**题目5：进程/线程通信**

1. **设计目的与要求**

本实验旨在通过多线程编程解决阅览室有限资源调度问题。假设阅览室共有5个座位，多个读者需要进行注册操作填写个人信息以占用座位进行阅读，并在完成阅读后注销信息释放座位。通过实验，深入掌握信号量和互斥锁机制在资源管理和线程同步中的作用。实验要求使用信号量机制限制最多5个读者线程同时占用座位，实现资源合理分配；使用互斥锁保护共享数据，确保注册与注销操作的互斥性，避免数据竞争；实现读者线程的注册、阅读和注销流程，保证共享数据的正确性和一致性；记录座位状态和线程操作过程，便于分析资源管理的运行效果。

1. **设计内容**

实验的核心内容是模拟一个具有5个座位的阅览室，通过多线程实现多个读者的并发访问和操作。每个读者线程需要经过注册、阅读和注销三个阶段。注册阶段，线程填写个人身份信息（如手机号）并占用座位；阅读阶段，线程模拟一个固定的阅读过程；注销阶段，线程释放座位并清空个人信息。实验中需要通过信号量机制对座位的分配进行限制，保证最多只有5个线程能够同时阅读；通过互斥锁机制保护共享数据的操作，防止线程并发访问共享数据时出现竞争条件。

1. **设备与环境**

操作系统：Ubuntu 20.04

虚拟化软件：VMware Workstation 16 Pro

开发环境：Visual Studio Code

内核版本：5.6.8

1. **设计思想**

实验采用信号量和互斥锁结合的方式来解决资源管理问题。信号量主要用于控制阅览室座位的最大占用数量，保证最多只有5个线程能够同时进入阅览室进行阅读；互斥锁用于保护共享数据结构，防止多个线程在并发环境中对座位状态或注册信息的同时读写导致数据不一致。

1. **主要数据结构和流程**

主要数据结构：

|  |
| --- |
| *#define SEAT 5 // 定义座位数*  *int queue[SEAT]; // 存储电话号码的整型数组*  *pthread\_mutex\_t mutex; // 互斥锁，用于控制对共享资源的安全访问*  *sem\_t empty; // 表示空座位的数量的信号量*  *sem\_t full; // 表示已占用座位的数量的信号量*  *int count = 0; // 记录阅览室内的读者数量* |

实验流程：

1. 初始化数据结构，程序通过定义 SEAT 为5来设定阅览室的座位数。接着，初始化了一个整型数组 queue 用于记录每个座位上读者的电话号码，一个互斥锁 mutex 以保护对 queue 和读者计数 count 的访问，以及两个信号量 empty 和 full 来分别跟踪空座位和已占座位的数量。

|  |
| --- |
| *#define SEAT 5*  *int queue[SEAT];*  *pthread\_mutex\_t mutex;*  *sem\_t empty, full;*  *int count = 0;*  *sem\_init(&empty, 0, SEAT);*  *sem\_init(&full, 0, 0);*  *pthread\_mutex\_init(&mutex, NULL);* |

1. 创建了两个线程，分别负责模拟读者进入和离开阅览室的过程。intoreadroom 线程在有空座位（通过 empty 信号量表示）时执行，线程锁定互斥锁，提示用户输入电话号码，并将其记录在 queue 中，更新 count 并释放互斥锁，随后通过增加 full 信号量标记座位已被占用。

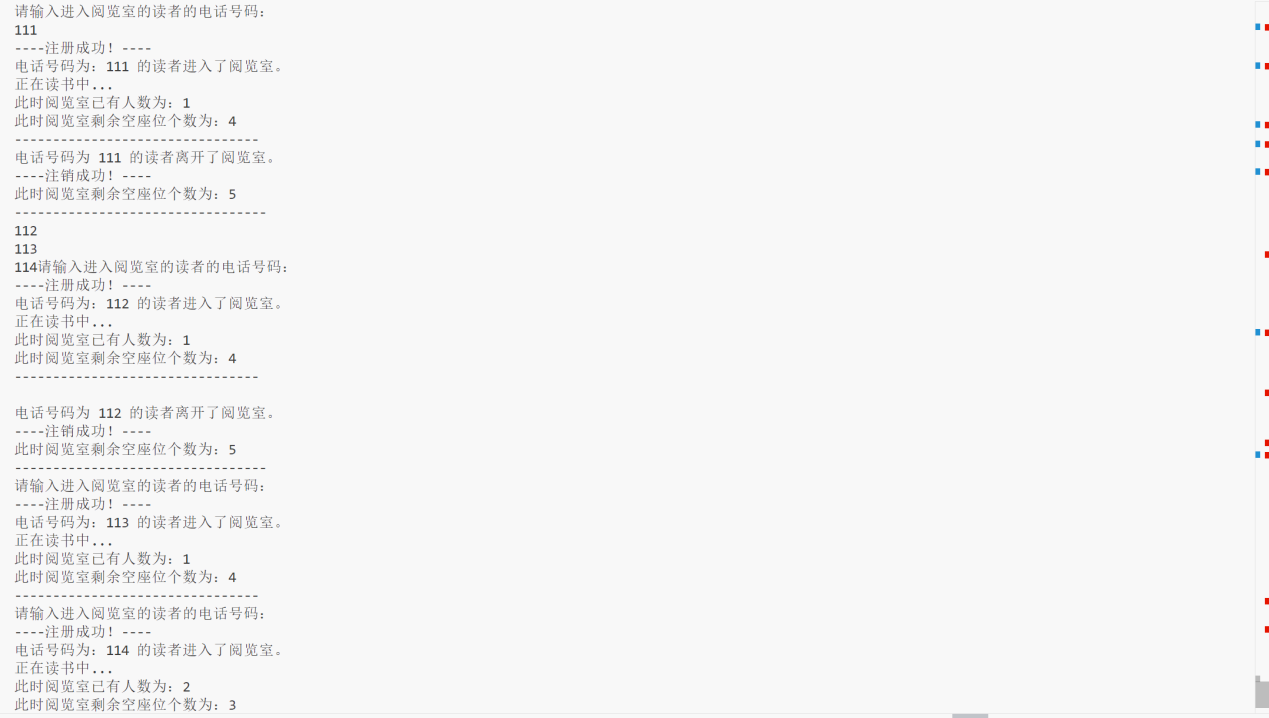
|  |
| --- |
| *void \*intoreadroom(void \*arg) {*  *int i = 0;*  *char input[100];*  *while (1) {*  *sem\_wait(&empty);*  *pthread\_mutex\_lock(&mutex);*  *fgets(input, sizeof(input), stdin);*  *queue[i] = atoi(input);*  *count++;*  *pthread\_mutex\_unlock(&mutex);*  *sem\_post(&full);*  *i = (i + 1) % SEAT;*  *}*  *}* |

1. leavereadroom 线程等待有读者占用的座位（通过 full 信号量表示），线程锁定互斥锁，清空对应座位的电话号码，更新 count，释放互斥锁，并通过 empty 信号量标记座位已空出。

|  |
| --- |
| *void \*leavereadroom(void \*arg) {*  *int i = 0;*  *while (1) {*  *sem\_wait(&full);*  *pthread\_mutex\_lock(&mutex);*  *queue[i] = 0;*  *count--;*  *pthread\_mutex\_unlock(&mutex);*  *sem\_post(&empty);*  *i = (i + 1) % SEAT;*  *}*  *}* |

1. 在主函数中，程序初始化信号量和互斥锁，创建两个线程，并等待这些线程结束。程序结束时，销毁创建的互斥锁和信号量，确保所有资源被正确释放。
2. 编译程序并执行
3. **实验测试结果及结果分析**

在实验中，创建了5个读者线程，模拟了一个并发访问阅览室的场景。以下是运行中的关键输出：



1.线程注册成功后，输出对应的座位号和身份信息。

2.当座位满时，新线程进入信号量等待状态，直到其他线程完成阅读并释放座位。

3.每个线程完成阅读后注销个人信息并释放座位，允许新的线程进入。

4.运行结果显示，每个线程在注册和注销阶段对共享数据的操作都得到了正确的保护，座位状态在多线程环境中保持一致。以下是部分运行输出的示例：