华南理工大学

《操作系统》课程实验报告

实验题目：实验二：线程同步和互斥

姓名： 梁馨元 学号： 201730613410

班级： 17计科1班 组别： 无

合作者： 无

指导教师： 钟竞辉

|  |
| --- |
| **实验概述** |
| 【实验目的及要求】  实验目的：  1. 线程的同步互斥  2. 利用Linux的库函数实现睡觉理发师问题和写者优先的读写问题。  实验要求：  1．用线程实现睡觉的理发师问题。（同步互斥采用信号量）理发 师问题的描述：一个理发店接待室有n张椅子，工作室有1张椅子；没有顾客时，理发师睡觉；第一个顾客来到时，必须将理 发师唤醒；顾客来时如果还有空座的话，他就坐在一个座位上等待；如果顾客来时没有空座位了，他就离开，不理发了；当理发师处理完所有顾客而又没有新顾客来时，他又开始睡觉。  2. 用线程实现读者写者问题。编写一个写者优先解决读者写者问题的程序，其中读者和写者均是多个线程，用信号量作为同步互斥机制。  3．提交源代码以及实验报告。  【实验环境】  操作系统：Linux |
| **实验内容** |
| 【实验过程】  一、理发师问题   1. 步骤和注意事项  * 本实验分别采用sem\_t和pthread\_mutex\_t & pthread\_cond\_t实现信号量的同步和互斥 * 互斥变量（也就是锁）用于检测椅子数是否不够，同步变量分别关心三个问题：(1)有顾客来了就要唤醒理发师、(2)理发师准备好了就要叫顾客来坐好剪发以及(3)理发师剪完要通知下一个顾客 * sem\_wait和sem\_post分别可以实现P操作和V操作，既可以实现互斥也可以实现同步 * pthread\_mutex\_t是为了实现互斥，pthread\_cond\_wait和pthread\_cond\_signal是为了实现同步，类似（但不等同）P和V操作      * sem\_trywait和sem\_wait的区别：前者如果得不到锁不会把线程block而是立刻返回一个错误值。这个特性恰好可以“监视”barber是不是睡着了XD      * pthread\_cond\_wait和pthread\_cond\_timedwait有点类似上面的一对，后者保证了线程安全，举例下面的代码：     假设所有的顾客都走了，没有线程signal(&customerArrive)，那pthread\_cond\_wait就一直不返回，这个while变成死循环，线程不能退出。所以这里一定要用pthread\_cond\_timedwait. 注意timedwait接口要加上relativetime，超过这个时间return     1. 实验结果和分析 2. /bin/barber (sem\_t实现)      1. /bin/barber\_1 (pthread\_con\_t & pthread\_mutex\_t实现)     上图可以看出customer和barber线程的状态。  图1用sem\_t实现的结果比较合理：每一个customer线程都是想arrived，然后began->haircut done或者leave（比如customer0剪完离开，customer1正在剪发的时候，customer7达到并离开，整合有5个线程“坐着”）；  图2用pthread\_cond\_t & pthread\_mutex\_t实现的结果就比较奇怪...比如出现”插队“，customer3->4->2依次到达，也按这个顺序依次began和haircut done. 按道理先后创建234线程不应该这样子。推测是这个实现方法的critical section范围比前者更大，线程互斥的时间更长，也就是抢占锁更加激烈，加大线程的创建时间间隔后好像就没问题了...  二、读者写者创建   1. 步骤和注意事项  * 本实验分别采用pthread\_mutex\_t和sem\_t实现互斥信号量 * 模拟共享状态，读者和写者共享的互斥变量shared\_var进行加减操作，读者数量=0时释放写锁，写者数量不为0的时候加读锁。另外在修改读者和写者数量时也要线程互斥 * 为了实现写者优先，除了上述的锁以外，还需要writePending锁，意思就是如果存在写者，那么读者（除了已经获得mutex1并在等待writeBlock的）都只能等待writePending（堵在第一行）。如果没有writePending锁，那么其他读者会被堵在等待readBlock，一旦没有写者，那其他读者就能立刻占有readBlock，下一个需要占有readBlock的写者就不能“优先”了      1. 实验结果和分析 2. /bin/reader\_writer     图1  （修改版）    图2    图3   1. /bin/reader\_writer\_1     图4  创建10个读者->10个写者->10个读者->10个写者，因为写者优先，所以即便第二轮先创建读者，还是先写者线程工作，符合“写者优先”的期望。  两个结果情况基本一致，读者运行时没有写者工作，反之亦然，写者也能依次对shared\_var++，证明线程互斥成功。  注：图1-3都是reader\_writer的结果，其中图2-3是详细版为了（减少行数还减少了一半的线程），但显然图1更简洁清晰明了；图4是reader\_writer\_1的结果 |
| **小结** |
| 本实验探索了线程的同步和互斥，经过一个下午磕磕巴巴的实验，终于完成了实验并同时留下了很多问题。  在写理发师问题的时候，就在思考为什么semaphor要同时糅合了互斥和同步两个功能？实际上，unix的确有其他接口分别处理线程的互斥和同步（pthread\_mutex\_t和pthread\_cond\_t,或者说monitor）。管程的接口从理解思路上来看，比信号量好理解，但写起来就发现...pthread\_cond\_t的使用可真的复杂太多了（所以只有第一个任务用了管程）  在这里总结一下这两套接口的区别：  从互斥的角度来看，虽然sem\_t初始值置1和pthread\_mutex\_t基本没有区别，但两者在实现上的区别还是要提一下的：sem\_t内有一个counter，可以用来显示有多少个线程在使用这个共同变量，而pthread\_mutex\_t只有0和1；  从同步的角度来看，pthread\_cond\_wait如果不能满足condition variable（同步条件）就会自动释放锁，然后在自动获取锁，直到成功获得返回。而sem\_wait如果cnt = 0不能返回就会一直阻塞进程，而且不会自动释放拥有的锁。从这个角度看，感觉pthread\_cond\_wait的效率更高一点，但这取决于系统对此的实现。  第二个任务虽然按照上面的分析，去掉z变量（writePending）就不能完美实现写者优先。但着实不能模拟出来这个“不完美”的情况，这也是很令人困惑的地方。  最后提一下，gdb在检查线程错误上功不可没，可以看见线程的创建和退出，也能看线程在哪里segfault。 |
| **指导教师评语及成绩** |
| 评语：  成绩：           指导教师签名：                                                 批阅日期： |