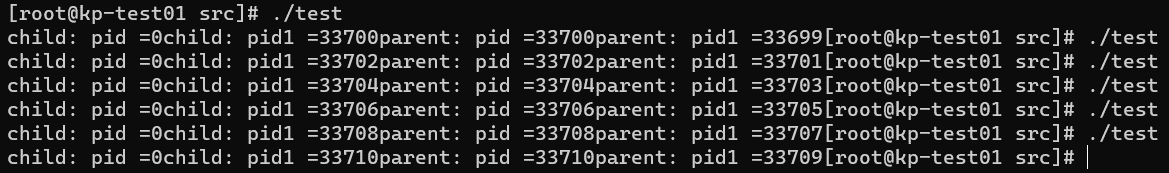
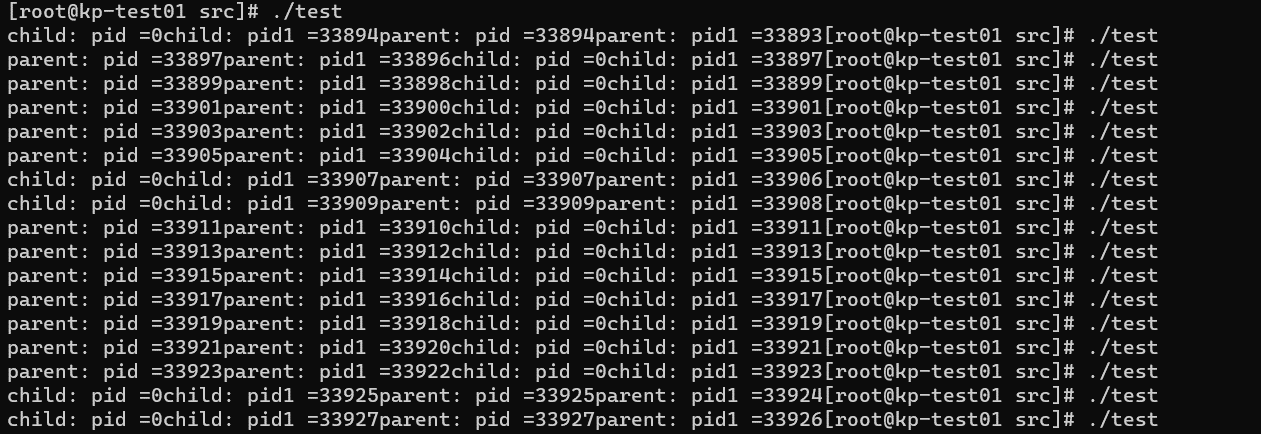
1.1进程

**步骤一：**



**步骤二：**

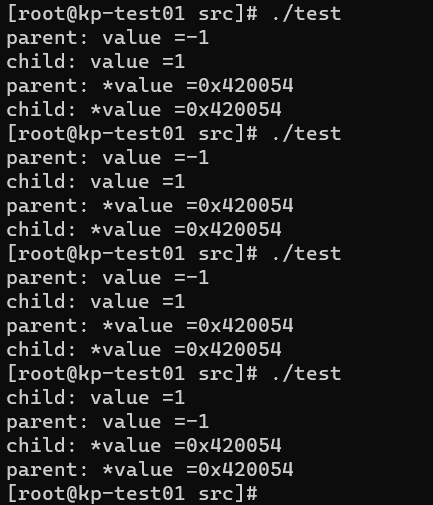
去掉wait后：



分析：父进程创建子进程后，父子进程进入if语句执行各自的代码，父进程和子进程的执行顺序没有先后之分，完全随机，所以在去掉wait后的多次运行结果中child和parent出现的先后顺序是随机的。

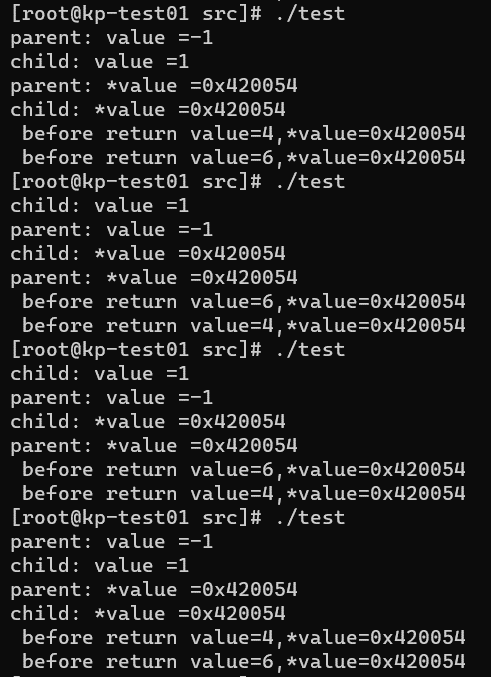
**步骤三：**

增加全局变量value：



在if语句的else部分设置value--,在else if(pid==0)部分设置value++,在这两部分的printf中都输出value的值和地址，随后运行程序，child中的value为1，parent中的value为-1,且child中value的地址和parent中value的地址相同，这是因为父进程在创建子进程时子进程完全复制了一份父进程的数据，输出value的地址时输出的其实是value在进程中的虚拟地址，因为是复制的缘故，两个value在父进程和子进程中的虚拟地址是一样的，但是映射到内存中的物理地址是不同的，所以在一个进程中修改value不会影响到另一个进程中的value

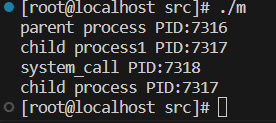
**步骤四：**



在return语句之前、if语句之后添加一段对全局变量value的操作代码，父进程和子进程在执行完if语句之后都会执行这一段代码，实现对value的操作后打印出各自的value值，因为父进程和子进程的执行顺序是完全随机的，所以child和parent出现的先后顺序也无法确定，如截图中第一次执行是chlid在前，第二次执行是parent在前，第三次执行是chlid在前

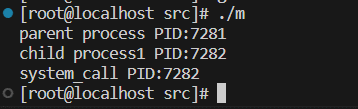
**步骤五：**

子进程调用system函数



子进程调用system函数时执行了system\_call文件，可以发现在执行system\_call文件时进程的pid和子进程的pid并不一样，这是因为system函数是在原进程上创建了一个新的进程。system函数执行完之后会返回原进程继续执行后续代码。

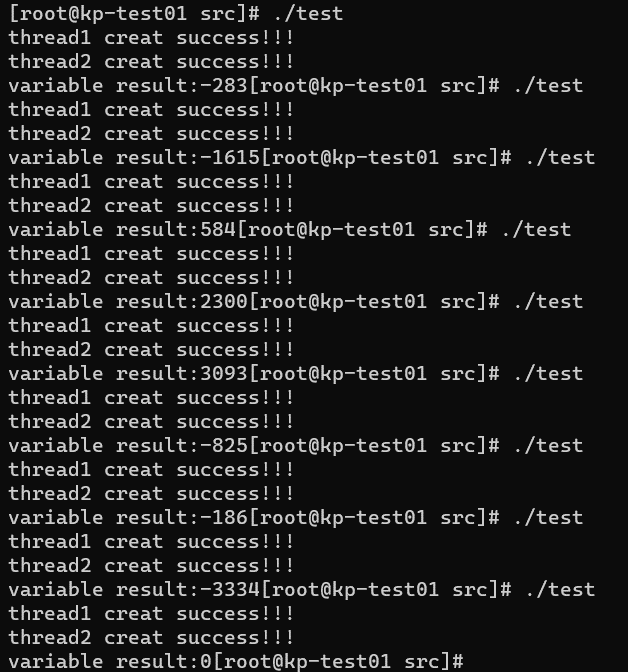
子进程调用execl函数



子进程调用execl函数时执行了system\_call文件，可以发现在执行system\_call文件时进程的pid和子进程的pid是一样的，并且执行完excel函数之后并没有后续输出，说明子进程在执行excel函数时并没有创建新进程，并且执行excel函数后并没有执行后续的printf代码，这是因为exec是用新进程(命令)覆盖了原有的进程

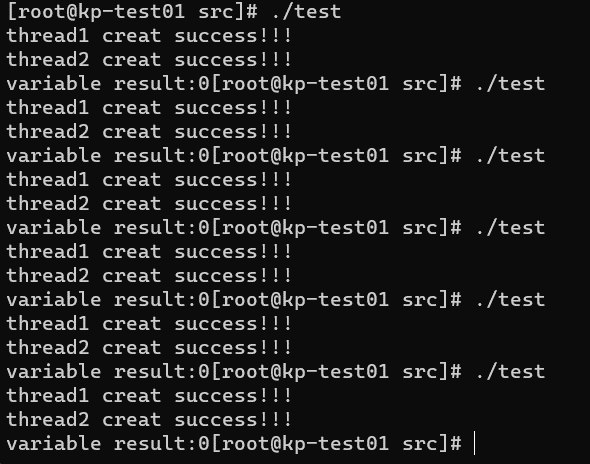
**1.2线程**

**步骤一：**



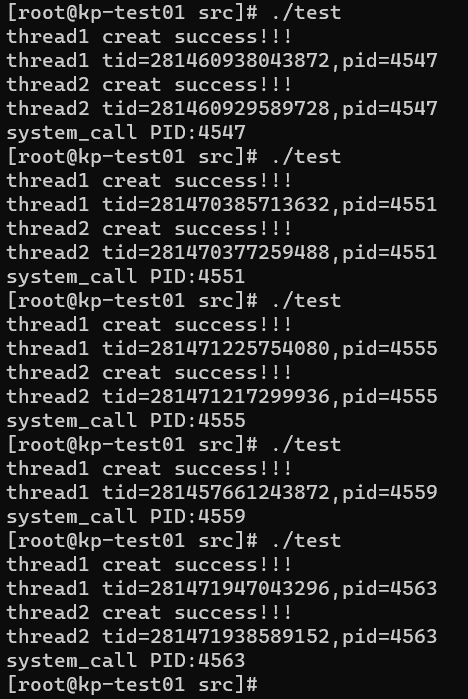
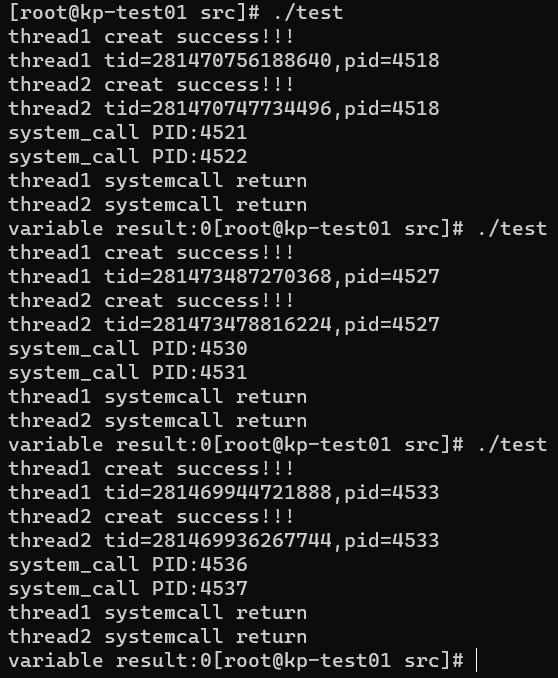
由于线程并发执行，存在竞态条件，即两个线程可能同时读取和修改共享变量。在没有适当的同步措施的情况下，不同线程的操作可能会交叉执行，导致结果不稳定，每次运行可能都会得到不同的结果。

步骤二：



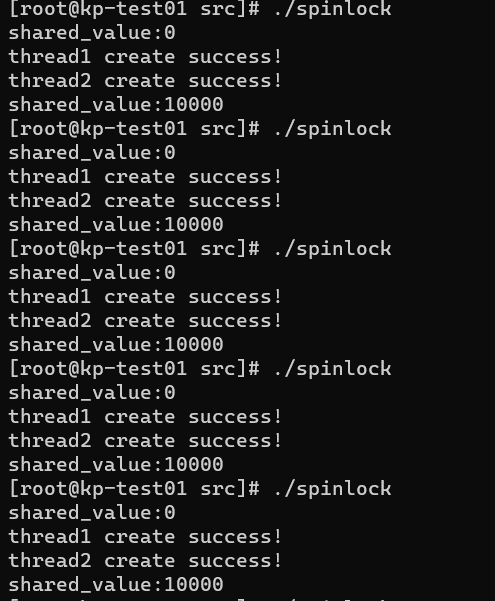
通过设置信号量以及对信号量的PV操作使得任意一个时刻最多只有一个线程能够对value进行修改，避免了多线程之间的竞争，不管重复多少次，value最终的值都是恒定为0

**步骤三：**



左图是有system函数的线程运行的结果，有图是有exec函数的线程运行的结果，可以发现线程执行system函数后会返回原线程并继续执行后续代码，而线程执行exec函数后直接结束，并不会继续执行后续代码，这是因为system函数是在原进程上开辟了一个新的进程，而exec函数是用新进程(命令)覆盖了原有的进程

**1.3****自旋锁**



在线程调用全局变量shared\_value的前后分别使用了自旋锁的获取函数和释放函数，且在调用这两个函数的时候不会被中断，以此来保证在任意一个时刻shared\_value最多只能被一个线程所修改，这就消除了多线程的竞争状态，实现了多线程的同步，最终shared\_value的值跟预期一样是10000.

2.困难：

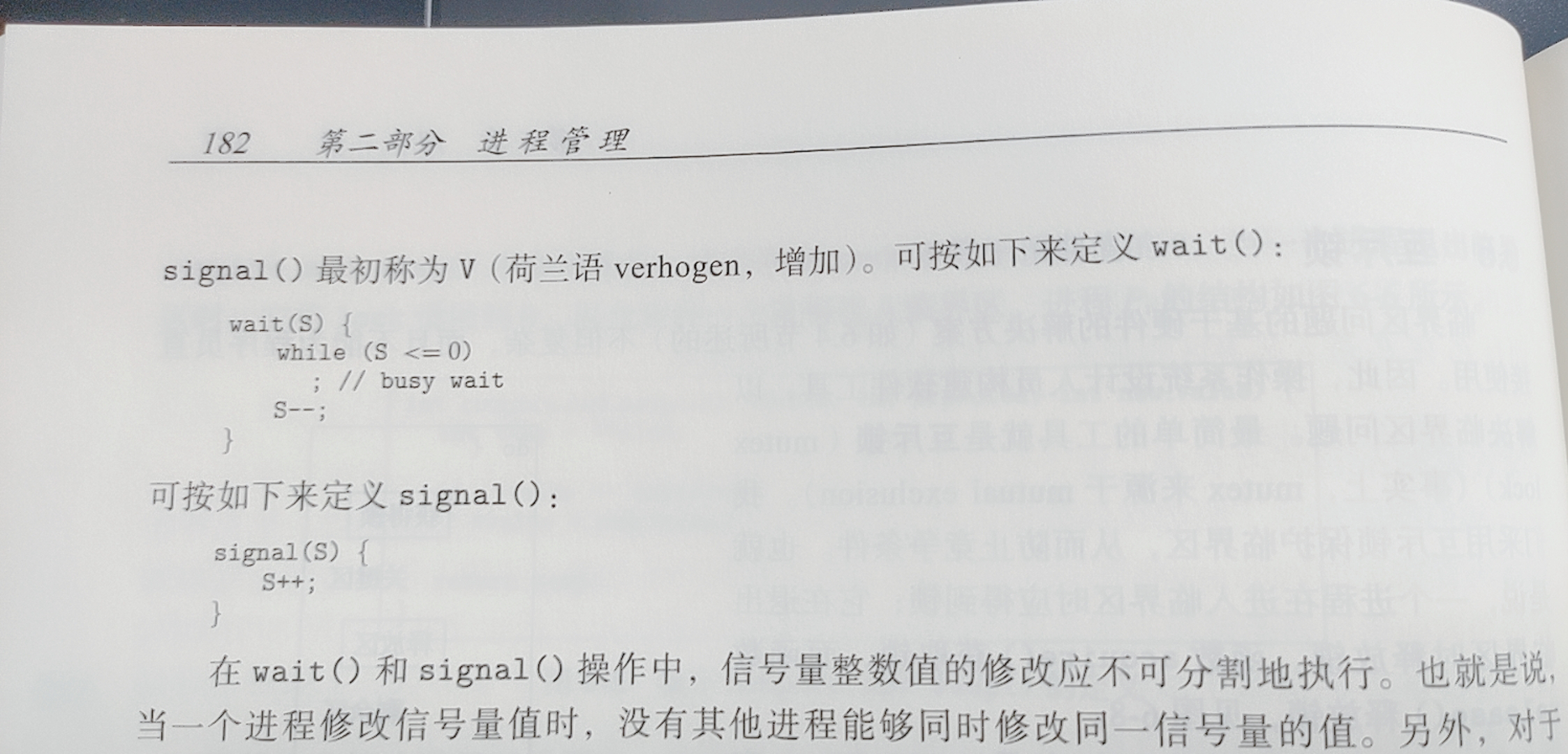
➀在遇到system和exce函数时一头雾水，在看到system的介绍 int system(const char \*\_\_command)时一直没弄明白这个command命令具体指代的是什么，后面看了具体实例后发现是需要执行文件所对应的目录

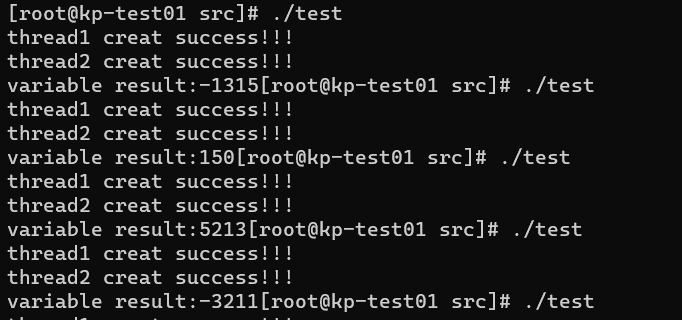
[Linux下使用system（）和execv（）实现对外部程序的调用\_linux采用system调用另一个进程-CSDN博客](https://blog.csdn.net/qq_25349629/article/details/78364247)

➁在完成实验一步骤三的过程中发现，父进程和子进程中全局变量value的地址是一样的，可是对应的value的值却不相同，当时感到很不解，因为在以前一直都认为地址相同对应的是同一个量，后面发现printf(“%p”,&value);打印的只是value在进程中的虚拟地址，而相同的虚拟地址在不同的进程中会映射到不同的物理地址上面，子进程的创建是复制了父进程的虚拟地址空间的

[父子进程能否共享全局变量\_牛客博客 (nowcoder.net)](https://blog.nowcoder.net/n/a6cdab4f10644fe18d85c9ad704b4638#:~:text=%E7%88%B6%E5%AD%90%E8%BF%9B%E7%A8%8B%E4%B8%8D%E8%83%BD%E5%85%B1%E4%BA%AB%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F%E3%80%82,%E4%BD%86%E6%98%AF%E5%A6%82%E6%9E%9C%E7%88%B6%E5%AD%90%E8%BF%9B%E7%A8%8B%E5%8F%AA%E6%98%AF%E5%AF%B9%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F%E5%81%9A%E8%AF%BB%E6%93%8D%E4%BD%9C%EF%BC%8C%E5%88%99%E7%88%B6%E5%AD%90%E8%BF%9B%E7%A8%8B%E5%9C%A8%E5%86%85%E5%AD%98%2A%2A%2A%E4%BA%AB%E5%90%8C%E4%B8%80%E4%BB%BD%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F%E3%80%82%20%E5%A6%82%E6%9E%9C%E7%88%B6%E5%AD%90%E8%BF%9B%E7%A8%8B%E4%B8%AD%E7%9A%84%E4%BB%BB%E4%BD%95%E4%B8%80%E4%B8%AA%E5%AF%B9%E5%8F%98%E9%87%8F%E5%81%9A%E4%BF%AE%E6%94%B9%E6%93%8D%E4%BD%9C%EF%BC%8C%E4%BC%9A%E5%9C%A8%E5%86%85%E5%AD%98%E4%B8%AD%E6%8B%B7%E8%B4%9D%E4%B8%80%E4%BB%BD%E5%89%AF%E6%9C%AC%EF%BC%8C%E7%84%B6%E5%90%8E%E5%9C%A8%E8%BF%99%E4%B8%AA%E5%89%AF%E6%9C%AC%E4%B8%8A%E8%BF%9B%E8%A1%8C%E4%BF%AE%E6%94%B9%EF%BC%8C%E4%BF%AE%E6%94%B9%E5%AE%8C%E6%88%90%E5%90%8E%E5%86%8D%E9%80%9A%E8%BF%87MMU%EF%BC%88%E5%86%85%E5%AD%98%E7%AE%A1%E7%90%86%E5%8D%95%E5%85%83%EF%BC%89%E6%98%A0%E5%B0%84%E5%9B%9E%E5%8E%BB%E3%80%82)

➂





在给多线程程序实现互斥锁的时候，我一开始想到的是课本6.6节有关互斥锁实现的内容，简单实现之后运行程序，发现结果还是和没有互斥锁时的结果一样，这代表我的互斥锁是不合格的，在查阅一些资料后发现我实现的这个互斥锁有可能在运行过程中被中断，于是我选用了教材6.9.4节Pthreads同步章节所介绍的unnamed semaphore作为信号量，最终成功解决问题。