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 | **银行家算法（4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10日 |
| 学生姓名 | 张旭君 | 学号 | 20210715B0602 | 专业班级 | | 2021软件工程 6 班 |
| 指导教师 | 喻学才 | | 实验成绩 | |  | |
| 一、实验目的  1.掌握有关资源申请、死锁等相关概念，进一步理解死锁产生的原因和必要条件。  2.掌握利用银行家算法避免死锁的方法，能结合具体应用分析系统状态。  二、实验内容及要求  本实验要求通过编写和调试一个系统动态分配资源的简单模拟程序，观察死锁产生的条件，并采用适当的算法，有效地防止和避免死锁的发生。具体要求如下：  （1）模拟一个银行家算法，判断是否处于安全状态；  （2）初始化时让系统拥有一定的资源；  （3）用键盘输入的方式申请资源；  （4）如果预分配后，系统处于安全状态，则修改系统的资源分配情况，判断其安全序列；  （5）如果预分配后，系统处于不安全状态，则提示不能满足请求；  三、实验源代码及说明  四、实验结果及分析  代码如下：  package caozuoxiotn.demo4;  import java.util.Scanner;  public class ResourceAllocationSimulation {  // 系统资源数量  private static final int SYSTEM\_RESOURCES = 5;  // 进程数量  private static final int PROCESSES = 3;  // 进程拥有的最大资源需求量  private static final int MAX\_RESOURCE\_NEED = 3;  // 系统资源向量  private static int[] systemResources;  // 每个进程已分配的资源矩阵  private static int[][] allocatedResources;  // 每个进程还需的资源矩阵  private static int[][] neededResources;  // 系统资源向量的副本，用于判断是否处于安全状态  private static int[] availableResources;  public static void main(String[] args) {  init();  while (true) {  displayStatus();  int processIndex = getProcessIndex();  int[] requestResources = getRequestResources();  allocateResources(processIndex, requestResources);  if (isSafeState()) {  System.out.println("资源分配成功！");  } else {  System.out.println("无法满足请求！");  }  }  }  // 初始化系统资源、进程已分配的资源和进程还需的资源  private static void init() {  systemResources = new int[SYSTEM\_RESOURCES];  allocatedResources = new int[PROCESSES][SYSTEM\_RESOURCES];  neededResources = new int[PROCESSES][SYSTEM\_RESOURCES];  availableResources = new int[SYSTEM\_RESOURCES];  // 初始化系统资源  for (int i = 0; i < systemResources.length; i++) {  systemResources[i] = (int) (Math.random() \* MAX\_RESOURCE\_NEED) + 1;  availableResources[i] = systemResources[i];  }  // 初始化进程已分配的资源和进程还需的资源  for (int i = 0; i < PROCESSES; i++) {  for (int j = 0; j < SYSTEM\_RESOURCES; j++) {  allocatedResources[i][j] = (int) (Math.random() \* systemResources[j]);  neededResources[i][j] = systemResources[j] - allocatedResources[i][j];  }  }  }  // 显示系统资源、进程已分配的资源和进程还需的资源  private static void displayStatus() {  System.out.println("系统资源向量：");  displayArray(systemResources);  System.out.println("进程已分配的资源矩阵：");  displayMatrix(allocatedResources);  System.out.println("进程还需的资源矩阵：");  displayMatrix(neededResources);  }  // 显示数组  private static void displayArray(int[] array) {  for (int i = 0; i < array.length; i++) {  System.out.print(array[i] + " ");  }  System.out.println();  }  // 显示矩阵  private static void displayMatrix(int[][] matrix) {  for (int i = 0; i < matrix.length; i++) {  for (int j = 0; j < matrix[i].length; j++) {  System.out.print(matrix[i][j] + " ");  }  System.out.println();  }  }  // 获取用户输入的进程编号  private static int getProcessIndex() {  Scanner scanner = new Scanner(System.in);  System.out.print("请输入进程编号（0-" + (PROCESSES - 1) + "）：");  int processIndex = scanner.nextInt();  return processIndex;  }  // 获取用户输入的请求资源数量  private static int[] getRequestResources() {  Scanner scanner = new Scanner(System.in);  int[] requestResources = new int[SYSTEM\_RESOURCES];  for (int i = 0; i < SYSTEM\_RESOURCES; i++) {  System.out.print("请输入请求的第" + (i + 1) + "个资源数量：");  requestResources[i] = scanner.nextInt();  }  return requestResources;  }  // 分配资源  private static void allocateResources(int processIndex, int[] requestResources) {  for (int i = 0; i < SYSTEM\_RESOURCES; i++) {  allocatedResources[processIndex][i] += requestResources[i];  neededResources[processIndex][i] -= requestResources[i];  availableResources[i] -= requestResources[i];  }  }  // 判断是否处于安全状态  private static boolean isSafeState() {  int[] work = availableResources.clone();  boolean[] finish = new boolean[PROCESSES];  // 初始化所有进程的状态为未完成  for (int i = 0; i < PROCESSES; i++) {  finish[i] = false;  }  // 找到一个满足以下条件的未完成的进程：  // (1) 所需的资源小于等于当前可用的资源  // (2) 进程未完成  for (int i = 0; i < PROCESSES; i++) {  if (!finish[i] && isResourceNeedLessEqual(work, neededResources[i])) {  for (int j = 0; j < SYSTEM\_RESOURCES; j++) {  work[j] += allocatedResources[i][j];  }  finish[i] = true;  i = -1; // 重新开始遍历未完成的进程  }  }  // 检查所有进程是否都已完成  for (boolean isFinished : finish) {  if (!isFinished) {  return false;  }  }  return true;  }  // 判断所需的资源是否小于等于当前可用的资源  private static boolean isResourceNeedLessEqual(int[] availableResources, int[] neededResources) {  for (int i = 0; i < availableResources.length; i++) {  if (neededResources[i] > availableResources[i]) {  return false;  }  }  return true;  }  }  运行效果图如下：        代码解析：  首先描述一下银行家算法是一种避免死锁的算法，主要用于操作系统中进程对资源的请求和分配。该算法的核心思想是：在系统运行过程中，对于每个进程，都需要检查其对资源的请求是否会导致死锁。如果不会，则允许该进程进行资源分配；否则，拒绝该进程的请求。根据上述代码可以模拟资源的分配过程，并使用银行家算法来判断系统是否处于安全状态。  1、定义一个名为ResourceAllocationSimulation的公共类。  2、定义一些常量，如系统资源数量、进程数量和每个进程拥有的最大资源需求量。  3、定义三个数组，分别表示系统资源向量、已分配的资源矩阵和还需的资源矩阵。  4、定义一个数组，表示系统资源向量的副本，用于判断是否处于安全状态。  5、定义主函数main，程序的入口。  6、调用init函数初始化系统资源、已分配的资源和还需的资源。  7、进入一个无限循环，不断进行资源分配和判断是否安全。  8、调用displayStatus函数显示当前系统资源、已分配的资源和还需的资源的状态。  9、调用getProcessIndex函数获取用户输入的进程编号。  10、调用getRequestResources函数获取用户输入的请求资源数量。  11、调用allocateResources函数根据用户输入的资源数量分配资源。  12、调用isSafeState函数判断系统是否处于安全状态。  13、如果系统处于安全状态，输出"资源分配成功！"；否则，输出"无法满足请求！"，结束循环。  14、定义init函数，用于初始化系统资源、已分配的资源和还需的资源。  15、初始化系统资源向量、已分配的资源矩阵和还需的资源矩阵。  16、初始化系统资源向量的副本。  17、初始化进程已分配的资源和进程还需的资源。  18、根据用户输入的资源数量分配资源，判断系统是否处于安全状态，并判断所需的资源是否小于等于当前可用的资源。 | | | | | | |