lab3实验报告

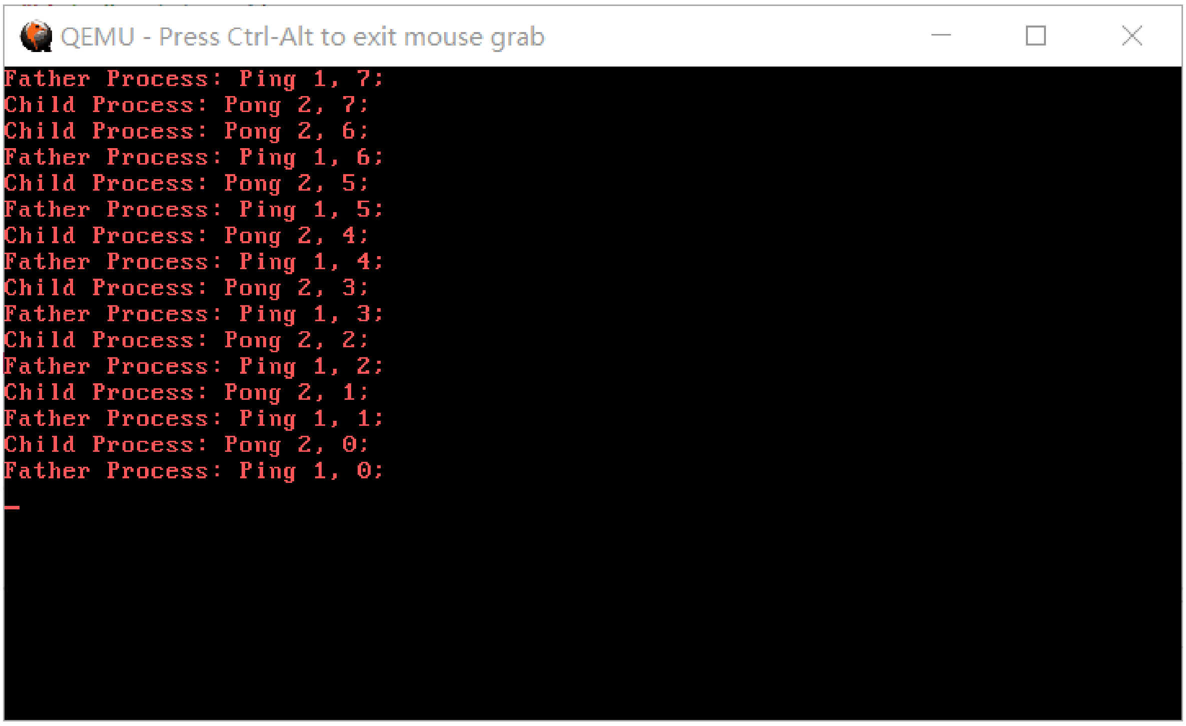
刘国涛 181860055 计算机科学与技术系

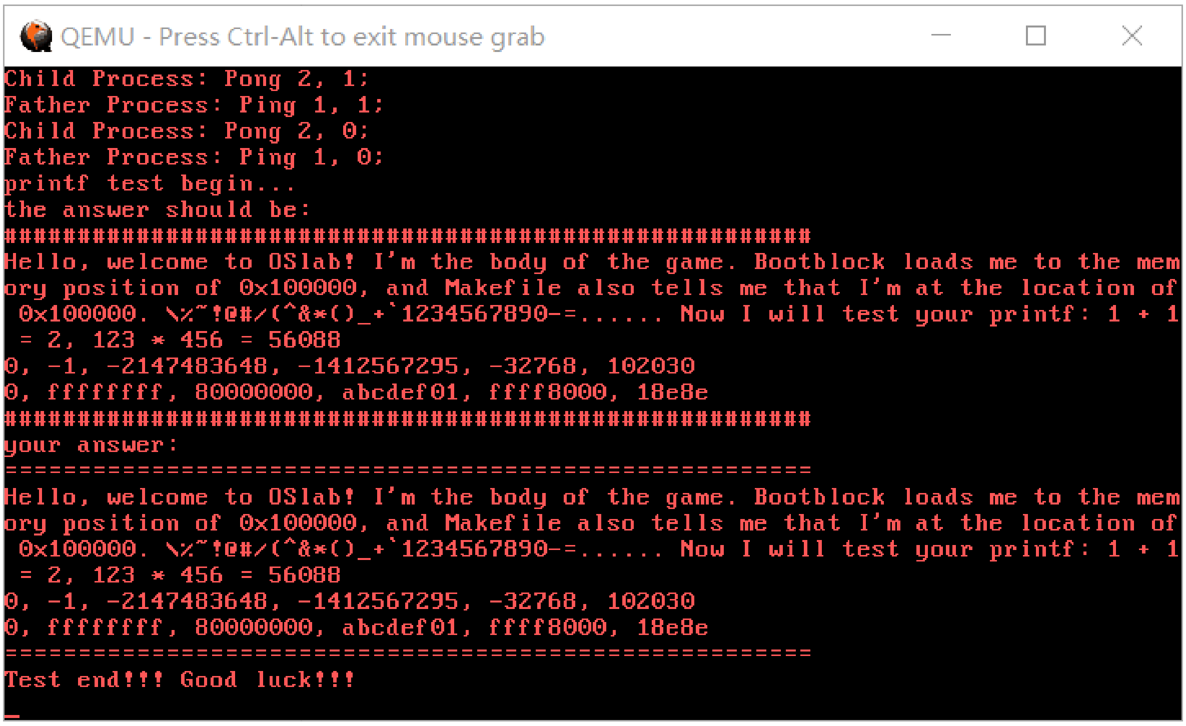
181860055@smail.nju.edu.cn

实验描述

实验进度:完成了所有必做的实验内容和中断嵌套的选做内容

实验结果:

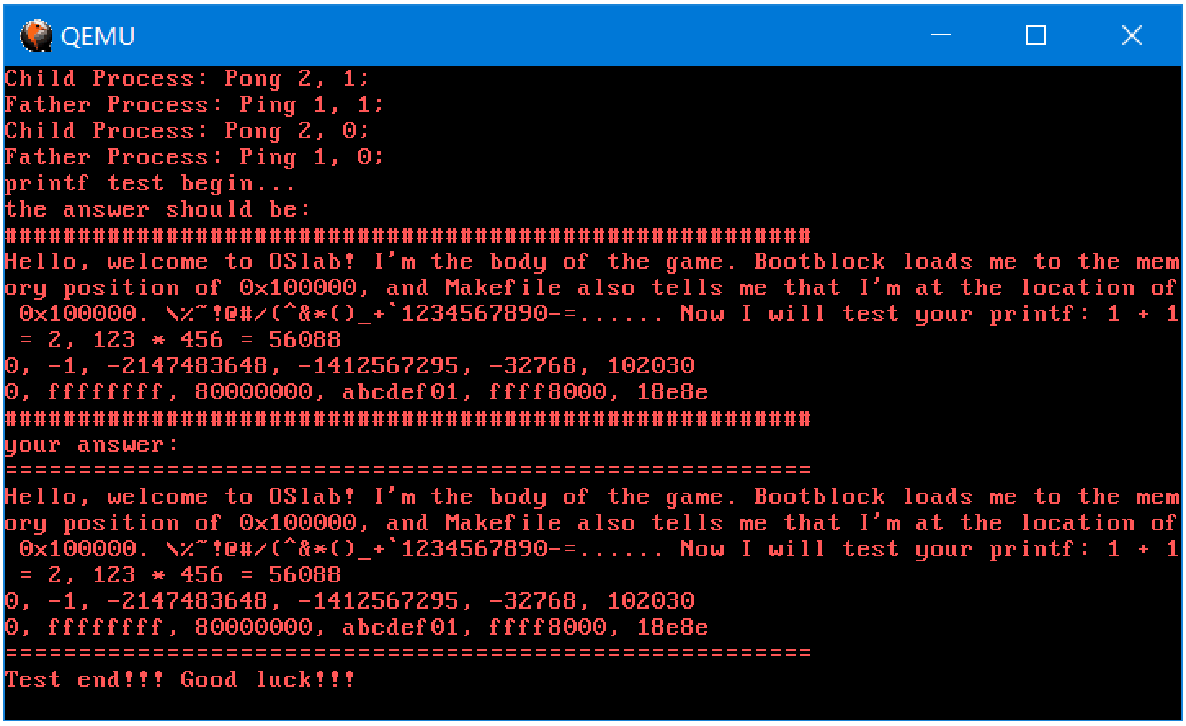




中断嵌套的结果:

```c
int count = 0;
    //拷贝父进程内容到子进程中
    enableInterrupt();
    for(int i=0;i<0x100000;++i){
        *(uint8_t*)(i+(slot+1)*0x100000) = *(uint8_t*)(i+(current+1)*0x100000);
        if(++count>1000){
            count = 0;
            asm volatile("int $0x20");
        }

    }
    disableInterrupt();
```



实验过程

1、完成库函数

通过在 `syscall.c` 中对 `fork` `exec` `sleep` `exit` 添加系统调用函数，并传入需要的参数即可

```
pid_t fork() {
    return syscall(SYS_FORK,0,0,0,0,0);
}

int exec(const char *filename, char * const argv[]) {
    return syscall(SYS_EXEC,(uint32_t)filename,(uint32_t)argv,0,0,0
);
}

int sleep(uint32_t time) {
    return syscall(SYS_SLEEP,time,0,0,0,0);
}

int exit() {
    return syscall(SYS_EXIT,0,0,0,0,0);
}
```

2、实现时间中断处理

该函数需要完成的功能分为两个部分：

1. 将BLOCKED的进程的 `sleepTime` 减一，并将 `sleepTime` 的进程重设为RUNNABLE
2. 当前进程的时间片加一，如果时间片达到最大值，则切换进程

```c
void timerHandle(struct TrapFrame *tf) {
    // TODO in Lab3

    for(int i=1;i<MAX_PCB_NUM;++i){
        if(pcb[i].state == STATE_BLOCKED){
            --pcb[i].sleepTime;
            if(pcb[i].sleepTime<=0){
                pcb[i].state = STATE_RUNNABLE;
            }


        }
    }
    if(++pcb[current].timeCount>=MAX_TIME_COUNT){
        switch_proc();  //进程切换
    }
    return;
}
```

对于进程切换的函数：

1. 找到一个RUNNABLE的进程 `next`
2. 将 `next` 进程设为RUNNING， `current` 进程设为RUNNABLE
3. 将当前进程的时间片清空
4. 将 `current` 切换到 `next`

```c
    int next = 0;
    for(int i=MAX_PCB_NUM - 1;i>=0;--i){
        if(pcb[i].state == STATE_RUNNABLE){
            next = i;
            break;
        }
    }

    pcb[next].state = STATE_RUNNING;
    if(pcb[current].state == STATE_RUNNING)
        pcb[current].state = STATE_RUNNABLE;
    pcb[current].timeCount = 0;
    current = next;
```

然后再调用给出的进程切换代码完成切换：

```
    uint32_t tmpStackTop = pcb[current].stackTop;
    pcb[current].stackTop = pcb[current].prevStackTop;
    tss.esp0 = (uint32_t)&(pcb[current].stackTop);
    asm volatile("movl %0, %%esp" ::"m"(tmpStackTop));
// switch kernel stack
    asm volatile("popl %gs");
    asm volatile("popl %fs");
    asm volatile("popl %es");
    asm volatile("popl %ds");
    asm volatile("popal");
    asm volatile("addl $8, %esp");
    asm volatile("iret");
```

3、实现系统调用例程

（1）syscallFork

首先需要再进程池中找一个DEAD进程作为即将fork的进程

```
    //find a dead process
    struct ProcessTable* new_pcb = NULL;
    int slot = -1;
    for(int i=1;i<MAX_PCB_NUM;++i){
        if(pcb[i].state == STATE_DEAD){
            new_pcb = &pcb[i];
            slot = i;
            break;
        }
    }
    if(new_pcb == NULL){
        pcb[current].regs.eax = -1; //fork fail
        return;
    }
```

然后将父进程的内容拷贝到子进程中，内容设置参考 `initSeg` 函数， `stackTop` 和 `prevStackTop` 分别是从父进程加上两个进程所在内存空间的起始地址的差值得到

gs ds ss fs es都是USEL(2+slot*2)，cs是USEL(1+slot*i)

其他的内容都由父进程拷贝而来

```
    //拷贝父进程内容到子进程中

    for(int i=0;i<0x100000;++i){
        *(uint8_t*)(i+(slot+1)*0x100000) = *(uint8_t*)(i+(current+1)
*0x100000);
    }
    uint32_t delta = (uint32_t)new_pcb-(uint32_t)&pcb[current];
    new_pcb->stackTop = pcb[current].stackTop+delta;
    new_pcb->prevStackTop = pcb[current].prevStackTop+delta;
    new_pcb->regs.gs = new_pcb->regs.fs = new_pcb->regs.es = new_pcb
->regs.ss = new_pcb->regs.ds = USEL(2+slot*2);
    new_pcb->regs.cs = USEL(1+slot*2);
    new_pcb->state = STATE_RUNNABLE;
    new_pcb->timeCount = 0;
    new_pcb->sleepTime = 0;
    new_pcb->pid = slot;

    new_pcb->regs.esp = pcb[current].regs.esp;
    new_pcb->regs.eip = pcb[current].regs.eip;
    new_pcb->regs.ecx = pcb[current].regs.ecx;
    new_pcb->regs.edx = pcb[current].regs.edx;
    new_pcb->regs.ebx = pcb[current].regs.ebx;
    new_pcb->regs.xxx = pcb[current].regs.xxx;
    new_pcb->regs.ebp = pcb[current].regs.ebp;
    new_pcb->regs.esi = pcb[current].regs.esi;
    new_pcb->regs.edi = pcb[current].regs.edi;
```

然后设置返回值，子进程的返回值设为0，父进程的返回值设为子进程的pid

```
    // return success value and return pid for parent process
    new_pcb->regs.eax = 0;
    pcb[current].regs.eax = new_pcb->pid;
    return;
```

（2）syscallSleep

实现sleep的例程需要将当前进程设为BLOCKED，然后将 `sleepTime` 设为参数值，最后要将 `timeCount` 设为最大值，强制发生进程切换

```c
void syscallSleep(struct TrapFrame *tf) {
    // TODO in Lab3
    putString("sleep\n");
    pcb[current].state = STATE_BLOCKED;
    pcb[current].sleepTime = tf->ecx;
    pcb[current].timeCount = MAX_TIME_COUNT;
    timerHandle(tf);
    return;
}
```

(3) syscallExit

将当前进程设为DEAD，并将timeCount设为最大值以强制发生进程切换

```c
void syscallExit(struct TrapFrame *tf) {
    // TODO in Lab3
    putString("exit\n");
    pcb[current].state = STATE_DEAD;
    pcb[current].timeCount = MAX_TIME_COUNT;
    timerHandle(tf);
    return;
}
```

(4) syscallExec

实现 `syscallExec` 分为三步

1. 实现loadElf
2. 在syscallExec内获取filename，并传入loadElf中
3. 将在loadElf中设置的entry传给eip

首先，loadElf的实现可参考loadUMain

```c
int loadElf(const char *filename, uint32_t physAddr, uint32_t *entry
)
{

    int i = 0;
    int phoff = 0x34;    // program header offset
    int offset = 0x1000; // .text section offset
    uint32_t elf = physAddr; // physical memory addr to load
    Inode inode;
    int inodeOffset = 0;
    int ret = readInode(&sBlock, &inode, &inodeOffset, filename);
    if(ret == -1)
        return -1;
    for (i = 0; i < inode.blockCount; i++)
    {
        ret = readBlock(&sBlock, &inode, i,
            (uint8_t *)(elf + i * sBlock.blockSize));
        if(ret == -1)
            return -1;
    } // entry address of the program
    *entry = ((struct ELFHeader *)elf)->entry;
    phoff = ((struct ELFHeader *)elf)->phoff;
    offset = ((struct ProgramHeader *)(elf + phoff))->off;
    for (i = 0; i < 200 * 512; i++)
    {
        *(uint8_t *)(elf + i) = *(uint8_t *)(elf + i + offset);
    }
    return 0;
}
```

然后在syscallExec中获取filename，参考了syscallPrint，使用段选择子进行获取

```c
    char filename[100];
    int sel = tf->ds;
    char *str = (char*)tf->ecx;
    char character = 1;
    asm volatile("movw %0, %%es"::"m"(sel));
    for(int i=0;character!='\0';++i){
        asm volatile("movb %%es:(%1), %0":"=r"(character):"r"(str +
 i));
        //putChar(character);
        filename[i] = character;
    }
```

然后调用loadElf并设置entry

```c
uint32_t entry;
int ret = loadElf(filename, (current + 1) * 0x100000, &entry);
if(ret == -1){
    tf->eax = -1;
    putString("load elf failed!\n");
    return;
}

tf->eip = entry;
```

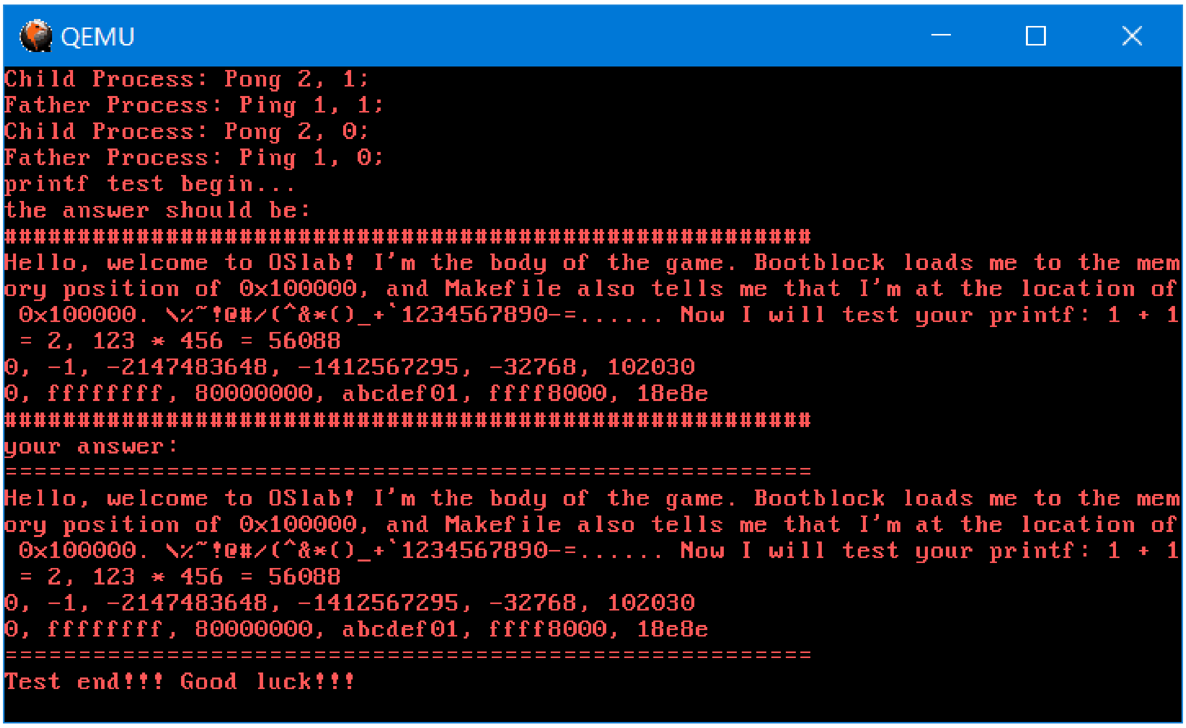
4、中断嵌套

在syscallFork中加入手动模拟时钟中断

```c
int count = 0;
    //拷贝父进程内容到子进程中
    enableInterrupt();
    for(int i=0;i<0x100000;++i){
        *(uint8_t*)(i+(slot+1)*0x100000) = *(uint8_t*)(i+(current+1)*0x100000);
        if(++count>1000){
            count = 0;
            asm volatile("int $0x20");
        }

    }
    disableInterrupt();
```

运行后得到结果：



说明能够完成中断嵌套

至此必做部分和中断嵌套的选做部分都已完成

实验感受

本次实验在某些部分需要参考框架代码进行实现，因此需要对框架代码进一步理解。本次实验也让我对系统的进程管理有了进一步的理解