先设置为 3 2 1  ArrayList<Integer> judge = new ArrayList<Integer>(n\_f);  for (int i = 0; i < n\_f; i++) {  judge.add(3 - i);  }  for (int i = 0; i < n\_p; i++) {  System.out.print(page.get(i) + "===");  if (i < n\_f) {  // 预装入  frame.set(i, page.get(i));  System.out.println(frame);  } else {  // 每个页面存在次数加1  for (int j = 0; j < n\_f; j++) {  judge.set(j, judge.get(j) + 1);  }  if (frame.contains(page.get(i))) {  // 页面已经存在在物理块中  System.out.println("页面已经存在于物理块");  // 这一步fifo没有  // 将页面的使用重置为1  judge.set(frame.indexOf(page.get(i)), 1);  } else {  // 根据最久未使用的（即judge对应最大的）替换  int index\_max = judge.indexOf(Collections.max(judge));  int rep\_page = frame.get(index\_max);  frame.set(index\_max, page.get(i));  // 将新换进的使用状态设置为1  judge.set(index\_max, 1);  System.out.print(frame);  System.out.println(" 替换掉了页面：" + rep\_page);  n\_lack = n\_lack + 1;  }  }  }  float p\_lack = 100 \* (float) n\_lack / n\_p;  System.out.println("===================================");  System.out.printf("缺页次数：%d\n", n\_lack);  System.out.printf("缺页率： %.2f%%\n", p\_lack);  System.out.println("===================================");  }  // 最佳置换算法 opt 最佳页面置换算法  static void OPT(ArrayList<Integer> frame, ArrayList<Integer> page) {  System.out.println("============最佳页面置换算法============");  // 框和页面长度  int n\_f = frame.size();  int n\_p = page.size();  // 缺页  int n\_lack = n\_f;  // 判断块:初始每个块对应的页面很大  ArrayList<Integer> judge = new ArrayList<Integer>(n\_f);  for (int i = 0; i < n\_f; i++) {  judge.add(99);  }  for (int i = 0; i < n\_p; i++) {  System.out.print(page.get(i) + "===");  if (i < n\_f) {  // 预装入  frame.set(i, page.get(i));  System.out.println(frame);  } else {  if (frame.contains(page.get(i))) {  // 页面已经存在在物理快中  System.out.println("页面已经存在于物理块");  } else {  // 更新往后页面第一次出现的位置  for (int j = 0; j < 3; j++) {  int index = 99;  for (int k = i + 1; k < n\_p; k++) {  if (frame.get(j) == page.get(k)) {  index = k;  break;  }  }  // 更新（  judge.set(j, index);  }  // 根据出现最后的（即judge对应最大的）替换  int index\_max = judge.indexOf(Collections.max(judge));  int rep\_page = frame.get(index\_max);  frame.set(index\_max, page.get(i));  System.out.print(frame);  System.out.println(" 替换掉了页面：" + rep\_page);  n\_lack = n\_lack + 1;  }  }  }  float p\_lack = 100 \* (float) n\_lack / n\_p;  System.out.println("===================================");  System.out.printf("缺页次数：%d\n", n\_lack);  System.out.printf("缺页率： %.2f%%\n", p\_lack);  System.out.println("===================================");  }  // fifo 先行先出算法 fifo 先进先出置换算法  //主方法测试  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.print("请分配给该作业的物理页框块数：");  int n\_frame = sc.nextInt(); // 物理页框数  ArrayList<Integer> frame = new ArrayList<Integer>(n\_frame);  for (int i = 0; i < n\_frame; i++) {  frame.add(-1);  }  System.out.print("请输入该作业的页面走向：");  sc.nextLine(); // 控制输入格式  String inputPages = sc.nextLine();  String[] split = inputPages.split("\\s+|,|，");  int n\_page = split.length; // 作业的页面走向总次数  ArrayList<Integer> page = new ArrayList<Integer>(n\_page); // 作业的页面走向  for (int i = 0; i < n\_page; i++) {  page.add(Integer.parseInt(split[i]));  }  sc.close();  // 测试输入  // 3  // 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  OPT(frame, page);  FIFO(frame, page);  LRU(frame, page);  }  }  **代码解析：**  这段代码一共分为了三个部分：最佳置换算法（OPT）、先进先出置换算法（FIFO）和最近最久未使用算法（LRU）。在原题目的有的基础上，添加多了一个最佳置换算法。**首先解释这三种算法有什么用和什么定义。**  （1）先进先出算法（FIFO）是最早被提出的置换算法之一，该算法按照页面进入内存的时间顺序进行页面替换，即最早进入的页面最先被替换出去；  （2）最近最久未使用算法（LRU）是根据页面的访问时间来进行页面替换决策的。该算法记录每个页面最后被访问的时间，并在需要替换页面时选择最近最少被访问的页面。这样，最近经常使用的页面将更可能保留在内存中，从而提高程序的性能。  （3）最佳置换算法（OPT）是一种理想化的模型，其目标是最小化缺页率，即在需要替换页面时选择未来最长时间内不会被访问的页面。然而，由于需要预知未来的页面访问情况，这种算法在实际操作中往往难以实现；  **分别对这三个算法进行逻辑解析：**  （1）先进先出置换算法（FIFO）：  初始化一个空的物理页框数组和一个空的页面队列。  遍历作业的页面走向，对于每个页面：  如果页面已经在物理页框中，则将其移动到页面队列的末尾。  如果页面不在物理页框中，根据以下规则选择一个页面进行替换：  如果物理页框还有空闲空间，直接将页面放入物理页框的末尾。  如果物理页框已满，直接将页面放入物理页框的末尾，并从物理页框中移除最早进入队列的页面。  计算并输出页面置换次数。  （2）最近最久未使用算法（LRU）：  初始化一个空的物理页框数组和一个空的页面队列。  遍历作业的页面走向，对于每个页面：  如果页面已经在物理页框中，则将其移动到页面队列的末尾。  如果页面不在物理页框中，根据以下规则选择一个页面进行替换：  如果物理页框还有空闲空间，直接将页面放入物理页框的末尾。  如果物理页框已满，找到页面队列中最长时间未被访问的页面进行替换。  计算并输出页面置换次数。  （3）最佳置换算法（OPT）：  初始化一个空的物理页框数组和一个空的页面队列。  遍历作业的页面走向，对于每个页面：  如果页面已经在物理页框中，则将其移动到页面队列的末尾。  如果页面不在物理页框中，根据以下规则选择一个页面进行替换：  如果物理页框还有空闲空间，直接将页面放入物理页框的末尾。  如果物理页框已满，找到页面队列中最长时间未被访问的页面进行替换。  计算并输出页面置换次数。  **总结：**通过了解这三种算法的基础上，加上算法实现的逻辑，实现出上述代码。然后通过输入指定的数据，来在主方法进行测试三种算法的效果。具体效果图如下：  输入 3 和 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  （1）先进先出FIFO 效果图如下：    （2）最近最久未使用算法（LRU）效果图：    （3）最佳置换算法（OPT）效果图：（拓展的算法） | | | | | | | | |