操作系统实验 Lab3

一、基本信息：

姓名：黄云鹏

学号：181860036

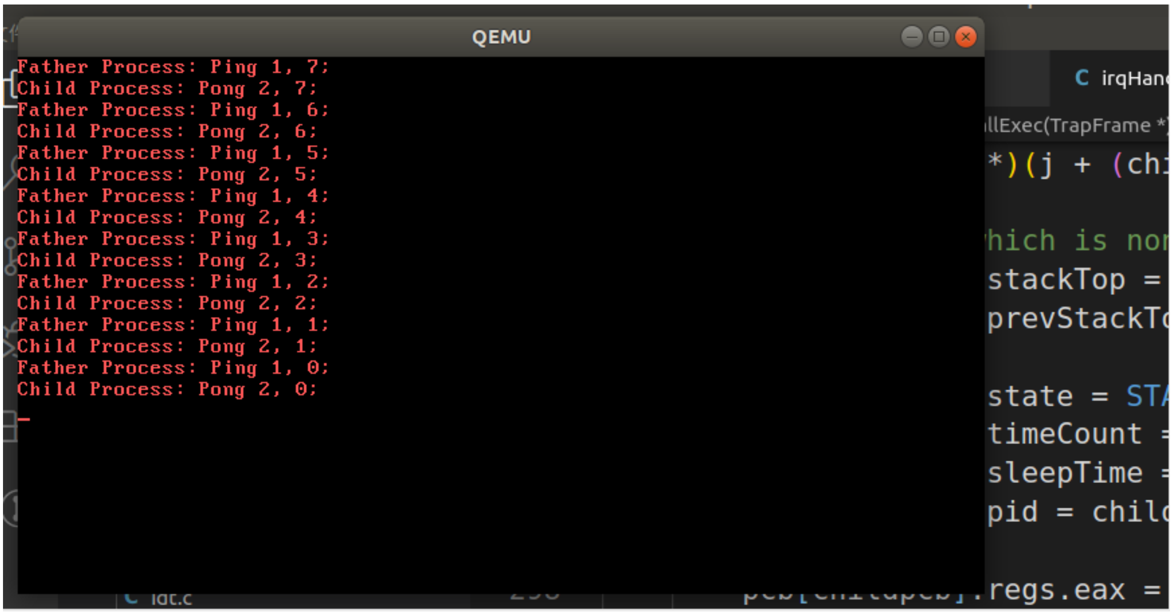
邮箱：181860036@smail.nju.edu.cn

二、实验进度：

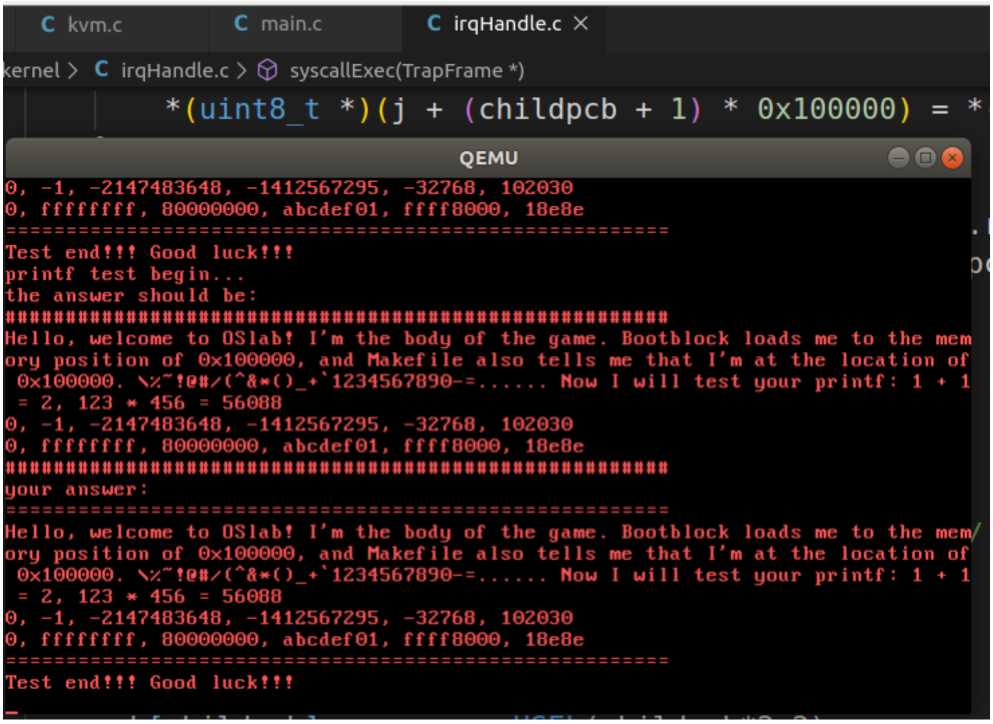
我完成了所有必做内容

三、实验结果：

=>1. 实现一对父子进程的“ping pong”输出程序：



=>2. 并在1.程序之后调用lab2中的打印程序：



四、修改代码：

1）通过调用syscall(), 设置不同的参数以实现四个库函数：

```
pid_t fork() {
    // TODO in lab3
    pid_t ret = syscall(SYS_FORK,0,0,0,0,0);
    return ret;
}

int exec(const char *filename, char * const argv[]) {
    // TODO in lab3        You, 20 hours ago • add lab_3
    int ret = syscall(SYS_EXEC,(uint32_t)filename,(uint32_t)argv,0,0,0);
    return ret;
}

int sleep(uint32_t time) {
    // TODO in lab3
    //printf("time is %d\n",time);
    int ret = syscall(SYS_SLEEP,time,0,0,0,0);
    return ret;
}

int exit() {
    // TODO in lab3
    int ret = syscall(SYS_EXIT,0,0,0,0,0);
    return ret;
}
```

2）通过完成时钟中断处理例程timerHandle()，实现进程切换机制：

```c
void timerHandle(struct TrapFrame *tf) {
    // TODO in lab3
    //putString("get in time break\n");
    int minrunnable = MAX_PCB_NUM;  // the runnab
    int i=0;
    // find blocked pcbs, whose sleeptime--,
    // and when sleeptime = 0, turn to runnable p
    for(i=0; i<MAX_PCB_NUM;i++)
    {
        if (pcb[i].state == STATE_BLOCKED)
        {⋯
        }
    }
```

```c
if(pcb[current].timeCount == MAX_TIME_COUNT)
{
         // find next runnable user pcb after currentpcb
        for(i=current;i<MAX_PCB_NUM;i++)
        {⋯
        }
        // find next runnable user pcb from beginning
        if(minrunnable == MAX_PCB_NUM)
        {⋯
        }
        // minrunnable user pcb exists
        if(minrunnable != MAX_PCB_NUM)
        {⋯
        }
        // only IDLE runnable
        else if(pcb[0].state == STATE_RUNNABLE)
        {⋯
        }
        // no pcb runnable
        else
        {⋯
        }
```

```c
// switch process's stack and registers
uint32_t tmpStackTop = pcb[current].stackTop;
pcb[current].stackTop = pcb[current].prevStackTop;
tss.esp0 = (uint32_t)&(pcb[current].stackTop);  // set tss
asm volatile("movl %0, %%esp"::"m"(tmpStackTop)); // switc
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syscallFork()处理例程，完成fork()函数的调用：

```c
void syscallFork(struct TrapFrame *tf) {
    // TODO in lab3
    int has = 0;
    int i= 0; int j=0;
    int childpcb = current;
    for(i=1;i<MAX_PCB_NUM;i++)
    {        You, 15 hours ago • Done syscallFork() i
    }

    if(has == 0)  //no empty, fork failed,
    {…
    }
    else // found one, fork successed,
    {…
    }

    return;
}
```

4）实现syscallExec()处理例程和loadElf()，完成exec()函数的调用：

```c
void syscallExec(struct TrapFrame *tf) {
    // TODO in lab3
    // read filename
    int sel = tf->ds;
    char *str = (char *)tf->ecx;  // str: filename
    char character[200];  // filename copy
    uint32_t count = 0;
    int i=0;
    int ret = -1;
    uint32_t entry = 0;        You, 5 hours ago • Done syscallExec(

    asm volatile("movw %0, %%es"::"m"(sel));
    for (i = 0; i < 200; i++) {…
    }

    //putString("Character[] is:\n ");
    //putString(character);

    // loadElf
    ret = loadElf(character,(current + 1) * 0x100000, &entry);
```

```c
int loadElf(const char *filename, uint32_t physAddr, uint32_t *entry) {
    // TODO in lab3
    Inode inode;    // file inode
    int inodeOffset = 0;
    int i=0;
    int j=0;
    int success = -1;
    uint32_t phoff = 0; // program header offset
    uint32_t ephoff = 0;  // end of program headers
    uint32_t offset = 0; //section offset
    uint32_t vaddr = 0;  // virtual address
    uint32_t paddr = 0;  // physical address
    uint32_t filesz = 0;  // file size
    uint32_t memsz = 0;  // memory size
    uint32_t elf = 0; // physical memory addr to load

    success = readInode(&sBlock, &inode, &inodeOffset, filename);
    if(success == -1)  //readinode failed
    {…
    }
```

5）实现syscallSleep()处理例程，完成sleep()函数的调用：

```c
void syscallSleep(struct TrapFrame *tf) {
    // TODO in lab3
    //putString("tf->ecx = ");
    //putInt(tf->ecx);
    //putString("\n");
    if(tf->ecx > 0)
    {
        pcb[current].state = STATE_BLOCKED;   // state -> b
        pcb[current].timeCount = MAX_TIME_COUNT;
        pcb[current].sleepTime = tf->ecx; // ecx = time (f
        asm volatile("int $0x20");  // call timerHandle()
    }
    return;
```

6）实现syscallExit()处理例程，完成exit()函数的调用：

```c
void syscallExit(struct TrapFrame *tf) {
    // TODO in lab3
    pcb[current].state = STATE_DEAD;
    asm volatile("int $0x20");
    return;
}
```

五、实验思考题：

六、实验心得：

=> 复习了有关elf文件结构以及加载过程的知识

=> 大致了解了通过时间片轮转机制以及时钟中断处理实现进程切换的过程

=> 对于在进程管理中，进程控制块的结构和作用的理解更清晰了

=> 对于在中断和进程切换过程中，堆栈的转化方法更加熟悉了

=> 对库函数的实现方法， 以及系统调用的相关机制理解更深刻了

=> 在处理如字符串等数据时，要时刻明确内存空间（即堆栈）是否一致，保证访问段的正确

=> 对于返回值的设置，需要函数和硬件进行约定，通过堆栈进行保存，并放在规定的寄存器中