南京信息工程大学 操作系统实验报告

实验项目 哲学家就餐问题

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

|  |
| --- |
| 一、实验目的  1、使用Java代码实现经典的哲学家就餐问题（Dining Philosophers Problem）。  2、研究多线程环境下的资源竞争与死锁问题。  二、实验内容与步骤  1.类与功能的设计  **1. Monitor 类**  这个类负责管理哲学家的状态和筷子的使用。  **1.1 构造函数 Monitor(deadlock dp)**   * **功能**：初始化哲学家状态和筷子的状态。 * **参数**：接收一个 deadlock 类型的对象，用于更新用户界面的状态。 * **实现**：   + 创建一个 philStates 数组，用于记录每个哲学家的状态（0-思考，1-饥饿，2-吃饭）。   + 创建一个 chopsticks 数组，用于记录每根筷子的使用状态（是否被占用）。   **1.2 pickupL(int i)**   * **功能**：试图拾起哲学家左边的筷子。 * **参数**：哲学家的索引 i。 * **实现**：   + 使用 synchronized 关键字确保线程安全。   + 检查左边筷子是否可用。如果不可用，调用 wait()，使线程等待。   + 将筷子的状态设为已占用，并更新哲学家的状态为“饥饿”。   **1.3 pickupR(int i)**   * **功能**：试图拾起哲学家右边的筷子。 * **参数**：哲学家的索引 i。 * **实现**：   + 类似于 pickupL，确保线程安全。   + 检查右边筷子是否可用。如果不可用，调用 wait()，使线程等待。   + 更新筷子的状态和哲学家的状态为“吃饭”。   **1.4 putdown(int i)**   * **功能**：放下哲学家手中的筷子。 * **参数**：哲学家的索引 i。 * **实现**：   + 将左右两根筷子的状态设为未占用。   + 更新哲学家的状态为“思考”。   + 调用 notify()，唤醒等待的哲学家线程。   **2. deadlock 类**  这个类负责创建用户界面，展示哲学家的状态以及控制哲学家的活动。  **2.1 构造函数 deadlock()**   * **功能**：初始化图形用户界面。 * **实现**：   + 加载哲学家的状态图标，并创建标签来显示每个哲学家的状态。   + 设置窗口大小和位置，并设置默认关闭操作。   + 初始化 Monitor 对象和哲学家线程数组。   + 创建菜单，添加控制哲学家开始和停止的功能，以及退出程序的选项。   **2.2 菜单项的事件处理**   * **功能**：为菜单项添加事件监听器，实现用户与界面的交互。 * **实现**：   + ON 菜单项：启动所有哲学家的线程。   + OFF 菜单项：中断所有哲学家的线程。   + EXIT 菜单项：退出程序。   + Readme 菜单项：弹出对话框显示说明信息。   **3. philosopher 类**  这个类代表每个哲学家，负责其行为逻辑。  **3.1 构造函数 philosopher(int i, Monitor m)**   * **功能**：初始化哲学家的索引和监视器对象。 * **参数**：哲学家的索引 i 和 Monitor 对象 m。   **3.2 run()**   * **功能**：重写 Thread 类的 run 方法，定义哲学家的行为。 * **实现**：   + 循环执行哲学家的状态。   + 尝试拾起左边和右边的筷子，并在每个动作后调用 Thread.sleep() 模拟哲学家思考和吃饭的时间。   + 期间可能会中断，因此需要处理 InterruptedException。 |

1. 实验结果

|  |
| --- |
| 2.结果演示  790e90938b52bee2693967265df99bc0  530f52edcc2292c47064673ce93c93a7  发现出现了死锁现象  具体流程：  **1. 启动哲学家线程**   * **ActionListener**：MenuItem mi1 中的 ActionListener 将被触发。这个监听器的 actionPerformed 方法会被调用。 * **线程启动**：在这个方法中，程序会循环遍历 phil 数组，该数组存储了所有哲学家的线程实例。在每次迭代中，会调用每个哲学家的 start() 方法，启动相应的线程。   **2. 哲学家的行为**  每个哲学家的线程启动后，会执行其 run() 方法中的逻辑。具体行为如下：   * **获取左侧筷子**：哲学家首先调用 pickupL(pid) 方法，试图获取左侧的筷子。如果该筷子已经被其他哲学家占用，哲学家将进入等待状态，直到筷子可用。 * **获取右侧筷子**：如果成功获取左侧筷子，哲学家接着调用 pickupR(pid) 方法，尝试获取右侧的筷子。同样地，如果右侧的筷子被占用，哲学家会再次等待。 * **吃饭**：一旦成功获取到左右两侧的筷子，哲学家就进入吃饭状态。这时，界面上的状态图标和文本会更新，显示哲学家正在吃饭。 * **放下筷子**：吃完饭后，哲学家调用 putdown(pid) 方法，放下左右两侧的筷子，状态更新为思考。 * **重复循环**：哲学家的线程会循环执行上述步骤，持续进行思考、饥饿和吃饭的状态转换。   **3. 界面更新**  在整个过程中，界面中的相关 JLabel 组件会被更新，以反映每个哲学家的当前状态，包括图标和文本。这种实时反馈能够让用户直观地看到每个哲学家的行为。 |

1. 实验遇到问题及解决

|  |
| --- |
| 在整个过程中，界面中的相关 JLabel 组件会被更新，以反映每个哲学家的当前状态，包括图标和文本。这种实时反馈能够让用户直观地看到每个哲学家的行为。 |

1. 总结

|  |
| --- |
| **哲学家就餐问题的挑战**  哲学家就餐问题是经典的并发编程问题，体现了资源共享与竞争的冲突。在实验中，观察到多个哲学家争夺有限的筷子，深刻认识到如何有效地管理共享资源是多线程编程中的一个重要课题。  **死锁与饥饿的识别**  实验中模拟的场景让我对死锁和饥饿现象有了直观的认识。当多个线程同时请求资源时，容易发生死锁，从而导致程序无法继续运行。此外，哲学家长时间无法获得筷子时会进入饥饿状态，这提醒我在设计程序时要考虑资源的公平分配。  此次实验让我将理论知识应用于实际代码中，深入理解了同步、互斥以及线程间的竞争条件。这不仅增强了我对多线程编程的理解，也让我意识到在设计并发程序时需要考虑的复杂性。 |