

# Lab 3 实验报告

181250068 李淳

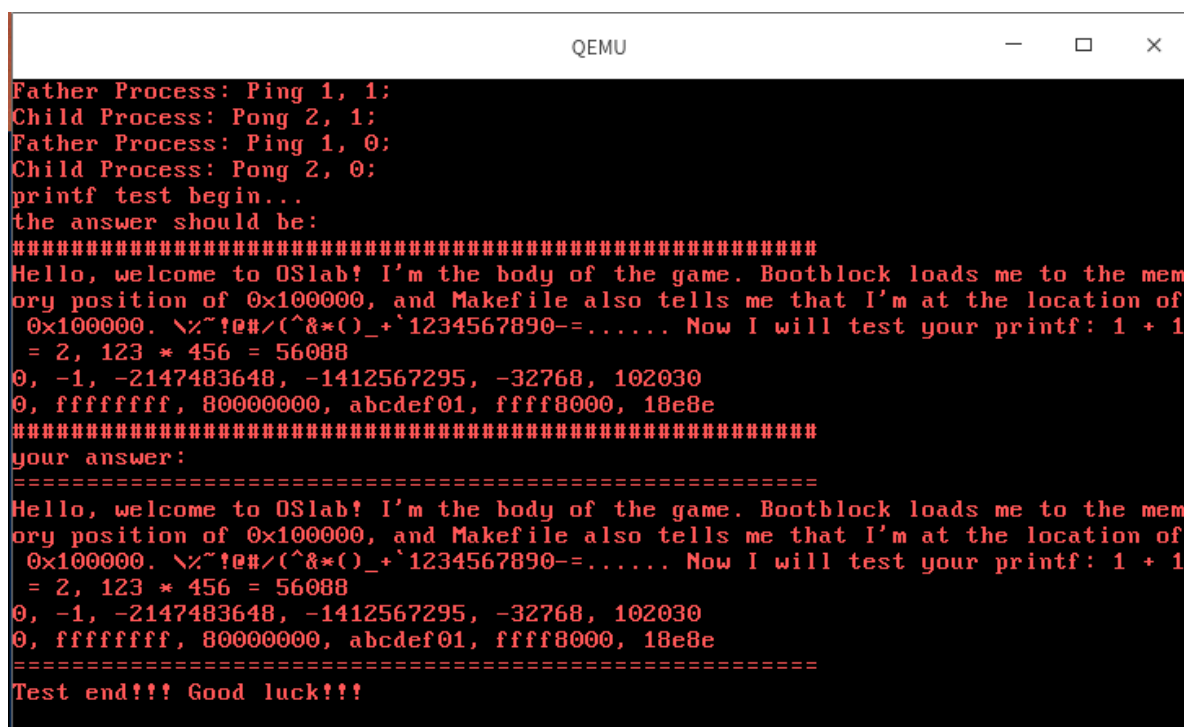
邮箱: [181250068@smail.nju.edu.cn](mailto:181250068@smail.nju.edu.cn)

QQ: 936637810

## 实验进度

我完成了基于时间中断进行进程切换完成任务调度的全过程

## 实验结果



```
QEMU
Father Process: Ping 1, 1;
Child Process: Pong 2, 1;
Father Process: Ping 1, 0;
Child Process: Pong 2, 0;
printf test begin...
the answer should be:
#####
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \x~!@#/(^&*())_+`1234567890-=..... Now I will test your printf: 1 + 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
#####
your answer:
=====
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \x~!@#/(^&*())_+`1234567890-=..... Now I will test your printf: 1 + 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
=====
Test end!!! Good luck!!!
```

## 实验修改的代码

`irqHandle.c` 中所有标识了 `//TODO in lab3` 的方法

`kvm.c` 中的 `loadElf`

`syscall.c` 中所有标识了 `//TODO in lab3` 的方法

## 实验中遇到的问题

`timerHandle` 中进程切换时, 原先如果原先的进程的 `timeCount` 没有被

`parent` 和 `child` 两个进程之间没法正常切换

仔细研究后发现问题是出在 `timerHandle` 方法中，在进行进程切换时，不能从第0个进程开始，因为第0个进程是最开始的内核进程，如果我们从0开始会直接切换到内核进程，但是子进程应该是第二个进程，进一步推广来说，假设此时我们的进程号是 $j$ ，那么我们应该从 $j + 1$ 开始轮询pcb，直到走完整个循环

## 没有理解临界区的更新一致性问题

在代码中添加 `asm volatile("int $0x20")` 后，父子进程的打印开始混乱，这是在意料之中的事情，但是我并没有理解到底该如何验证这个一致性的问题。是让我们通过PV信号来解决displayrow的访问吗？如果在print里面打开允许进行进程切换的话，前一个进程会在打印的中途切换到后一个进程，后一个进程会跟着前一个进程打印结束的地方继续打印，所以对于临界区的代码，需要用PV信号的方法，在发生进程中断的时候阻塞其他想要进入的进程。

## 关于中断嵌套的问题

想了一下，感觉需要在 `irqHandle` 中添加代码来处理中断嵌套，尝试利用lab2中提到的tff和压栈的方法，但是读了一下代码后，感觉这部分的功能是不是已经被实现好了？

```
void irqHandle(struct TrapFrame *tf) { // pointer tf = esp
    /*
     * 中断处理程序
     */
    /* Reassign segment register */
    asm volatile("movw %%ax, %%ds"::"a"(KSEL(SEG_KDATA)));

    uint32_t tmpStackTop = pcb[current].stackTop;
    pcb[current].prevStackTop = pcb[current].stackTop;
    pcb[current].stackTop = (uint32_t)tf;

    switch(tf->irq) {
        case -1:
            break;
        case 0xd:
            GProtectFaultHandle(tf); // return
            break;
        case 0x20:
            timerHandle(tf);          // return or iret
            break;
        case 0x21:
            keyboardHandle(tf);        // return
            break;
        case 0x80:
            syscallHandle(tf);         // return
            break;
        default: assert(0);
    }

    pcb[current].stackTop = tmpStackTop;
}
```

准备直接在代码中添加中断来测试，但是根据手册上的方法，在循环中，每一次都会调用一次 `asm volatile("int $0x20")`，这导致代码运行的时间非常的长。尝试只让这条语句被调用一次来测试是否完成了嵌套。

```
enableInterrupt();
int k = 1;
for(int j = 0; j < 0x100000; j++)
{
    *(uint8_t*)(j + (i+1)*0x100000) = *(uint8_t*)(j + (current+1)*0x100000);
    if(k==1 && j==10000)
    {
        putString("begin int 20");
        asm volatile("int $0x20");
        putString("end int 20");
        k = 0;
    }
}
disableInterrupt();
```

最后输出结果依然正确。