

**学生作业报告册**

**作业3报告**

|  |  |
| --- | --- |
| **学年学期：** | **2024-2025 学年 🗹春🞎秋学期** |
| **课程名称：** | **操作系统** |
| **学生学院：** | **国际学院** |
| **专业班级：** | **34082201** |
| **学生学号：** | **2022214961** |
| **学生姓名：** | **周明宇** |
| **联系电话：** | **13329148059** |

**重庆邮电大学教务处制**

|  |  |  |  |
| --- | --- | --- | --- |
| **课程名称** | **操作系统** | **课程编号** | **A2130330** |
| **作业名称** | **使用信号量的多线程编程** | | |
| **一、作业目的**   1. 掌握Linux环境下使用pthread库创建多线程程序的方法 2. 理解信号量(semaphore)的线程同步机制 3. 实现特定执行顺序的线程间同步控制 4. 分析多线程程序的执行流程和竞态条件   **二、作业内容**  修改提供的shopping.c程序，使用信号量实现以下同步要求：   1. "Salad"必须早于"Butter"打印 2. "Milk"必须早于"Apples"打印 3. 确保线程安全，避免竞态条件   **三、实现步骤**   1. **总体实现步骤**     图1系统流程图    图2 同步关系图   1. **全局信号量声明**   sem\_t sem\_salad, sem\_milk;  定义两个信号量变量，用于线程间同步：  **• sem\_salad**：控制Salad和Butter的打印顺序  **• sem\_milk**：控制Milk和Apples的打印顺序   1. **print\_produce函数实现**   void \***print\_produce**(void \*items)  {      char\*\* array = (char\*\*)items;    **printf**("got %s\n", array[0]); // 1. 先打印Salad  **sem\_post**(&sem\_salad);         // 2. 释放sem\_salad信号量    **sem\_wait**(&sem\_milk);          // 5. 等待sem\_milk信号量  **printf**("got %s\n", array[1]); // 6. 最后打印Apples        return NULL;  }   1. 立即打印Salad（无阻塞） 2. 通过**sem\_post**通知dairy线程可以打印Butter 3. 等待milk信号量确保Milk已打印 4. 最后打印Apples 5. **print\_dairy函数实现**   void \***print\_dairy**(void \*items)  {      char\*\* array = (char\*\*)items;    **sem\_wait**(&sem\_salad);         // 3. 等待sem\_salad信号量  **printf**("got %s\n", array[0]); // 4a. 打印Milk  **printf**("got %s\n", array[1]); // 4b. 打印Butter  **sem\_post**(&sem\_milk);          // 4c. 释放sem\_milk信号量        return NULL;  }   1. 首先等待Salad打印完成的信号 2. 连续打印Milk和Butter 3. 通过**sem\_post**通知produce线程可以打印Apples 4. **main函数实现**   int **main**()  {      // 初始化信号量（初始值为0）  **sem\_init**(&sem\_salad, 0, 0);  **sem\_init**(&sem\_milk, 0, 0);        // 商品数组      char \*produce[] = { "Salad", "Apples", NULL };      char \*dairy[] = { "Milk", "Butter", NULL };        // 创建线程      pthread\_t th1, th2;  **pthread\_create**(&th1, NULL, print\_produce, (void\*)produce);  **pthread\_create**(&th2, NULL, print\_dairy, (void\*)dairy);        // 等待线程结束  **pthread\_join**(th1, NULL);  **pthread\_join**(th2, NULL);        // 清理信号量资源  **sem\_destroy**(&sem\_salad);  **sem\_destroy**(&sem\_milk);        return 0;  }   1. 信号量初始化为0，表示初始时不可用 2. 创建两个线程分别处理produce和dairy商品 3. 使用**pthread\_join**确保主线程等待所有子线程完成 4. 最后销毁信号量释放资源   **四、分析与结果**  **① 编译运行**    图3 编译运行  **② 运行结果**    图4 结果  经过多次运行，结果中均保证Salad 先于出现Butter，Milk先于Apples出现。  **五、心得体会**  通过本次线程同步作业，我对多线程编程有了更深刻的认识。最初运行时出现的随机输出顺序让我直观理解了线程竞争的不可预测性，而通过信号量的引入，我不仅掌握了sem\_init、sem\_wait、sem\_post等关键函数的使用技巧，更重要的是领悟到同步机制的本质是线程间的约定与协作。实验过程中，当出现"Butter"意外先于"Salad"打印时，通过添加调试语句逐步排查，最终发现是信号量释放时机不当所致，这个调试经历让我深刻认识到多线程编程中时序控制的精密性。特别是在分析两种合法输出序列（Salad-Milk-Butter-Apples与Salad-Milk-Apples-Butter）时，我理解了同步约束下仍存在的合理并发多样性。这次实验不仅锻炼了我的代码能力，更培养了我对并发程序设计的系统性思维，意识到在资源竞争环境下，良好的同步机制就像交通信号灯，既能防止冲突又能保证效率，这为今后开发更复杂的并发系统打下了坚实基础。 | | | |
| **源代码**  /\* shopping.c - 修改后的版本 \*/  #include <stdio.h>  #include <unistd.h>  #include <pthread.h>  #include <semaphore.h>  // 声明两个信号量  sem\_t sem\_salad, sem\_milk;  void \***print\_produce**(void \*);  void \***print\_dairy**(void \*);  void \***print\_dairy**(void \*items)  {      char\*\* array = (char\*\*)items;        // 等待Salad先打印  **sem\_wait**(&sem\_salad);  **printf**("got %s\n", array[0]); // 打印Milk  **printf**("got %s\n", array[1]); // 打印Butter        // 释放信号量，允许Apples打印  **sem\_post**(&sem\_milk);      return NULL;  }  void \***print\_produce**(void \*items)  {      char\*\* array = (char\*\*)items;    **printf**("got %s\n", array[0]); // 打印Salad      // 释放信号量，允许Butter打印  **sem\_post**(&sem\_salad);        // 等待Milk先打印  **sem\_wait**(&sem\_milk);  **printf**("got %s\n", array[1]); // 打印Apples        return NULL;  }  int **main**()  {      // 初始化信号量  **sem\_init**(&sem\_salad, 0, 0);  **sem\_init**(&sem\_milk, 0, 0);        char \*produce[] = { "Salad", "Apples", NULL };      char \*dairy[] = { "Milk", "Butter", NULL };        pthread\_t th1, th2;  **pthread\_create**(&th1, NULL, print\_produce, (void\*)produce);  **pthread\_create**(&th2, NULL, print\_dairy, (void\*)dairy);    **pthread\_join**(th1, NULL);  **pthread\_join**(th2, NULL);        // 销毁信号量  **sem\_destroy**(&sem\_salad);  **sem\_destroy**(&sem\_milk);        return 0;  } | | | |