南京信息工程大学 操作系统实验报告

实验项目 银行家算法的实现与验证

|  |  |  |  |
| --- | --- | --- | --- |
| 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

|  |
| --- |
| 本实验的主要目标是实现银行家算法，确保在资源分配过程中系统能够避免进入不安全状态，从而防止死锁的发生。实验的主要步骤如下：  1. 初始化数据  Available：表示系统当前可用的资源数。  Max：表示每个进程所需的最大资源数。  Allocation：表示每个进程当前已分配的资源数。  Need：表示每个进程还需要的资源数。  代码中初始化了这些数据结构：  int available[] = new int[]{3, 3, 2};  int max[][] = new int[][]{  {7, 5, 3},  {3, 2, 2},  {9, 0, 2},  {2, 2, 2},  {4, 3, 3}  };  int allocation[][] = new int[][]{  {0, 1, 0},  {2, 0, 0},  {3, 0, 2},  {2, 1, 1},  {0, 0, 2}  };  int need[][] = new int[][]{  {7, 4, 3},  {1, 2, 2},  {6, 0, 0},  {0, 1, 1},  {4, 3, 1}  };  2. 展示数据  通过 showData() 方法输出系统的当前状态，包括每个进程的最大资源需求、已分配资源和还需要的资源。  3. 资源请求处理  使用 change() 方法处理用户输入的资源请求，首先判断请求是否合法，然后更新系统的 Available、Allocation 和 Need 数组，并调用 checkSafe() 方法验证系统是否处于安全状态。  4. 安全性检查  checkSafe() 方法实现了银行家算法的安全性检查流程，通过模拟资源分配过程，判断系统是否能够进入一个安全序列。如果找到一个安全序列，则系统处于安全状态；否则，系统进入不安全状态，需要回退。  boolean checkSafe(int a, int b, int c) {  int work[] = new int[3];  work[0] = a;  work[1] = b;  work[2] = c;  boolean finish[] = new boolean[5];  int i = 0;  while (i < 5) {  if (!finish[i] && need[i][0] <= work[0] && need[i][1] <= work[1] && need[i][2] <= work[2]) {  for (int m = 0; m < 3; m++)  work[m] += allocation[i][m];  finish[i] = true;  i = 0;  } else {  i++;  }  }  for (i = 0; i < 5; i++)  if (!finish[i])  return false;  return true;  } |

1. 实验结果

|  |
| --- |
| 1. 初始状态输出  程序运行后首先输出系统的初始状态：  进程号 Max Allocation Need  A B C A B C A B C  0 7 5 3 0 1 0 7 4 3  1 3 2 2 2 0 0 1 2 2  2 9 0 2 3 0 2 6 0 0  3 2 2 2 2 1 1 0 1 1  4 4 3 3 0 0 2 4 3 1  请输入要请求的进程号(0--4):  1  请输入请求的资源数目  A资源的数目: 1  B资源的数目: 0  C资源的数目: 2  能够安全分配 |

1. 实验遇到问题及解决

|  |
| --- |
| 1. 问题：初次实现时安全性检查出错  在编写 checkSafe() 方法时，初始版本的循环逻辑存在问题，导致不能正确识别安全序列。  解决方法  通过重新审视银行家算法的步骤，优化了循环条件和 work 数组的更新逻辑，确保每个未完成的进程在满足条件时都能分配资源。 |

1. 总结

|  |
| --- |
| 通过本次实验，我们实现了银行家算法，验证了其在资源分配中的应用。银行家算法是一种有效的避免死锁的策略，在实际的操作系统资源管理中具有重要意义。通过本次实验，我们深入理解了资源分配与安全性检查的过程，并掌握了如何通过编程实现复杂的算法逻辑。  本实验还帮助我们提升了编码能力，尤其是在处理用户输入和实现算法的关键步骤时，学会了如何调试和优化代码。未来可以尝试扩展银行家算法，支持更多类型的资源和更加复杂的进程调度场景。 |