实验报告

东北林业大学

信息与计算机科学技术实验中心

|  |
| --- |
| 一、实验目的  **1. 掌握信号通信机制；**  **2. 使用信号量机制实现经典进程同步问题。** |
| **二、实验环境**  实验环境：VMware, Ubuntu 22.04.5 LTS(内核版本2.6.18)  硬件环境：内存2G以上，硬盘空间20G以上 |
| **三、实验内容及结果**  1. 模拟生产者-消费者问题  问题描述：  生产者向缓冲区添加数据，消费者从缓冲区中取出数据，通过信号量机制防止缓冲区溢出或空取。  设计思路：  使用两个信号量 full 和 empty 分别表示缓冲区已满和未满状态。通过互斥信号量 mutex 确保缓冲区操作的原子性。  代码实现：  void \*producer(void \*arg) {  int item;  while (1) {  item = rand();  sem\_wait(&empty);  pthread\_mutex\_lock(&mutex);  buffer[in] = item;  printf("Producer produced: %d\n", item);  in = (in + 1) % BUFFER\_SIZE;  pthread\_mutex\_unlock(&mutex);  sem\_post(&full);  sleep(1);  }  }  void \*consumer(void \*arg) {  int item;  while (1) {  sem\_wait(&full);  pthread\_mutex\_lock(&mutex);  item = buffer[out];  printf("Consumer consumed: %d\n", item);  out = (out + 1) % BUFFER\_SIZE;  pthread\_mutex\_unlock(&mutex);  sem\_post(&empty);  sleep(1);  }  }  代码编写  代码编译执行  2. 模拟哲学家就餐问题  问题描述：  五位哲学家围坐在餐桌周围，只有五根筷子，解决死锁和饥饿问题。  设计思路：  为每根筷子设置一个信号量，使用 P() 和 V() 操作模拟哲学家取筷子和放筷子。使用限制条件避免死锁，如哲学家只能同时取到两根筷子后再吃饭。  代码实现：  void \*philosopher(void \*arg) {  int id = \*(int \*)arg;  while (1) {  printf("Philosopher %d is thinking.\n", id);  sleep(1);  sem\_wait(&chopsticks[id]);  sem\_wait(&chopsticks[(id+1)%NUM\_PHILOSOPHERS]);  printf("Philosopher %d is eating.\n", id);  sleep(1);  sem\_post(&chopsticks[id]);  sem\_post(&chopsticks[(id+1)%NUM\_PHILOSOPHERS]);  printf("Philosopher %d finished eating.\n", id);  }  }  代码编写  代码编译执行 |
| **四、实验过程分析与讨论**  本次实验通过模拟生产者-消费者问题，深刻体会了多线程编程中同步机制的重要性，尤其是信号量和互斥锁的协作应用。在实验过程中，首先理解了 sem\_wait 和 sem\_post 操作的作用，它们分别用于资源的申请与释放，保证了线程在访问共享资源时的有序性。同时，互斥锁 mutex 的使用避免了多个线程同时访问共享缓冲区所可能引发的竞态条件，确保了数据一致性和程序的正确性。  empty 和 full 的初始值必须准确对应缓冲区的状态，错误的初始化可能导致线程永久阻塞或资源越界。：每次调用 sem\_wait 后都必须有对应的 sem\_post，否则线程可能无法释放资源，导致系统进入死锁状态。生产者和消费者分别使用 in 和 out 指针管理缓冲区操作，这些指针的更新必须在互斥锁保护下完成，否则可能因多个线程同时修改而出现指针错误，导致缓冲区数据丢失或混乱。  进程同步的核心目标是协调多个线程对共享资源的访问，既要避免资源争夺引发的不一致性，又要防止线程因等待资源而长时间阻塞。生产者和消费者问题中，如果信号量操作不当或缺少边界处理，容易导致死锁（如两个线程彼此等待资源释放）或饥饿（某线程长期无法获取资源）。实验中通过正确设计信号量和互斥锁避免了这些问题。  通过这次实验，我进一步巩固了多线程编程中常见同步机制的应用，并学会从实际问题中抽象出模型（如缓冲区的管理）。本实验不仅提升了编程技巧，更让我认识到细节处理在并发编程中的重要性，以及严格测试对确保代码健壮性的必要性。 |
| 五、指导教师意见  ☐ 实验完成内容完整，分析及讨论充分，写作规范  ☐ 实验完成内容基本完整，分析及讨论基本充分，写作基本规范  ☐ 实验完成内容不完整，分析及讨论不充分，写作不规范  指导教师签字：侯畅  **2024年12月17日** |