**Locking**

1. 实验目的

探索中断与锁机制之间的一些交互和锁定

二、实验内容

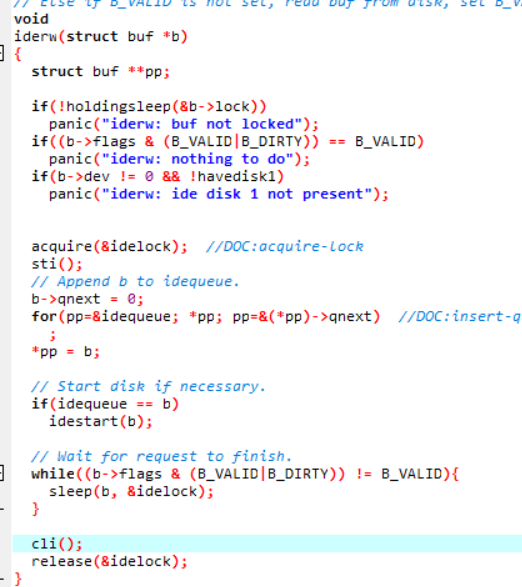
[Homework: xv6 locking (mit.edu)](https://pdos.csail.mit.edu/6.828/2018/homework/xv6-lock.html)

三、实验步骤与实验结论

第一部分：ide.c的中断

（1）确保中断关闭 在本地处理器上使用指令 （通过 ）， 并且中断保持关闭状态，直到该处理器持有的最后一个锁

（2）在 ide.c 中的 iderw 中，在 acquire（）之后添加对 sti（）的调用，并在 release()之前调用 cli()。



连续两次、在同一函数内进行acquire(),会出现panic,因为在 函数acquire()中：if(holding(lk)) panic("acquire");

出现panic的原因大概就是，在iderw()函数里面获取了锁之后，因为中断，转去执行中断处理函数，但是中断处理函数的相关

函数也需要锁，所以发生了错误

出现panic的原因大概就是，在iderw()函数里面获取了锁之后，因为中断，转去执行中断处理函数，但是中断处理函数的相关

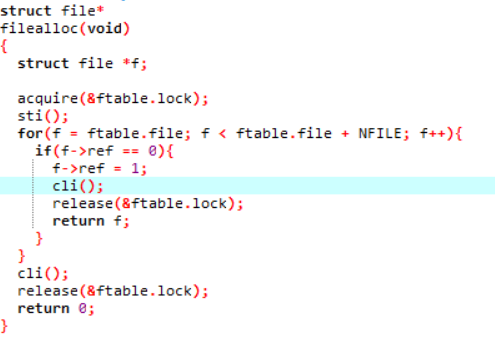
函数也需要锁，所以发生了错误

出现panic的原因大概就是，在iderw()函数里面获取了锁之后，因为中断，转去执行中断处理函数，但是中断处理函数的相关函数也需要锁，所以发生了错误

第二部分：flie.c的中断

（1）删除添加， 重建内核，并确保它再次工作

（2）在 file.c 中的 filealloc()中，在调用 acquire()之后添加对 sti()的调用，并在每个 release()之前添加一个 cli()。 还需要在其他#include 行之后的文件顶部添加#include "x86.h"。



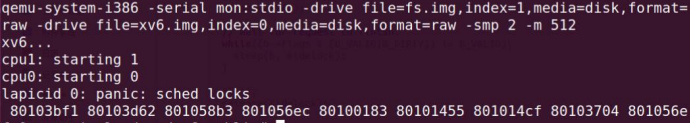


添加file.c之后不会发生错误，是因为中断处理函数里面不涉及file\_table\_lock 这个锁

实验：

（1）查看 acquire 的源码，发现对同一变量使用 acquire（）函数会进入 panic

（2）在启动第一个用户进程时，执行到 iderw()时(推测是在 sti()后，cli()前)发生 计时器中断，然后进行调度。



（3）不会panic，因为filealloc占用时间短，只有极小概率与其他读写文件发生冲突。



（4）在release函数中,为什么要把清除lk->pcs[0]和lk->cpu的工作放在释放锁之前而非释放锁之后?

如果考虑这样一种情况,如果先释放锁再做清除工作,线程一释放锁->线程二获取锁->线程二设置pcs[0],cpu->线程一清除pcs[0]和cpu。那么线程二的lk->pcs[0]和lk->cpu变量都没有被成设置应有的值,实际上lk->pcs[0]和lk->cpu也应看做是一种临界资源,在锁释放之前完成对它们的修改才能保证互斥访问。

四、反思总结

最开始对于实验二中下面那一行参数不知道代表什么意义，然后我仔细阅读了实验要求之后， 知道了那些参数代表的是机器语言。代表代码执行的顺序