Operating System Design & Implementation

Lab5: Kernel Lock

TA:陳勇旗

Objective:

In this lab you can learn

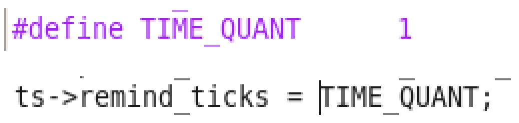
- The simple kernel lock mechanism
- Interruptible kernel
- Optimize your lab4 task scheduler

1. Preparation: Modify your lab4 code

In this lab, we want you do some modification based on lab4.

1.1. Modify task's scheduling time quantum

In order to improve kernel's response time, we need let OS do scheduling each timer tick occurs.



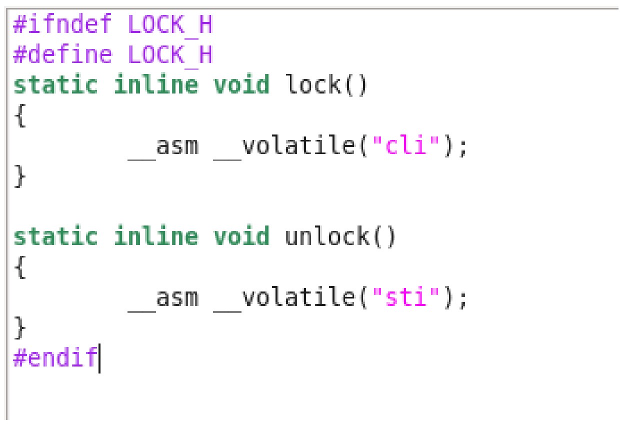
```
#define TIME_QUANT      1

ts->remind_ticks = TIME_QUANT;
```

1.2. Add lock.h

When CPU trap into kernel mode, the OS usually use a lock to protect some kernel variable or critical section.

In this lab, we implement a simple kernel lock by disabling interrupt.



```
#ifndef LOCK_H
#define LOCK_H
static inline void lock()
{
        __asm __volatile("cli");
}

static inline void unlock()
{
        __asm __volatile("sti");
}
#endif
```

1.3. Modify your trap_dispatch function

```
static void
trap_dispatch(struct Trapframe *tf)
{
        // Handle spurious interrupts
        // The hardware sometimes raises these because of noise on the
        // IRQ line or other reasons. We don't care.
        if (tf->tf_trapno == IRQ_OFFSET + IRQ_SPURIOUS) {
                printk("Spurious interrupt on irq 7\n");
                print_trapframe(tf);
                return;
        }

        //lock(); //(1)
        /* Lab3: Check the trap number and call the interrupt handler. */
        if (trap_hnd[tf->tf_trapno] != NULL)
        {
                extern Task *cur_task;

                if ((tf->tf_cs & 3) == 3)
                {
                        // Trapped from user mode.

                        cur_task->tf = *tf;
                        tf = &(cur_task->tf);
                }
                // Do ISR
                trap_hnd[tf->tf_trapno](tf);

                //lock(); //(2)
                env_pop_tf(tf);

                return;
        }

        // Unexpected trap: The user process or the kernel has a bug.
        print_trapframe(tf);
        panic("Unexpected trap!");

}
```

1.4. Add a function task2() in main.c, then child run *task2()* function and parent run shell().

```
void task2()
{
        int i, pid;
        i= getpid();
        while(1)
        {
                if ( (pid=fork()) > 0)
                {
                        cprintf("Im %d, bye!\n", pid);
                        kill_self();
                }
        }
}
```

2. Questions

After you modified above code, please answer these question.

2.1. About the locker

Why we don’t need to **unlock()** before **env_pop_tf()** and the OS still works fine?

2.2. Lock all kernel section

Uncomment (1) and check whether it still works fine.

If it works fine, please explain why it works.

If not, please debug it by gdb, and explain why it crashes.

2.3. Lock partial kernel section

Uncomment (2) lock, and comment (1) lock.

Check the new result and explain why it behaves like that.

2.4. Lock shared kernel variable

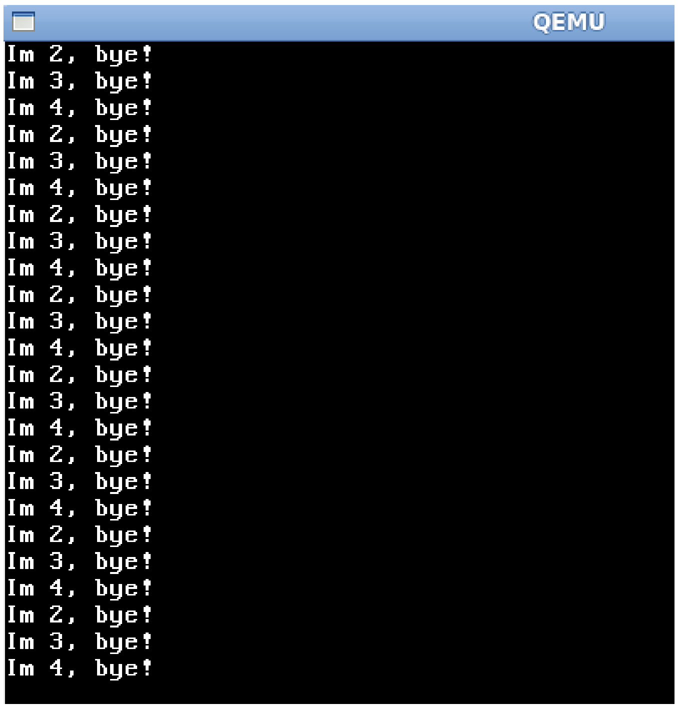
In OS kernel, there are many variables which are shared by tasks (ex. Tasks[], *cur_task, etc). Due to the unpredictable interrupt/trap happens, these share data may be polluted. In this part, you need to find up the potential share data and add lock mechanism to ensure them are well protected. Besides, critical section will slow down our kernel, please modify your kernel code (system call and scheduler) to use as less critical section as possible.

Hint: Each tasks shared same kernel stack, add lock before use this stack.

3. Demo and lab report

In lab5, you are required to demo and hand in a report.

During demo, you need show the result to TAs and they will ask you some question form 2.1~2.4,your result will looks like this.(To simulate the environment that tasks enter into kernel much frequently, we write a loop fork function (just print a string and then kill itself). If your OS can handle this, you can see the following result.(NR_TASK=5, pid=0 is shell, pid=1 is task2)



```
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
Im 2, bye!
Im 3, bye!
Im 4, bye!
```

In report, you need to answer the questions (2.1~2.4), write down what you learned in lab1~lab5 and give some suggestion about the lab content and what you want to learn in feature OSDI course.

Note: The report's deadline will be 4/14 23:59:59. Please check e3 system.