

Lab3 进程切换

181860044 李翰

邮箱: 181860044@smail.nju.edu.cn

1. 实验进度: 完成了所有必做内容, 以及选做部分的中断嵌套

2. 实验结果:

2.1 未开启中断嵌套 (必做内容):

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

C cpu.h 209 pcb[i].timeCount = 0;
C io.h 210 pcb[i].sleepTime = 0;
C irq.h 211 pcb[i].pid = i;
```

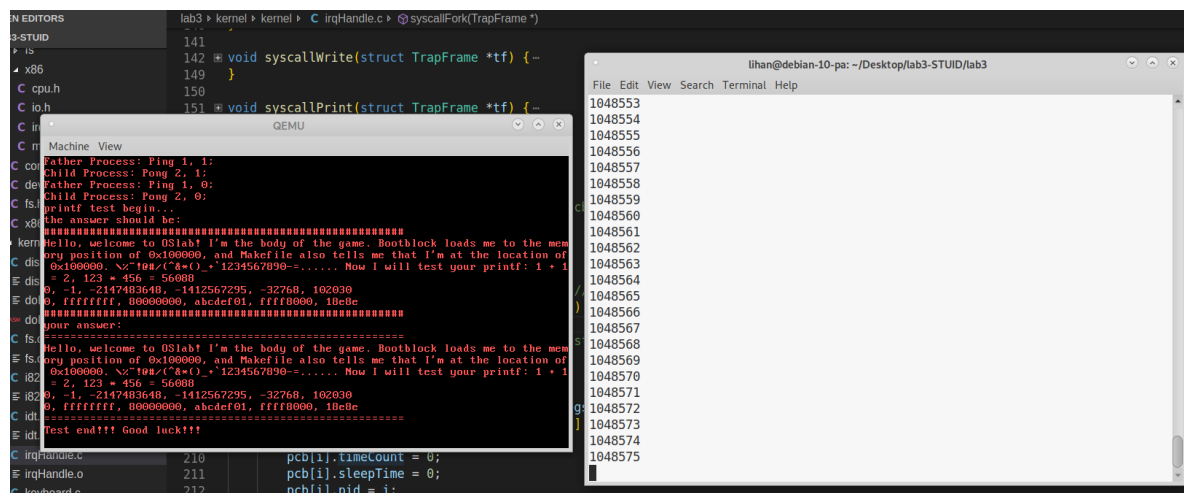
```
lihan@debian-10-pa: ~/Desktop/lab3-STUID/lab3
File Edit View Search Terminal Help
1019 inodes and 3929 data blocks available.
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: ., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13708.
LS success.
1019 inodes and 3929 data blocks available.
ls /boot/initrd
Inode: 3, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13708.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: ., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: print, Inode: 5, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13628.
LS success.
1019 inodes and 3929 data blocks available.
make[1]: Leaving directory '/home/lihan/Desktop/lab3-STUID/lab3/app'
cat bootloader/bootloader.bin kernel/kMain.elf app/fs.bin > os.img
lihan@debian-10-pa: ~/Desktop/lab3-STUID/lab3$ make play
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write
operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
```

```
Machine View
Father Process: Ping 1, 1;
Child Process: Pong 2, 1;
Father Process: Ping 1, 0;
Child Process: Pong 2, 0;
printf test begin...
the answer should be:
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. Now I will test your printf: 1 * 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
your answer:
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. Now I will test your printf: 1 * 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
Test end!!! Good luck!!!

F is 200 pcb[i].stackTop = (uint
x86 207 pcb[i].prevStackTop = (
cpu.h 208 pcb[i].state = STATE RU
io.h 209 pcb[i].timeCount = 0;
irq.h 210 pcb[i].sleepTime = 0;
211 pcb[i].pid = i;
```

```
lihan@debian-10-pa: ~/Desktop/lab3-STUID/lab3
File Edit View Search Terminal Help
1019 inodes and 3929 data blocks available.
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: ., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13708.
LS success.
1019 inodes and 3929 data blocks available.
ls /boot/initrd
Inode: 3, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13708.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: ., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: print, Inode: 5, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13628.
LS success.
1019 inodes and 3929 data blocks available.
make[1]: Leaving directory '/home/lihan/Desktop/lab3-STUID/lab3/app'
cat bootloader/bootloader.bin kernel/kMain.elf app/fs.bin > os.img
lihan@debian-10-pa: ~/Desktop/lab3-STUID/lab3$ make play
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write
operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
```

2.2 开启中断嵌套（选做）：



其中命令行输出的是 `syscallFork` 函数中拷贝内存时循环变量 `j` 的值，实际上测试时也是在该循环中手动模拟时钟中断，可见当循环结束即经历多次时钟中断后代码将继续正确运行。

3. 代码修改 (框架代码中TODO in lab3部分)

3.1 完成库函数

在 lab3/lib/syscall.c 中调用 syscall 完善如下函数:

fork , exec , sleep , exit .

3.2 时间中断处理

完成时钟中断功能，相关函数为位于

lab3/kernel/kernel/irqHandle.c 中的 timerHandle。

3.3 系统调用例程

完成四个系统调用函数，即位于

lab3/kernel/kernel/irqHandle.c 中的 syscallFork ,
syscallSleep , syscallExit 和 syscallExec ; 此外完成
syscallExec 之前还需完成位于 lab3/kernel/kernel/kvm.c
中的 loadElf 函数以加载文件到内存。

3.4 中断嵌套

这一部分主要利用进程控制块 `ProcessTable` 中的 `prevStackTop` , 以保存待恢复的栈顶信息。相关代码主要涉及 `lab3/kernel/kernel/irqHandle.c` 中的 `irqHandle` 和 `timerHandle` 中对 `prevStackTop` 的使用。

4.思考&心得体会

1. 在完成 `syscallFork` 函数时, 起初虽然知道各个段寄存器的值不能直接从父进程复制, 但是完全弄不清楚该如何确定它们的值; 后来参照 `initProc` 中对 `proc[1]` 的初始化, 发现如下代

码:

```
pcb[1].regs.ss = USEL(4);
pcb[1].regs.esp = 0x100000;
asm volatile("pushfl");
asm volatile("popl %0":"=r"(pcb[1].regs.eflags));
pcb[1].regs.cs = USEL(3);
pcb[1].regs.eip = loadUMain();
pcb[1].regs.ds = USEL(4);
pcb[1].regs.es = USEL(4);
pcb[1].regs.fs = USEL(4);
pcb[1].regs.gs = USEL(4);
```

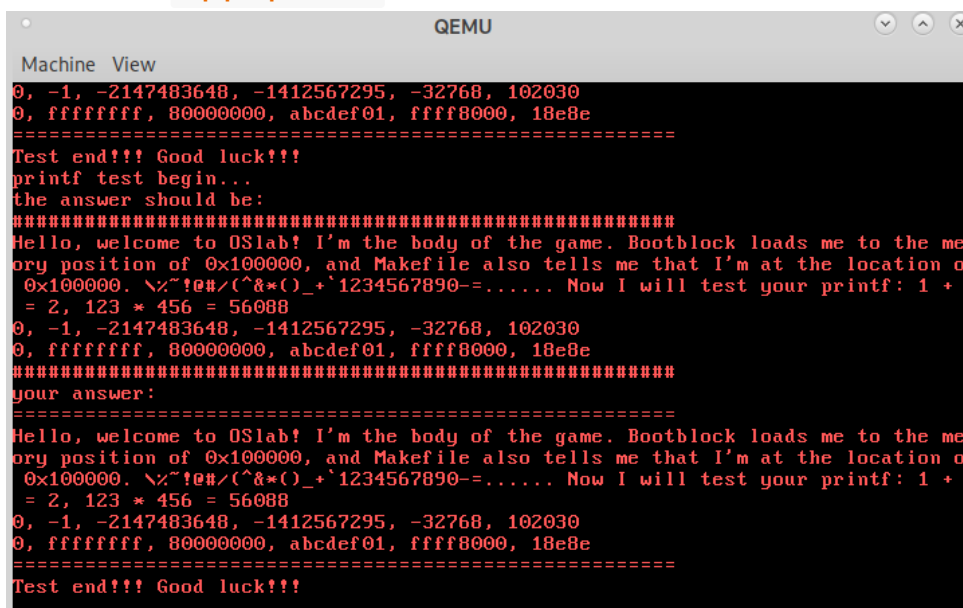
查看 `USEL` 的宏定义之后发现传入的参数即为段选择子的 `index` , 关于上述代码中 `USEL` 的参数 3 和 4 , 是因为: GDT表中第 0 个表项保留不使用, 第 1, 2 个表项用于 `proc[0]` 即内核进程, 故 `proc[1]` 从第 3, 4 个表项开始; 且除了代码段寄存器之外, 其他段寄存器的在同一个表项中, 这应该是由于其他段是重叠的。由此便可确定在 `syscallFork` 函数中, 对于 `proc[i]` 的各个段寄存器应按如下方式进行赋值:

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

2. 在实现 `loadElf` 时出现了一些bug导致程序无法正确运行, 经仔细检查后发现是由于一开始直接把文件加载到了对应的物理地址, 之后再将其需要加载的段加载到对应的物理地址, 而这两个地址都是用的传入的参数 `physAddr` , 于是产生了错误——在加载某个段到 `physAddr` 后对文件产生了覆盖, 可能导致

后续文件内容出错，故而程序无法正确运行。所以应该先把文件加载到其他地址，然后将需要加载的段加载到 `physAddr` 上。

3. 实现 `syscallExec` 后发现无法正确读取通过 `TrapFrame` 传入的字符（即文件名），后来发现是由于数据存储在用户态的数据段中，而当前位于内核态，故需要进行切换再读取 `TrapFrame` 参数，且读取后在将值复制到另一个遍历前应该切换回内核态，否则数据依然在用户态中而无法正确访问。
4. 一开始运行程序时得到结果时结果基本正确，但是运行了两次用户程序 `app_print`：



```
Machine View
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
=====
Test end!!! Good luck!!!
printf test begin...
the answer should be:
=====
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \x~!@#/(^&*)_+'1234567890-=..... Now I will test your printf: 1 + 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
=====
your answer:
=====
Hello, welcome to OSlab! I'm the body of the game. Bootblock loads me to the mem
ory position of 0x100000, and Makefile also tells me that I'm at the location of
0x100000. \x~!@#/(^&*)_+'1234567890-=..... Now I will test your printf: 1 + 1
= 2, 123 * 456 = 56088
0, -1, -2147483648, -1412567295, -32768, 102030
0, ffffffff, 80000000, abcdef01, ffff8000, 18e8e
=====
Test end!!! Good luck!!!
```

后来发现是由于 `syscallExec` 在成功加载并执行文件后并不需要返回（只有在加载文件失败的情况下才需要返回原始进程），并且由于原进程已经被替换掉，只能通过 `exit` 退出。

5. 本次实验使本人对时间中断处理以及进程切换的过程有了更加清晰的认识，同时也意识到在某些情况下进行用户态和内核态的切换的必要性；当然也认识到了在处理进程间切换时理解其内在逻辑、堆栈变化等的重要性，不然会出现很多合乎常理却又意想不到的bug。