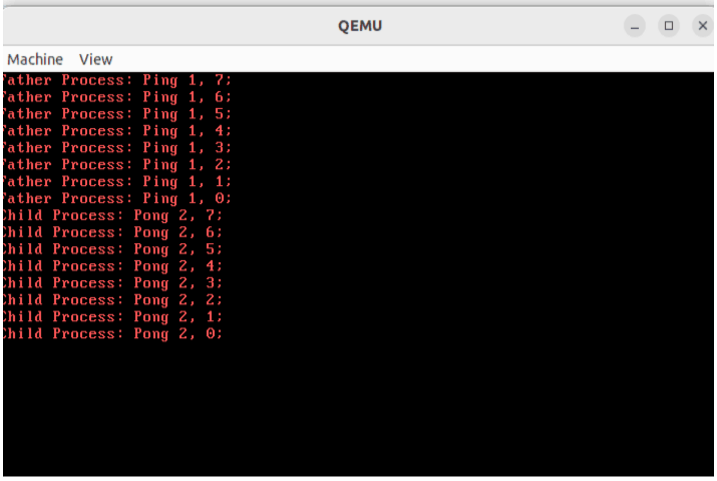
OS-lab3-report

姓名	时昌军
学号	221220085
邮箱	221220085@smail.nju.edu.cn

一、实验进度

实验要求全部完成

二、实验结果



```
ather Process: Ping 1, 7;
ather Process: Ping 1, 6;
ather Process: Ping 1, 5;
ather Process: Ping 1, 4;
ather Process: Ping 1, 3;
ather Process: Ping 1, 2;
ather Process: Ping 1, 1;
ather Process: Ping 1, 0;
hild Process: Pong 2, 7;
hild Process: Pong 2, 6;
hild Process: Pong 2, 5;
hild Process: Pong 2, 4;
hild Process: Pong 2, 3;
hild Process: Pong 2, 2;
hild Process: Pong 2, 1;
hild Process: Pong 2, 0;
```

三、实验修改的代码

1. 完成库函数

这一部分算是对实验2的复习

```
pid_t fork(){
    return syscall(SYS_FORK, 0, 0, 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. 时钟中断处理

1. 遍历pcb，将状态为STATE_BLOCKED的进程的sleepTime减一，如果进程的sleepTime变为0，重新设为STATE_RUNNABLE
2. 将当前进程的timeCount加一，如果时间片用完（timeCount==MAX_TIME_COUNT）且有其它 状态为STATE_RUNNABLE的进程，切换，否则继续执行当前进程

```
void timerHandle(struct StackFrame *sf)
{
    // TODO
    for (int i = 0; i < MAX_PCB_NUM; i++){
        if (pcb[i].state == STATE_BLOCKED){
            pcb[i].sleepTime--;
            if (pcb[i].sleepTime == 0)
                pcb[i].state = STATE_RUNNABLE;
        }
    }
    pcb[current].timeCount++;
    if (pcb[current].timeCount >= MAX_TIME_COUNT){
        int i = (current + 1) % MAX_PCB_NUM;
        while (i != current){
            if (pcb[i].state == STATE_RUNNABLE)
                break;
            i = (i + 1) % MAX_PCB_NUM;
        }

        if (i == current){
            if (pcb[current].state == STATE_RUNNABLE || pcb[current].state == STATE_RUNNING){
                pcb[current].timeCount = 0;
            }
            else
                current = 0;
        }
        else{
            current = i;
            pcb[current].state = STATE_RUNNING;
        }
    }

    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. 系统调用例程

3.1 syscallFork

syscallFork要做的是在寻找一个空闲的pcb做为子进程的进程控制块，将父进程的资源复制给子进程。如果没有空闲pcb，则fork失败，父进程返回-1，成功则子进程返回0，父进程返回子进程pid

在处理fork时有以下几点注意事项：

1. 代码段和数据段可以按照2.4.1.节最后的说明进行完全拷贝
2. pcb的复制时，需要考虑哪些内容可以直接复制，哪些内容通过计算得到，哪些内容和父进程无关
3. 返回值放在哪

   提示： initProc 中有初始化 pcb[0] 和pcb[1] 的经验可供参考

3.2 syscallSleep

将当前的进程的sleepTime设置为传入的参数，将当前进程的状态设置为STATE_BLOCKED，然后利用 asm volatile("int $0x20"); 模拟时钟中断，利用 timerHandle 进行进程切换 需要注意的是判断传入参数的合法性

3.3 syscallExit

将当前进程的状态设置为STATE_DEAD，然后模拟时钟中断进行进程切换

```
// TODO syscallFork ...
void syscallFork(struct StackFrame *sf) {
    int Pos = -1;
    for(int i = 1; i < MAX_PCB_NUM; ++i){
        if(pcb[i].state == STATE_DEAD){
            Pos = i;
        }
    }
    if(Pos == -1) {
        // FORK FAILED
        pcb[current].regs.eax = -1;
    } else {
        // FORK SUCCESSFUL
        enableInterrupt();
        for (int j = 0; j < 0x100000; j++) {
            *(uint8_t *)(j + (Pos+1)*0x100000) = *(uint8_t *)(j + (current+1)*0x100000);
        }
        disableInterrupt();
        for (int j = 0; j < sizeof(ProcessTable); ++j)
            *((uint8_t *)(&pcb[Pos]) + j) = *((uint8_t *)(&pcb[current]) + j);
        // user process  initProc_reference
        pcb[Pos].stackTop = (uint32_t)&(pcb[Pos].regs);
        pcb[Pos].prevStackTop = (uint32_t)&(pcb[Pos].stackTop);
        pcb[Pos].state = STATE_RUNNABLE;
        pcb[Pos].timeCount = 0;
        pcb[Pos].sleepTime = 0;
        pcb[Pos].pid = Pos;

        pcb[Pos].regs.ss = USEL(2+2*Pos);
        pcb[Pos].regs.cs = USEL(1+2*Pos);
        pcb[Pos].regs.ds = USEL(2+2*Pos);
        pcb[Pos].regs.es = USEL(2+2*Pos);
        pcb[Pos].regs.fs = USEL(2+2*Pos);
        pcb[Pos].regs.gs = USEL(2+2*Pos);

        pcb[Pos].regs.eax = 0;
        pcb[current].regs.eax = Pos;

    }
}

void syscallSleep(struct StackFrame *sf){
    int time = sf->ecx;
    if(time >= 0){
        pcb[current].sleepTime = time;
        pcb[current].state = STATE_BLOCKED;
        asm volatile("int $0x20");
    }
}

void syscallExit(struct StackFrame *sf){
    pcb[current].state = STATE_DEAD;
    asm volatile("int $0x20");
}
```

四、实验心得

这次实验还是相对比较友好的。下次实验再接再厉。(●'◡'●)