**《操作系统》实验报告**

实验序号：5　　　　　　　　 实验项目名称：线线程的同步和互斥

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学　　号 | 1910705127 | 姓　　名 | 宋宇昕 | 专业、班 | 软件一班 | |
| 实验地点 | 精工 | 指导教师 | 李远敏 | 实验时间 | 11.25 | |
| **一、实验目的及要求**  1. 加深对线程概念的理解，明确线程的控制。  2. 进一步认识线程的同步与互斥。  3. 学会利用pthread库，进行线程创建及同步与互斥  **二、实验设备（环境）及要求**  1.Ubuntu操作系统环境  2.搭建Ubuntu操作系统下的C语言编程环境。  **三、实验内容与步骤**  **1.利用互斥锁实现线程互斥：**  （1）利用互斥锁对两个子线程的共享变量进行互斥访问。  （2）编程要求：  a.定义一个用于互斥的互斥锁、和一个主函数和两个子线程都能访问的共享变量，一个主函数和两个用来创建子线程的子函数;在主函数中定义两个子线程ID的变量，初始化互斥锁，创建对应函数的子线程，等待两个子线程运行结束，获取并打印共享变量的结果，销毁互斥锁并返回退出。  b.在每个子线程对应的函数中，定义局部变量，先上锁，访问共享变量（三步法：将共享变量的值赋给局部变量，局部变量+1/-1, 将局部变量的值赋给全局变量），再解锁。  （3）程序修改：如果去掉子线程中的互斥控制，这时候程序的结果是并发，试通过sleep  （）控制子线程的不同并发过程，实现程序的不同结果（有三种）   1. **利用读写锁实现线程读写互斥：**   （1）使用vi将以下程序输入。  （2）利用gcc编译这个程序，写出编译命令和执行结果；如果不成功，尝试利用gdb调试。  （3）仔细阅读程序，并画出程序流程图  **3.利用线程信号量实现生产者和消费者之间的同步：**  具体的生产者和消费者问题：  有一个仓库生产者负责生产产品，并放入仓库，消费者从仓库拿走产品，要求仓库每次只能入一人，仓库中最多存放10个产品，仓库满时不能再放入产品，仓库空时不能再从中取出产品，生产消费速度不同  思路：生产和消费各一个线程，仓库为互斥，假设容量为10，库存为3假设生产速度比消费速度快，信号量的值等于剩余产品。  在以下程序中，已经给出主函数要求实现void \*produce(void \*arg)（）和void \*cost(void \*arg)（）两个函数并对整个程序进行调试运行。  在每一个函数中，如果工作条件满足情况下，先等待某个信号量，在进行线程各自的操作（产品数的变化，显示当前工作的产品数），再释放另一个信号量  **四、实验结果与数据处理**  **1.利用互斥锁实现线程互斥**  **（1）实验结果：**    图1：互斥锁实现进程互斥代码    图2：程序运行结果  **（2）结果分析：**  运行结果：我们可以看到会先打印出2，后打印出3。  原因：我们通过互斥锁，让第一个进程先对共享变量进行加1操作，然后轮到第二个进程来对共享变量进行加1操作，因为共享变量初始值为1，所以才能得到最后2、3的结果，这个实验体现了在线程实际运行过程中，我们经常需要多个线程保持同步，这时可以用互斥锁来完成任务。互斥锁的使用过程中，主要有pthread\_mutex\_init、pthread\_mutex\_destory、  pthread\_mutex\_lock、pthread\_mutex\_unlock这几个函数以完成锁的初始化，锁的销毁，上锁和释放锁操作。对共享资源的访问, 要对互斥量进行加锁, 如果互斥量已经上了锁, 调用线程会阻塞, 直到互斥量被解锁. 在完成了对共享资源的访问后,要对互斥量进行解锁。  **（3）程序修改：**  如果去掉子线程中的互斥控制，这时候程序的结果是并发，试通过sleep（）控制子线程的不同并发过程，实现程序的不同结果（有三种）    图3：第一种修改后的代码    图4：第一种修改后的代码运行结果    图5：第二种修改后的代码    图6：第二种修改后的代码运行结果    图7：第三种修改后的代码    图8：第三种修改后的代码运行结果  **总结：**  ①当我们在共享变量修改前加上sleep函数时，这样打印出来的数值都是原来的共享变量加1，原因是在共享变量修改之前，第一个进程试图修改共享变量的操作被阻塞了，这样导致马上进行的第二个进程修改值的操作和第一个进程基本一样，所以这样最后就让共享变量实际上只加了一次1，打印出来的结果就是两个2。  ②当我们在共享变量修改后加上sleep函数时，这样做的话，当第一个进程修改了共享变量令其加1，还没输出时，这时候进程一输出的操作被阻塞，然后进程二再对共享变量进行加1操作，所以这时候的共享变量一次都没被打印出来，但是却被实行了两次加1操作，当再回到两个进程的打印操作时，打印出的结果就是两个3。  ③当我们在共享变量打印后加上sleep函数时，这样做就和我们正常进行互斥锁的操作类似，打印输出的结果为2、3，这里的话就是进程一对共享变量进行加1操作，然后打印，再到进程二对共享变量进行加1操作，然后打印。  **2.利用读写锁实现线程读写互斥**  **（1）实验结果：**  **1.**输入示例代码调试并输出结果    图9：读写锁代码    图10：读写锁程序运行结果  qt_temp  图11：流程图   1. **总结：**   我们在编写多线程的时候，我们可能需要经常去读取某个共享数据变量，但是相对要改写这个变量的机会相对较少。在读的过程中，往往伴随着查找的操作，中间耗时很长，给这种代码加锁，会极大的降低我们程序的效率。所以提出了读写锁。读写锁具有写独占，读共享，写锁优先级高的特性。  一次只有一个线程可以占有写模式的读写锁, 但是可以有多个线程同时占有读模式的读写锁，正是因为这个特性，当读写锁是写加锁状态时，在这个锁被解锁之前, 所有试图对这个锁加锁的线程都会被阻塞。  当读写锁在读加锁状态时, 所有试图以读模式对它进行加锁的线程都可以得到访问权, 但是如果线程希望以写模式对此锁进行加锁, 它必须阻塞直到所有的线程释放锁。  通常，当读写锁处于读模式锁住状态时，如果有另外线程试图以写模式加锁，读写锁通常会阻塞随后的读模式锁请求, 这样可以避免读模式锁长期占用, 而等待的写模式锁请求长期阻塞。  读写锁适合于对数据结构的读次数比写次数多得多的情况。因为，读模式锁定时可以共享， 以写模式锁住时意味着独占, 所以读写锁又叫共享-独占锁。读写锁的规则：读读共享，读写排他，写写排他。  **3.利用线程信号量实现生产者和消费者之间的同步**  **（1）实验结果：**    图12：线程信号量实现生产者和消费者同步程序代码1    图14：线程信号量实现生产者和消费者同步程序运行结果1  **（2）结果分析：**  我们可以看到最后的produce和cost的值会因为我们设置的sleep函数的不同位置而被影响，当我们设置在cost函数中时，这时的消费线程慢于生产线程，所以就会让最后的剩余量几乎趋向于满，而当我们设置在produce函数中时，这时的消费线程快于生产线程，所以最后的剩余量几乎为零。  使用信号量的优点：信号量可以看作是进化版的互斥锁，由于互斥锁的粒度比较大，如果我们希望在多个线程间对某一对象的部分数据进行共享，使用互斥锁是没有办法实现的，只能将整个数据对象锁住。这样虽然达到了多线程操作共享数据时保证数据正确性的目的，却无形中导致线程的并发性下降。线程从并行执行，变成了串行执行。与直接使用单进程无异。信号量，是相对折中的一种处理方式，既能保证同步，数据不混乱，又能提高线程并发。   1. 分析与讨论   通过这一次的实验，我对线程的控制和管理有了更多的了解，也学会了如何线程的同步和互斥，也懂得了如何利用互斥锁实现线程互斥，同时也理解了利用读写锁实现线程读写互斥  和利用线程信号量实现生产者和消费者之间的同步。对于线程的整个运行过程和运行方式有了更加深入地体会，让我对linux操作系统的学习更加得心应手了。 | | | | | | |
| 六、教师评语  签名：  日期： | | | | | | 成绩 |