William Tran

2018/07/20

Homework (7) Report

For the code snippet below:

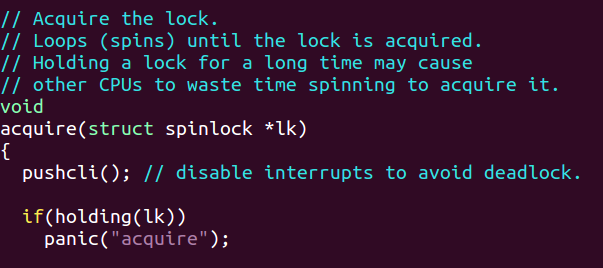
struct spinlock lk;

initlock(&lk, "test lock");

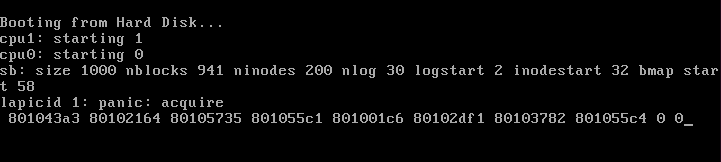
acquire(&lk);

acquire(&lk);

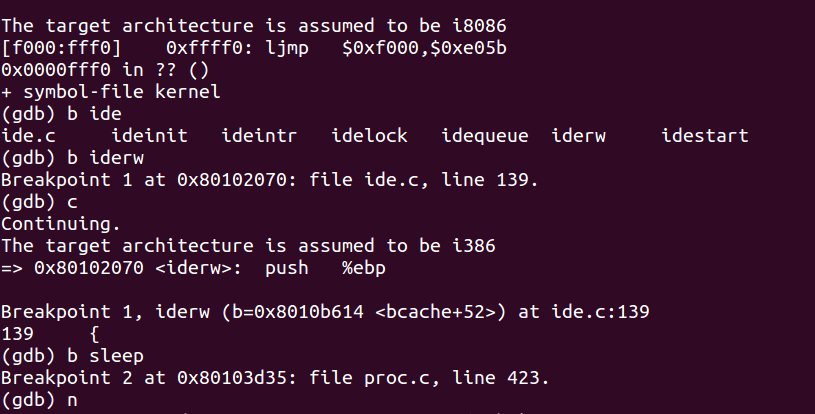
It acquires lock twice, so after the first acquire, the *acquire* function in spinlock.c panic. This panic happens because the second *acquire* detects that the CPU is occupying the lock. Therefore, it panics to prevent a deadlock. The code snippet below shows the if condition in *acquire* that panic when it detects that the CPU is holding a lock.



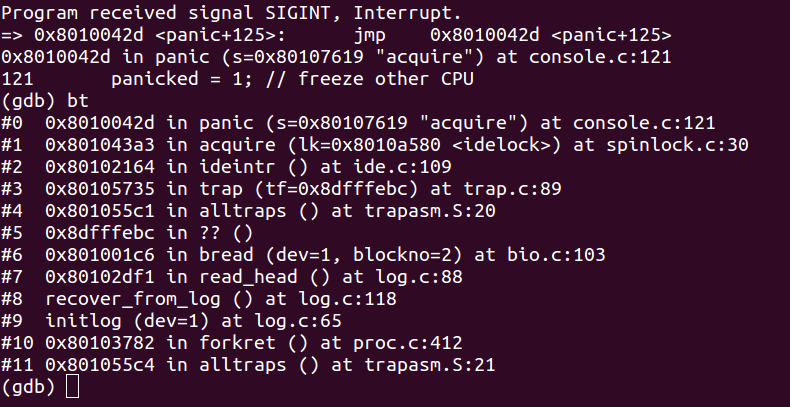
When adding sti() and cli() to iderw in ide.c according to the directions, the resulting panic is received in the kernel console:



To find the stack trace below the panic (as shown in the picture above), I used gdb to, firstly, set a breakpoint to the start of the function *iderw*. Below is the start of my gdb process.



Then, I continued to go through the function until I found an interrupt signal, which is the panic. I then back trace to reveal the stack trace and the locations of the addresses in the stack. The command *bt* below is the back trace.



Result: the addresses from line #1 (0x801043a3) to line #11 (0x801055c4) matches the addresses printed in the console (or at least most of them), and they point to specific locations in the code that are used.

**Explanation for panic:**

Since locks are designed to assure that threads don’t run concurrently without control, and that only 1 CPU at a time can hold the lock. In other words, locks are used to avoid races between CPUs. In this situation, sti() allows the second thread to run concurrently with the thread holding the lock. An interrupt (line #3) tries to call ideintr (line #2), which acquires idelock (line #1) on line 109 of ide.c. idelock here is already occupied by the sleep(b, &idelock) function in iderw function in ide.c, line 164. Since sleep already acquires the lock, the acquire function calls panic.

**Interrupt in file.c:**

After adding sti() and cli() in the designated locations, the kernel indeed did not panic. This is because the filealloc(void) function does not call sleep. Also, the ftable.lock is short enough that it does not trigger the clock interrupt. Therefore, it does not panic.

**xv6 lock implementation:**

If *release()* cleared *lk->pcs[0]* and *lk->cpu* **after** clearing *lk->locked*, then there would be a small gap. If the clock is interrupted in this gap, the lock is acquired, and *lk->pcs[0]* and *lk->cpu* will be set. Then, next time when lk->locked is called, there will be an error.