

# 操作系统原理

## 实 验 报 告

学 院 智能与计算学部

年 级 2019 级

班 级 留学生班

学 号 6319000359

姓 名 张明君

2020 年 12 月 5 日

# 天津大学

## 操作系统原理实验报告

题目: xv6 locking

学院名称	智能与计算学部
专 业	计算机科学与技术
学生姓名	张明君
学 号	6319000359
年 级	2019 级
班 级	留学生班
时 间	2020. 12. 05

## 目 录

实验名称 .....	1
实验目的 .....	1
实验内容 .....	1
实验步骤与分析 .....	3
实验结论及心得体会 .....	11

# Homework: xv6 locking

## 1. 实验目的

-**Don't do this:** Explain in one sentence what happens.

-**Interrupts in ide.c:** Explain in a few sentences why the kernel panicked. You may find it useful to look up the stack trace (the sequence of %eip values printed by panic) in the kernel.asm listing.

-**Interrupts in file.c:** Explain in a few sentences why the kernel didn't panic. Why do `file_table_lock` and `ide_lock` have different behavior in this respect?

-**xv6 lock implementation:** Why does `release()` clear `lk->pcs[0]` and `lk->cpu` before clearing `lk->locked`? Why not wait until after?

## 2. 实验内容

### Don't do this

Make sure you understand what would happen if the xv6 kernel executed the following code snippet:

```
struct spinlock lk;
initlock(&lk, "test lock");
acquire(&lk);
acquire(&lk);
```

(Feel free to use QEMU to find out. `acquire` is in `spinlock.c`.)

### Interrupts in ide.c

An `acquire` ensures that interrupts are off on the local processor using the `cli` instruction (via `pushcli()`), and that interrupts remain off until the `release` of the last lock held by that processor (at which point they are enabled using `sti`).

Let's see what happens if we turn on interrupts while holding the `ide` lock.

In `iderw` in `ide.c`, add a call to `sti()` after the `acquire()`, and a call to `cli()` just before the `release()`. Rebuild the kernel and boot it in QEMU. Chances are the kernel will panic soon after boot; try booting QEMU a few times if it doesn't.

## Interrupts in file.c

Remove the `sti()` and `cli()` you added, rebuild the kernel, and make sure it works again.

Now let's see what happens if we turn on interrupts while holding the `file_table_lock`. This lock protects the table of file descriptors, which the kernel modifies when an application opens or closes a file. In `filealloc()` in `file.c`, add a call to `sti()` after the call to `acquire()`, and a `cli()` just before each of the `release()`s. You will also need to add `#include "x86.h"` at the top of the file after the other `#include` lines. Rebuild the kernel and boot it in QEMU. It most likely will not panic.

## xv6 lock implementation

Why does `release()` clear `lk->pcs[0]` and `lk->cpu` before clearing `lk->locked`? Why not wait until after?

### 3. 实验步骤和分析（要细化如何实现的思路或流程图）

#### Don't do this

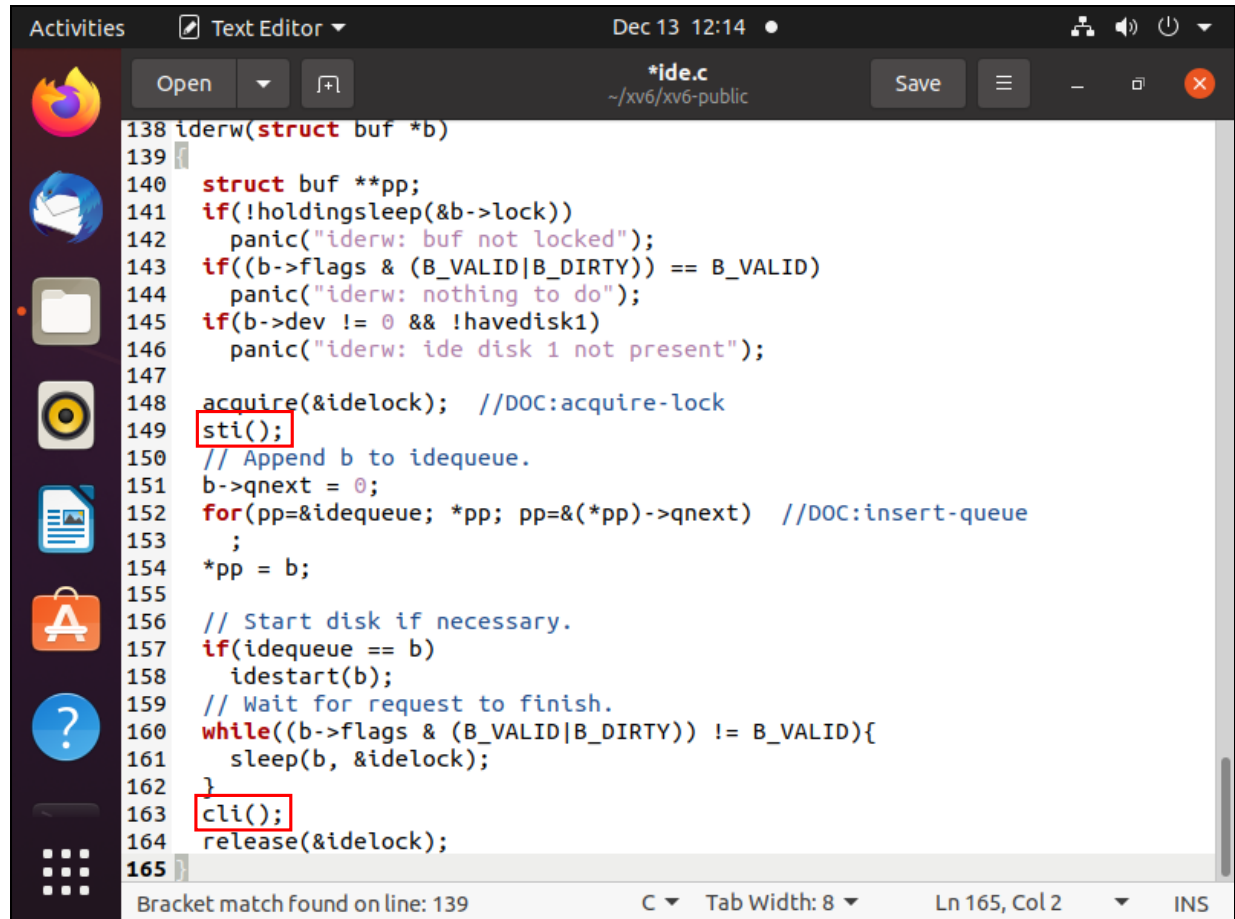
```
struct spinlock lk;
    initlock(&lk, "test lock");
    acquire(&lk);
    acquire(&lk);
```

First of all, take a look at the note of `acquire()` function acquire the lock. Loops (spins) until the lock is required. Therefore, if you do not apply for that lock, you will have to wait in a loop. Then, at the beginning of the `acquire()` function, there is a line of code if (holding (lk)) panic ("acquire"); therefore, applying for the same spinlock twice in succession will cause panic.

## Interrupts in ide.c

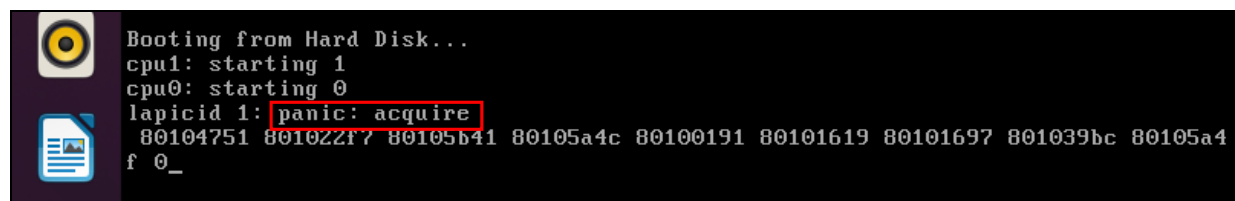
1/Let's see what happens if we turn on interrupts while holding the ide lock.

In `iderw` in `ide.c`, add a call to `sti()` after the `acquire()`, and a call to `cli()` just before the `release()`.



```
138 iderw(struct buf *b)
139 {
140     struct buf **pp;
141     if(!holdingsleep(&b->lock))
142         panic("iderw: buf not locked");
143     if((b->flags & (B_VALID|B_DIRTY)) == B_VALID)
144         panic("iderw: nothing to do");
145     if(b->dev != 0 && !havedisk1)
146         panic("iderw: ide disk 1 not present");
147
148     acquire(&idelock); //DOC:acquire-lock
149     sti();
150     // Append b to idequeue.
151     b->qnext = 0;
152     for(pp=&idequeue; *pp; pp=(*pp)->qnext) //DOC:insert-queue
153         ;
154     *pp = b;
155
156     // Start disk if necessary.
157     if(idequeue == b)
158         idestart(b);
159     // Wait for request to finish.
160     while((b->flags & (B_VALID|B_DIRTY)) != B_VALID){
161         sleep(b, &idelock);
162     }
163     cli();
164     release(&idelock);
165 }
```

After we changed the code in `ide.c` so now we have to rebuild the kernel and boot it in QEMU. Chances are the kernel will panic soon after boot; try booting QEMU a few times if it doesn't.



```
Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
lapicid 1: panic: acquire
80104751 801022f7 80105b41 80105a4c 80100191 80101619 80101697 801039bc 80105a4f 0_
```

Finally, we can see in the picture it's already panic and I tried to booting the qemu about 5 times to get that panic.

Now we can look up the stack trace (the sequence of %eip values printed by panic) in the `kernel.asm` listing.

```

8855    panic("acquire");
8856 80104744:    83 ec 0c          sub    $0xc,%esp
8857 80104747:    68 ed 79 10 80    push   $0x801079ed
8858 8010474c:    e8 3f bc ff ff    call   80100390 <panic>
8859 80104751:    8d b4 26 00 00 00 00 lea     0x0(%esi,%eiz,1),%esi
8860 80104758:    8d b4 26 00 00 00 00 lea     0x0(%esi,%eiz,1),%esi
8861 8010475f:    90                nop
8862
8863 80104760 <release>:
8864 {
8865 80104760:    f3 0f 1e fb      endbr32
8866 80104764:    55                push   %ebp
8867 80104765:    89 e5             mov     %esp,%ebp
8868 80104767:    53                push   %ebx
8869 80104768:    83 ec 10          sub     $0x10,%esp
8870 8010476b:    8b 5d 08          mov     0x8(%ebp),%ebx
8871    if(!holding(lk))
8872 8010476e:    53                push   %ebx
8873 8010476f:    e8 dc fe ff ff    call   80104650 <holding>
8874 80104774:    83 c4 10          add     $0x10,%esp
8875 80104777:    85 c0             test    %eax,%eax
8876 80104779:    74 22             je      8010479d <release+0x3d>
8877    lk->pcs[0] = 0;
8878 8010477b:    c7 43 0c 00 00 00 00 movl    $0x0,0xc(%ebx)
8879    lk->cpu = 0;

```

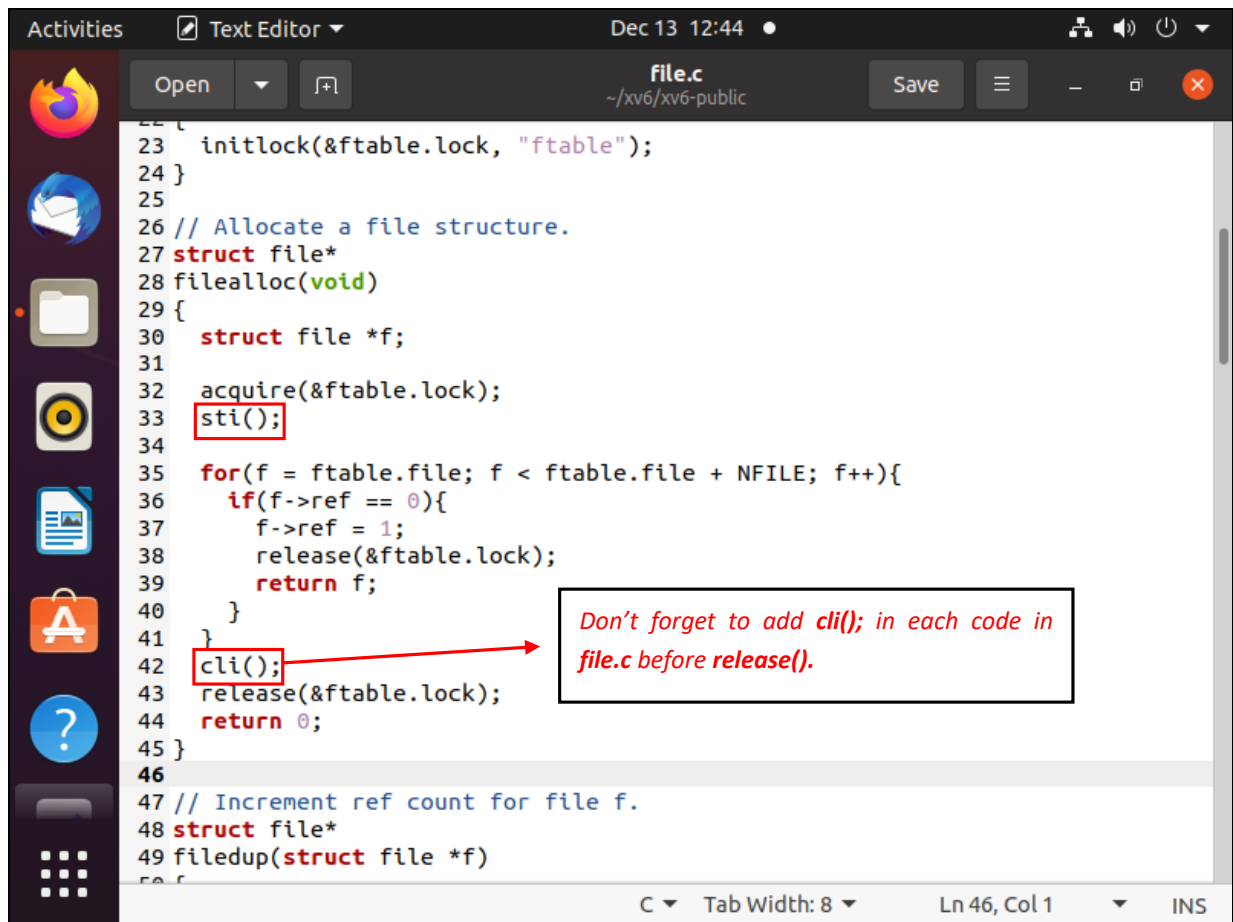
### why the kernel panicked?

At First, We have to look at the `ide.c` file. So we can see that both `iderw()` and `identr()` call `acquire()` for the same lock so after one acquire closes the interrupt, a process switch occurs. The other acquire executes and obtains the optional lock. After that, the interrupt is opened again. The interrupt handler returns to the original state and starts to try to obtain the optional lock. At this time, the panic is generated.

## Interrupts in file.c

At this point we have to remove the `sti()` and `cli()` that we have added in the `ide.c` file, Then rebuild the kernel, and make sure it works again.

Now we have to take a look at `file.c` file. We have to add a call to `sti()` after the call to `acquire()`, and a `cli()` just before `each` of the `release()`es.



```
23 initlock(&ftable.lock, "ftable");
24 }
25
26 // Allocate a file structure.
27 struct file*
28 filealloc(void)
29 {
30     struct file *f;
31
32     acquire(&ftable.lock);
33     sti();
34
35     for(f = ftable.file; f < ftable.file + NFILE; f++){
36         if(f->ref == 0){
37             f->ref = 1;
38             release(&ftable.lock);
39             return f;
40         }
41     }
42     cli();
43     release(&ftable.lock);
44     return 0;
45 }
46
47 // Increment ref count for file f.
48 struct file*
49 fileup(struct file *f)
50 {
```

Don't forget to add `cli();` in each code in `file.c` before `release()`.

One more thing is that we also need to add `#include "x86.h"` at the top of the file after the other `#include` lines. After that we have to re-boot the qemu and it will not panic. And `file_table_lock` and `ide_lock` have different behavior in this respect because compared with `ide_lock`, the number of times and time for kernel to open or close files are very small. It is basically impossible to read and write two files at the same time, so the possibility of conflict is very small.

## xv6 lock implementation

We do `release()` clear `lk->pcs[0]` and `lk->cpu` before clearing `lk->locked` is because if we releasing the lock, it will happen that:

- 1/Lock release, `lk->pcs[0]` and `lk->cpu` are not cleared
  - 2/Another CPU attempts to acquire the lock and succeeds. Set `lk->pcs[0]` and `lk->cpu`
  - 3/The current CPU clears `lk->pcs[0]` and `lk->cpu`
- So this results in incorrect lock information.



## 4. 实验结论与心得体会

In this experiment in the first part we have to explained why it' ll be panic and the second part is u have to make it panic by adding some code in to file that I have explained above and it' s not that easy to see the panic ,I have to run it many times to get the panic and there' re two different panic one is **panic acquire** and another one is **panic sched lock** and my pc it' s show the panic acquire and I already explained. After that, In the interrupt file.c part we have to change the code to default and make it run again without panic.