**安徽科技学院**

**软件类实验报告**

|  |  |
| --- | --- |
| 课程名称： | 操作系统 |
| 学 号： | 2701200421 |
| 姓 名： | 乔梁 |
| 班 级： | 计算机科学与技术204班 |
| 指导教师： | 李文才 |
| 开课学期： | 2022-2023-1 |
| 学 院： | 信息与网络工程学院 |

安徽科技学院教务处

|  |  |  |  |
| --- | --- | --- | --- |
| **实验名称** | **存储调度算法** | **实验学时** | 4 |
| **实验场所** | **软件实验室九** | **实验性质** | **验证** |
| **一、实验目标**：  通过这次的操作系统课程设计，我懂得了许多，首先是让我对操作系统磁盘调度策略有了更加深刻的认识，自己动手操作比光看书能更加了解磁盘调度的策略和原理，同时对磁盘调度的算法先来先服务算法(FCFS)有了更深刻的理解和掌握，使我能够为磁盘调度选择适当的算法，提高CPU工作效率。 | | | |
| **二、实验原理：**  使用合适的磁盘调度算法可以让操作系统更加高效。 | | | |
| **三、实验要求：**  设计page类和FIFO算法的细节和要求，优化和完整算法的结构。 | | | |
| 四、实验步骤： （包括详细实验操作步骤、流程图、代码及重要语句注释）  为了可以添加其他算法模式的代码，可以将main抽出并使用最简单的单例模式简化，使用抽象工厂模式来更好的添加调度算法代码。  package ch2;  import java.util.Scanner;  public class StartPage {  public static void main(String args[]) {  **FIFO fifo = FIFO.GetInstance();**  Scanner sc = new Scanner(System.in);  fifo.init();  int c = 16;  int pageNum, localTionPos, order;  while (c > 0) {  System.out.println("请输入指令页号：");  pageNum = sc.nextInt();   System.out.println("请输入偏移地址：");  localTionPos = sc.nextInt();   System.out.println("请选择指令：1、存指令 2、非存指令");  order = sc.nextInt();   fifo.input(pageNum, localTionPos, order);  fifo.running();  c--;  }  } }  package ch2;  import sun.security.jca.GetInstance; import testch2.Hungry;  import java.util.ArrayList; import java.util.Iterator;  public class FIFO {  **private final static FIFO fifo = new FIFO(); //饿汉式单例模式。  private FIFO(){  }  public static FIFO GetInstance(){  return fifo;  }**  private static final int PRO\_MEMORY = 4;  private static int countOldPoint;  private static int count = 4;  private static int lackTime;  private static Page[] pages = new Page[PRO\_MEMORY];  private ArrayList<Integer> usePageNumList = new ArrayList<Integer>();  public void init() {  for (int i = 0; i < pages.length; i++) {  pages[i] = new Page();  }  }  public void input(int pagenum, int localtion, int order) {  if (order == 1) {  setMod(pagenum);  }  usePageNumList.add(pagenum);  System.out.println("页面:" + pagenum);  pages[count - 1].setLocation(localtion);  }  public void running() {  // 列表置换  Iterator<Integer> it = usePageNumList.iterator();  while (it.hasNext()) {  // 对4取模，得到0，1，2，3，正好是下标  countOldPoint = countOldPoint % PRO\_MEMORY;  // 获取到输入的页号  int inPageId = it.next(); // 查找页表中是否存在  TestPageId(inPageId);  display();  }  System.out.println("缺页次数为：" + lackTime + "，缺页率是：" + (float) lackTime / usePageNumList.size());  }   private void TestPageId(int inPageId) {  if (search(inPageId)) {  // 如果为true，表示找到了指定的页号，说明数据在内存中，然后计算给出地址  System.out.println("页面" + inPageId + "已在内存中");  System.out.println(inPageId \* 1024 + searchLoc(inPageId));  } else if (count < PRO\_MEMORY) {  pages[count].setPagenum((Integer) inPageId);  System.out.println("页面" + pages[count].getPagenum() + "进入内存");  count++;  } else {  System.out.println("\*" + inPageId);  int tmp = searchMod(inPageId); // 正常情况下，为0  if (tmp == 1) {  System.out.println("OOTj");  } else {  System.out.println("INL");  }  replace(inPageId);  lackTime++;  countOldPoint++;  }  }   public boolean search(int pageId) {  // 在内存中查询给定的页码是否存在于内存  for (int i = 0; i < pages.length; i++) {  if (pages[i].getPagenum() == pageId) {  return true;  }  }  return false;  }  public int searchLoc(int pageId) {  for (int i = 0; i < pages.length; i++) {  if (pages[i].getPagenum() == pageId) {  return pages[i].getLocation();  }  }  return 0;  }   public void setMod(int pageId) {  for (int i = 0; i < pages.length; i++) {  if (pages[i].getPagenum() == pageId) {  pages[i].setModify(1);  }  }  }   public int searchMod(int pageId) {  for (int i = 0; i < pages.length; i++) {  if (pages[i].getPagenum() == pageId) {  return pages[i].getModify();  }  }  return 0;  }  public void replace(int pageId) {  // 被置换出的页号  int outPageId = -1;   // 在内存中，找到最先添加的页号，也就是在内存中存在时间最长的页号  outPageId = pages[countOldPoint].getPagenum();   // 把最长时间的页号换成最新的  pages[countOldPoint].setPagenum(pageId);   System.out.println("页面" + pageId + "进入内存，" + outPageId + "被置换出内存");  }  public void display() {  System.out.print("当前内存中的页面：");  for (Page page : pages) {  System.out.print((page.getPagenum() + " "));  }  System.out.println();  } }  package ch2;  public class Page {  private int pagenum = 0; //页号  private int flag = 0; //标志 是否在页框中  private int memory = 0; //页框号  private int modify = 0; //是否修改，1为修改  private int location = 0; // 地址偏移量   public int getPagenum() {  return pagenum;  }   public void setPagenum(int pagenum) {  this.pagenum = pagenum;  }   public int getModify() {  return modify;  }   public void setModify(int modify) {  this.modify = modify;  }   public int getLocation() {  return location;  }   public void setLocation(int location) {  this.location = location;  } }  **五、实验结果：** | | | |
| **六、实验小结：**  熟悉磁盘替换算法的操作，也可以添加和使用其他设计模式来完善代码，对个人的实战受益颇多。 | | | |