南京信息工程大学 操作系统实验报告

实验项目 FIFO页面置换算法实现

|  |  |  |  |
| --- | --- | --- | --- |
| 专业 计算机科学与技术 | 年级 22 | 学号 202283290159 | 姓名 张瑞晨 |

1. 主要实验步骤

|  |
| --- |
| 实验原理  在操作系统的内存管理中，当程序所需的页面不在内存中时会发生缺页中断。页面置换算法决定了哪个页面应被替换出内存，以为新页面腾出空间。  FIFO（先进先出）页面置换算法是一种简单的页面置换策略，其核心思想是：  按照页面进入内存的先后顺序进行管理。  当内存已满时，替换最先进入内存的页面。  实验步骤  1. 创建页面类 Page  用于表示内存中的页面对象，包含页面ID属性。  class Page {  private int id = -1;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  }  2. 创建 FIFO 类  用于实现FIFO页面置换算法，包含页面初始化、用户输入、运行算法、查找页面和替换页面的方法。  import java.util.\*;  public class FIFO {  private static final int PRO\_MEMORY = 5; // 系统分配的内存块数  private static Page[] pages = new Page[PRO\_MEMORY];  private static int countOldPoint = 0; // 记录最久的页面下标  private static int count = 0; // 记录当前在使用的总页面数  private static int lackTime = 0; // 缺页次数  private List<Integer> usePageNumList = new ArrayList<>(); // 页面使用列表  public static void main(String[] args) {  System.out.println("--------先进先出算法------------");  FIFO fifo = new FIFO();  fifo.init();  fifo.input();  fifo.running();  }  // 初始化内存页  public void init() {  for (int i = 0; i < pages.length; i++) {  pages[i] = new Page();  }  }  // 接收用户输入列表  public void input() {  System.out.print("请输入页面使用列表，以空格分开: ");  Scanner sc = new Scanner(System.in);  String input = sc.nextLine();  String[] inputList = input.split("\\s+");  try {  for (String in : inputList) {  usePageNumList.add(Integer.valueOf(in));  }  } catch (Exception e) {  System.out.println("输入的必须是数字，请重新开始!!!");  System.exit(0);  }  }  // 系统运行  public void running() {  Iterator<Integer> it = usePageNumList.iterator();  while (it.hasNext()) {  countOldPoint = countOldPoint % PRO\_MEMORY;  int inPageId = it.next();  // 查找内存中是否有该页面  if (search(inPageId)) {  System.out.println("内存中有ID为 " + inPageId + " 的页面，不进行替换");  } else if (count < PRO\_MEMORY) {  // 有空闲内存页  pages[count].setId(inPageId);  System.out.println("页号ID: " + pages[count].getId() + " 正在放入内存");  count++;  } else {  // 替换页面  replace(inPageId);  lackTime++;  countOldPoint++;  }  display();  }  System.out.println("缺页次数为: " + lackTime + ", 缺页率是: " + (float) lackTime / usePageNumList.size());  }  // 查找内存中是否有该页面  public boolean search(int pageId) {  for (Page page : pages) {  if (page.getId() == pageId) {  return true;  }  }  return false;  }  // 替换算法  public void replace(int pageId) {  int outPageId = pages[countOldPoint].getId();  pages[countOldPoint].setId(pageId);  System.out.println("页号ID: " + pageId + " 正在放入内存，页号ID: " + outPageId + " 被替换出去");  }  // 显示当前内存页  public void display() {  System.out.print("当前内存保留的页数是: ");  for (Page page : pages) {  System.out.print(page.getId() + " ");  }  System.out.println();  }  } |

1. 实验结果

|  |
| --- |
| 实验结果  输入示例  请输入页面使用列表，以空格分开: 1 2 3 4 5 1 6 7 1 2 3  输出示例  --------先进先出算法------------  请输入页面使用列表，以空格分开: 1 2 3 4 5 1 6 7 1 2 3  页号ID: 1 正在放入内存  当前内存保留的页数是: 1 -1 -1 -1 -1  页号ID: 2 正在放入内存  当前内存保留的页数是: 1 2 -1 -1 -1  页号ID: 3 正在放入内存  当前内存保留的页数是: 1 2 3 -1 -1  ...  缺页次数为: 8, 缺页率是: 0.72727275 |

1. 实验遇到问题及解决

|  |
| --- |
| 问题：用户输入错误的字符导致程序崩溃。  解决方法： 添加输入验证逻辑，确保输入的页面编号为整数。  问题：内存页显示格式不规范。  解决方法： 优化页面显示输出格式，确保显示清晰易懂。 |

1. 总结

|  |
| --- |
| 通过本实验，我们成功实现了FIFO页面置换算法，并通过输入页面访问序列验证了算法的正确性。FIFO算法虽然实现简单，但缺乏灵活性，容易导致较高的缺页率。相比之下，LRU（最近最少使用）等算法在实际应用中表现更优。因此，理解各种页面置换算法的优缺点，有助于更好地设计和优化内存管理策略。 |