**西安电子科技大学**

**操作系统课程设计**

**(2020年度)**

**实**

**验**

**报**

**告**

**实验名称：** Priority-scheduling Task2

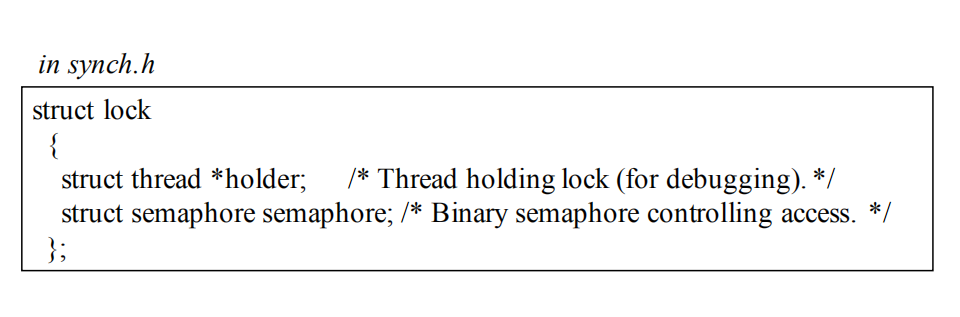
**班 级：** 1803011

**姓 名：** 张帅豪

**学 号：** 18030100101

**一、实验内容**

1. 解决由lock造成的优先级反转问题；



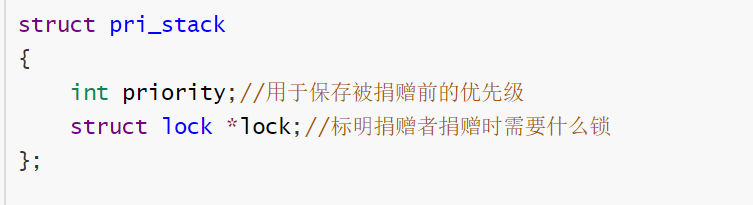
1. 在实验二的基础上，加入优先级捐赠机制，防止系统在高优先级线程所需资源被低优先级线程占有时出现优先级反转问题；
2. 通过本实验的完成，熟悉优先级捐赠算法，锻炼自己修改内核代码的能力。

**二、分析及设计**

1. 实现优先级捐赠问题，首先需要考虑保存原优先级的问题：因此需要定义一个包含原先优先级和被捐赠时捐赠者需要什么锁两个成员的结构体pri\_stack，并在thread结构体中加入类型为struct pri\_stack的成员；考虑到多重捐赠的情况，该成员是栈结构；
2. 在 thread 结构体中加入block\_sema、block\_lock两个成员，分别记录被哪个信号量阻塞、在等待哪个lock；
3. 修改 init\_thread()函数，使其在初始化时同时初始化这些新加入的成员；
4. 在调用 thread\_set\_priority()时，为了保证线程在捐赠完毕时其优先级回到新的优先级，在该函数中加入对线程是否被捐赠过的判断，若是，只修改栈底存储的优先级；否则，修改线程当前优先级，并执行第二次实验中加入的当前优先级确认；
5. 在 PV 操作中，当调用P操作时，若此时该信号量为0，应在线程的block\_sema成员中记录当前被哪个信号量阻塞；当调用 V 操作时，需在调用 thread\_unblock()后置空线程的block\_sema；
6. 在锁的acquire和release操作中，当线程发起对锁的索取时，需先判断该锁对应的信号量是否为0，若是，则说明已被占用，这时应在线程的block\_lock成员中保存阻塞该线程的锁，并调用优先级捐赠函数；
7. 而在释放锁的时候，如果该线程被捐赠过，会出现三种情况：该锁在线程被捐赠栈的栈顶，则还原优先级，删除记录；该锁既不在栈顶也不在栈底，直接删除记录；该锁在栈底，将栈底优先级赋值给其上一层后，删除记录；
8. 在实现优先级捐赠函数时，需要考虑被捐赠者是否在ready\_list中，若是，需调整其在 ready\_list 中的位置。同时，我们还需考虑被捐赠者是否在block\_list 中，若是，且被捐赠者也被某个锁阻塞，此时调整其在信号量等待队列中的位置，并触发递归捐赠。

**三、详细实现**

1. 在 thread.h 中加入结构体 pri\_stack，用于保存被捐赠前线程的优先级以及被捐赠时捐赠者所需的锁。



1. 修改 thread.h 中的 thread 结构体，为其加入保存步骤1中的数据的栈，同时加入两个成员block\_sema和block\_lock，分别用于存储该线程被哪个信号量阻塞和被哪个锁阻塞。

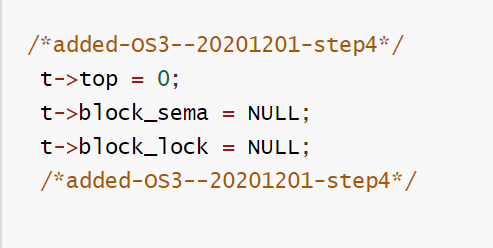


为了实现方便，这里采用了数组+额外一个int成员作为栈顶指针的方式来粗糙实现栈数据结构（本应使用Pintos内核中惯用的list方式）。考虑到 Pintos中只有64级优先级，故而此处数组只开了64的大小。

1. 在 thread.c 中修改 thread\_set\_priority()函数，使被设置优先级的线程在被捐赠过的情况下只修改栈底存储的优先级数值，而对于未被捐赠过的线程，该函数的功能与实验二中保持一致。



1. 在thread.c中修改init\_thread()函数，使每次初始化线程时，步骤2中新加入的成员也会被初始化。



其中，top 成员总指向栈最顶部元素的上一层（空层），若栈空，则为0。

1. 在 synch.c 中，修改 sema\_down()函数，使其在信号量为 0 时，将阻塞线程的信号量保存在线程的 block\_sema 成员中。



1. 在 synch.c 中修改sema\_up()函数，在调用thread\_unblock()后，将线程的block\_sema成员还原为NULL。



1. 在synch.c中修改lock\_acquire()函数，首先判断该锁的信号量是否为 0，若是，则说明该锁已被占用，此时将该锁保存在线程的block\_lock成员中，并调用优先级捐赠函数（该函数声明在thread.h中，定义在 thread.c中，具体将在步骤9中说明）。



1. 在synch.c中修改lock\_release()函数，当释放锁的线程被捐赠过时，删除所有与该锁有关的记录。





关于记录的删除，我们需对三种情况进行讨论：若该锁的记录在栈顶，还原该记录中保存的优先级，删除记录；若既不在栈顶也不在栈底，直接删除；若在栈底，将该记录保存的优先级赋值给 pri[1].priority，删除记录。

1. 在thread.h中声明、在thread.c中定义函数priority\_donate()，当索要lock的线程优先级高于占用lock的线程时，进行捐赠。

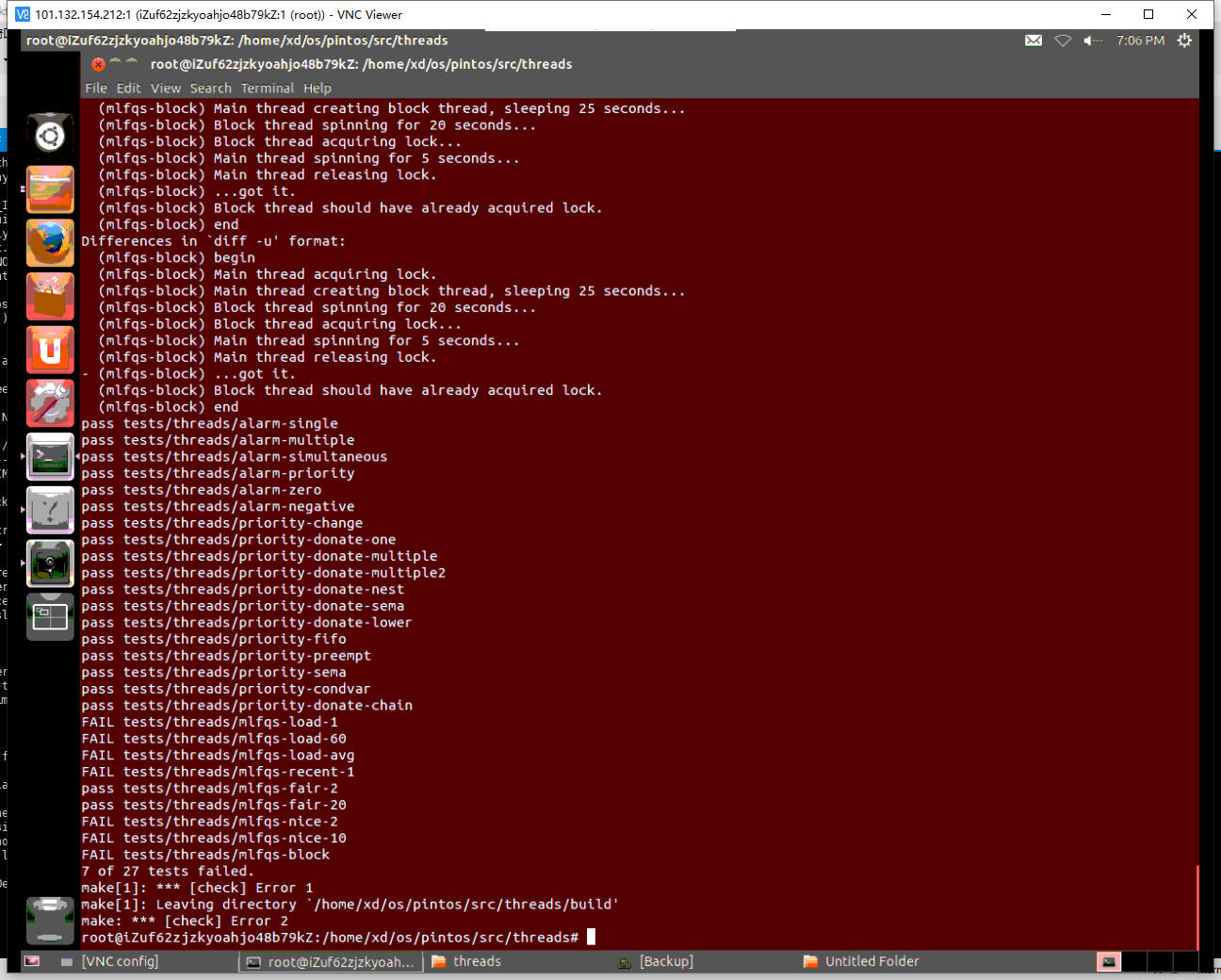
解释：在保存了当前优先级和捐赠者所需的 lock 后，栈顶指针（即 top）自增 1，占用 lock 的线程的优先级方被改变。在优先级改变后，我们需判断占有 lock 的线程是否在 ready\_list 中，若是，需按新的优先级调整其在 ready\_list 中的位置。并且，我们还需判断占有 lock 的线程是否处于阻塞队列中且存在一个阻塞了它的信号量，若是，我们需按新的优先级调整其在对应信号量的等待队列中的位置，同时若还存在一个阻塞该线程的锁，我们需要进行递归捐赠，使占有该锁的线程也被捐赠。



**四、实验结果**

在/home/xd/os/pintos/src/threads下执行make check指令，通过与正确结果中的pass项进行比对，发现结果一致，故实验成功。

具体运行结果截图如下：



**五、心得体会**

1. 通过本次实验，我了解到了什么是优先级反转问题以及遇到该问题的解决方法；
2. 优先级反转问指的是当系统在高优先级线程所需资源被低优先级线程占有、而低优先级线程的会因所需资源被中优先级线程占用而被阻塞时，中优先级线程会率先完成（率先完成的本应该是高优先级线程）的问题；
3. 本次实验，我对操作系统优先级反转问题等都有了更深刻的理解和学习，受益匪浅。