

Lab3 实验报告

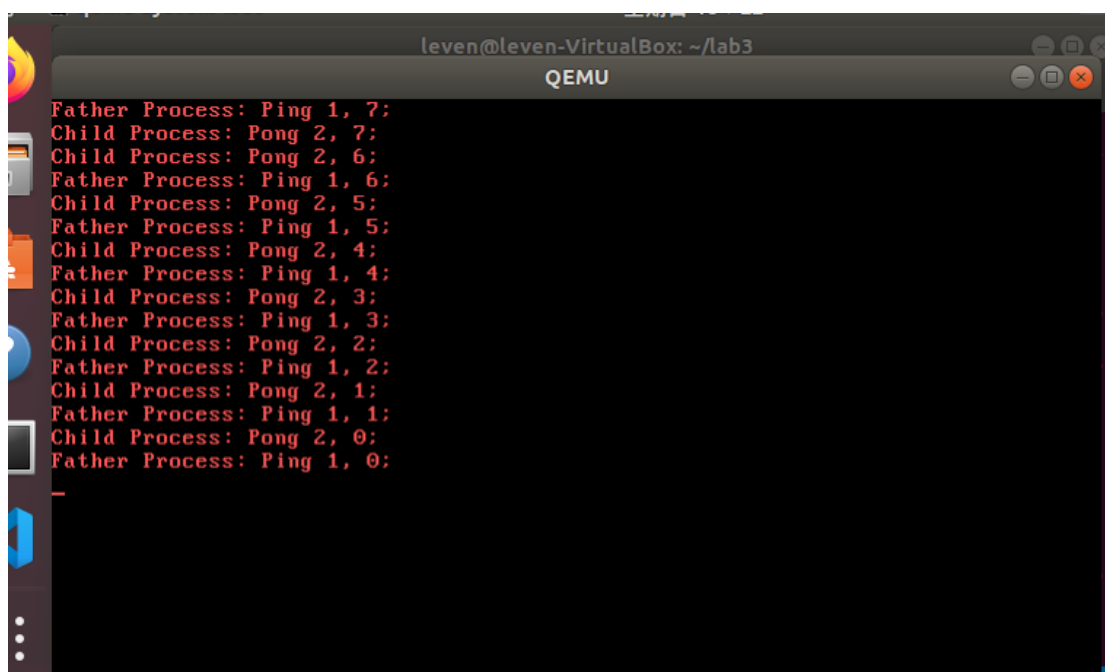
南京大学 人工智能学院

191300036 刘晔闻 450008668@qq.com

实验进度：我完成了 Lab3 的所有必做内容和选做内容的中断嵌套部分。

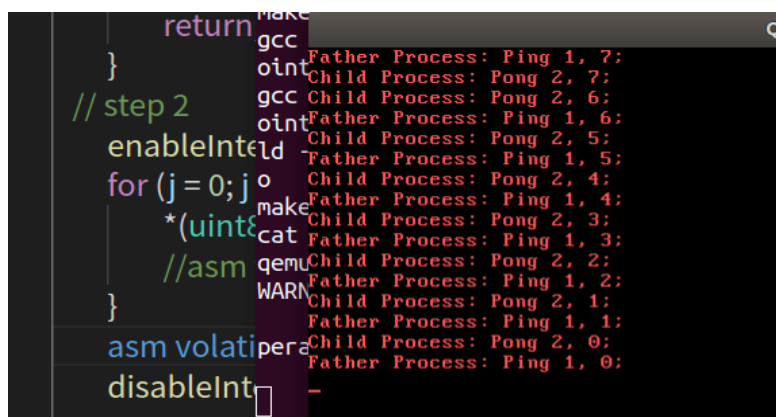
实验结果：

1. 必做内容：



```
leven@leven-VirtualBox: ~/lab3
QEMU
Father Process: Ping 1, 7:
Child Process: Pong 2, 7:
Child Process: Pong 2, 6:
Father Process: Ping 1, 6:
Child Process: Pong 2, 5:
Father Process: Ping 1, 5:
Child Process: Pong 2, 4:
Father Process: Ping 1, 4:
Child Process: Pong 2, 3:
Father Process: Ping 1, 3:
Child Process: Pong 2, 2:
Father Process: Ping 1, 2:
Child Process: Pong 2, 1:
Father Process: Ping 1, 1:
Child Process: Pong 2, 0:
Father Process: Ping 1, 0:
```

2. 中断嵌套：



```
return gcc
} oint
// step 2 gcc
enableInt oint
for (j = 0; j o
*(uint& make
//asm cat
} qemu
asm volatile WARN
disableInt -
```

实验过程:

1. lib/syscall.c:完成 sleep, exit 函数, 类似实例中的 fork 函数, 调用 syscall 函数即可。

2. kernel/kernel/irqHandle.c:

(1) 保存恢复 esp 到栈顶: 按照实验指南中的代码填写即可。

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

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

(2) syscallHandle: 填写 syscallSleep, syscallExit 的 case, 这两者分别为 case 3 和 case 4, case 2 的 syscallExec 无需填写。我在 lib/lib.h 中修改了 case 值, 使 sleep 和 exit 分别为 2 和 3。

(3) switch_progress: 按照指南中的进程切换代码, 我实现了一个进程切换函数。

```
void switch_progress(){
    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));
    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");
}
```

(4) timeHandle: Step 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；

```
// step 1
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;
    }
}
```

Step 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。其中，切换不仅是执行进程切换函数 switch_progress，首先要在 pcb 进程中寻找下一个处于 RUNNABLE 状态的进程，设置为 RUNNING 状态，然后将该进程变为 current 进程，之后才能使用进程切换函数。

```
// step 2
pcb[current].timeCount++;
if (pcb[current].timeCount >= MAX_TIME_COUNT){
    int temp = 0;
    for(int i = MAX_PCB_NUM - 1; i >= 0; i--){
        if(pcb[i].state == STATE_RUNNABLE){
            temp = i;
            break;
        }
    }
    pcb[temp].state = STATE_RUNNING;
    if(pcb[current].state == STATE_RUNNING)
        pcb[current].state = STATE_RUNNABLE;
    pcb[current].timeCount = 0;
    current = temp;
}
```

(5) syscallSleep, syscallExit: syscallSleep 首先将当前的

进程的 sleepTime 设置为传入的参数，然后将当前进程的状态设置为 STATE_BLOCKED，之后模拟时钟中断；syscallExit 首先将当前进程的状态设置为 STATE_DEAD，然后模拟时钟中断。要注意的是，模拟时钟中断的目的是利用 timeHandle 进行进程切换，所以可以直接令 current 进程时间片已满，然后进入进程切换状态。（也可以不模拟时钟中断，而是手动使用函数 switch_progress 进行进程切换）

```
pcb[current].timeCount = MAX_TIME_COUNT;
asm volatile("int $0x20");
```

(6) syscallFork: Step 1: 寻找一个空闲的 pcb 做为子进程的进程控制块，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；

```
// step 1
int i;
int j;
for (i = 0; i < MAX_PCB_NUM; i++) {
    if (pcb[i].state == STATE_DEAD)
        break;
}
```

Step 2: 将父进程的资源复制给子进程。进程控制块中默认了 pcb[i] 的大小和地址；

```
// step 2
//enableInterrupt();
for (j = 0; j < 0x100000; j++) {
    *(uint8_t *) (j + (i+1)*0x100000) = *(uint8_t *) (j + (current+1)*0x100000);
    //asm volatile("int $0x20"); //XXX Testing irqTimer during syscall
}
//disableInterrupt();
```

Step 3: pcb[i]初始化。(可以参考 kernel/kernel/kvm.c 中的 initProc 函数。) 主要参考 pcb[2]的初始化。

```
// step 3
pcb[i].stackTop = (uint32_t)&(pcb[i].regs);
pcb[i].prevStackTop = (uint32_t)&(pcb[i].stackTop);
pcb[i].state = STATE_RUNNABLE;
pcb[i].timeCount = 0;
pcb[i].sleepTime = 0;
pcb[i].pid = i;

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

要注意将父进程的寄存器值复制给子进程，并将父进程的进程号设为子进程号，子进程号设为 0。

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

pcb[i].regs.eax = 0; //子进程赋0
pcb[current].regs.eax = i; //父进程赋子进程号
```

(7) 选做-中断嵌套：添加了嵌套中断代码和模拟时钟中断的代码。

```
enableInterrupt();
for (j = 0; j < 0x100000; j++) {
    *(uint8_t*)(j + (i+1)*0x100000) = *(uint8_t*)(j + (current+1)*0x100000);
    asm volatile("int $0x20"); //XXX Testing irqTimer during syscall
}
disableInterrupt();
```

只添加嵌套中断代码依旧可以运行，但是在循环内加入模拟时钟中断代码后，迟迟不打印字符。将模拟时钟中断代码移到循环外时，可以正常打印。