

南京大学本科生实验报告

课程名称： 操作系统

学院	计算机科学与技术		
学号	191220029	姓名	傅小龙
Email	1830970417@qq.com		

1.实验名称

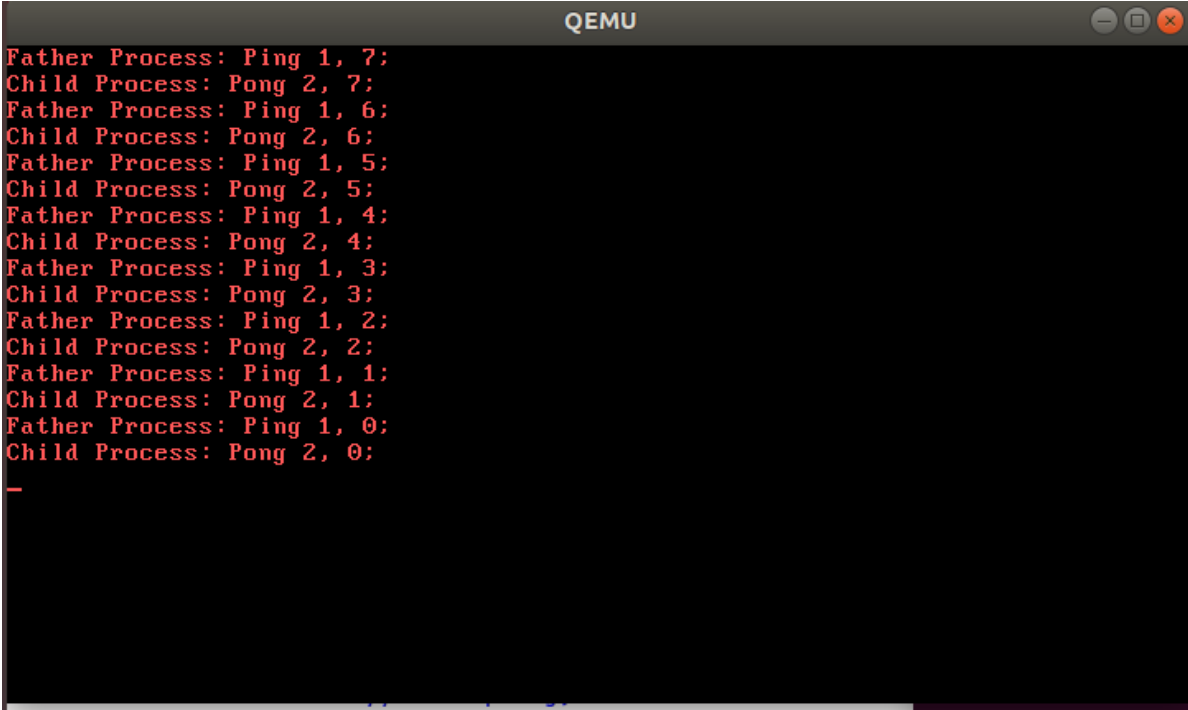
Lab3: 进程切换

2.实验进度

我完成了所有内容。

3. 实验结果

3.1系统调用例程



```
QEMU
Father Process: Ping 1, 7;
Child Process: Pong 2, 7;
Father Process: Ping 1, 6;
Child Process: Pong 2, 6;
Father Process: Ping 1, 5;
Child Process: Pong 2, 5;
Father Process: Ping 1, 4;
Child Process: Pong 2, 4;
Father Process: Ping 1, 3;
Child Process: Pong 2, 3;
Father Process: Ping 1, 2;
Child Process: Pong 2, 2;
Father Process: Ping 1, 1;
Child Process: Pong 2, 1;
Father Process: Ping 1, 0;
Child Process: Pong 2, 0;
_
```

3.2 中断嵌套：

加载时间受模拟中断影响大幅延长。

```
QEMU
Father Process: Ping 1, 7;
Child Process: Pong 2, 7;
Father Process: Ping 1, 6;
Child Process: Pong 2, 6;
Father Process: Ping 1, 5;
Child Process: Pong 2, 5;
Father Process: Ping 1, 4;
Child Process: Pong 2, 4;
Father Process: Ping 1, 3;
Child Process: Pong 2, 3;
Father Process: Ping 1, 2;
Child Process: Pong 2, 2;
Father Process: Ping 1, 1;
Child Process: Pong 2, 1;
Father Process: Ping 1, 0;
Child Process: Pong 2, 0;
```

3.3 临界区

参照 1ab3.pdf-2.5.中断嵌套与临界区 给出的方式模拟不同线程对同一资源的竞争。

```
QEMU
Father PrChild Process: Pong 2, 7;
;
Child Process: PFather Process: ong 2, 6;
Ping 1, 6;
Child Process: PFather Process: ong 2, 5;
Ping 1, 5;
Child Process: PFather Process: ong 2, 4;
Ping 1, 4;
Child Process: PFather Process: ong 2, 3;
Ping 1, 3;
Child Process: PFather Process: ong 2, 2;
Ping 1, 2;
Child Process: PFather Process: ong 2, 1;
Ping 1, 1;
Child Process: PFather Process: ong 2, 0;
Ping 1, 0;
```

4.代码修改处

kernel/kernel/irqHandle.c

添加外部变量的声明:

```
extern TSS tss;
extern int current;
extern ProcessTable pcb[MAX_PCB_NUM];
```

kernel/kernel/irqHandle.c void irqHandle(struct StackFrame *sf) 保存esp至栈顶和从栈顶恢复esp, 取消调用 timerHandle(sf); 语句的注释

kernel/kernel/irqHandle.c void syscallHandle(struct StackFrame *sf) 添加Fork, Sleep, Exit,系统调用

kernel/kernel/irqHandle.c 实现以下函数:

```
void syscallFork(struct StackFrame *sf);
void syscallSleep(struct StackFrame *sf);
void syscallExit(struct StackFrame *sf);
void timerHandle(struct StackFrame *sf);
```

lib/syscall.c int sleep(uint32_t time) 系统调用 syscallSleep 服务例程

lib/syscall.c int exit() 系统调用 syscallExit 服务例程

kernel/kernel/irqHandle.c void syscallPrint(struct StackFrame *sf) 模拟时钟中断实现临界区竞争 (提交的代码中已注释该语句)

5. 实验中遇到的问题及思考

5.1 程序控制块的赋值

在创建子进程的系统服务例程 void syscallFork(struct StackFrame *sf) 中, 发现在拷贝父进程资源时, 无法直接使用赋值语句 pcb[i] = pcb[current]; 而需要逐个字节拷贝或者逐个成员变量拷贝才能成功。原因在于子进程和父进程的程序控制块并不在同一用户空间, 如果直接赋值, 编译器生成的汇编代码中关于赋值的语句将采用带有 rep 的 mov 指令来实现对一片连续空间的赋值, 源和目标地址并没有标记不同的段, 故运行时qemu会报内存访问错。而采用指针逐字节拷贝则可以避免这一问题。

5.2 中断嵌套测试时间过长

尽管中断嵌套的测试有很多时间花费在了打印提示信息上, 但还是共花费了大约3.5h才完成。建议模拟时钟中断的自陷内联汇编可以间隔一小段时间再调用, 比如说根据拷贝空间时的指针 j, 每满足 j % x == 0 (x为某一值)时调用时钟中断, 如下所示

```
if(!(j % 0x1000))
    asm volatile("int $0x20"); //Testing irqTimer during syscall
```

这样就可以缩短不少测试时间, 也不影响测试结果。