

# OS Lab3 Report

---

## 1. 个人信息

---

姓名：余帅杰

学号：181860077

邮箱：[3121416933@qq.com](mailto:3121416933@qq.com)

## 2. 实验进度

---

我完成了所有的内容

成功的输出ping pong以及后续加载print进程

实现了中断嵌套

### 实验内容

1. 完成库函数
2. 处理时钟中断
3. 完成系统调用例程
4. 实现中断嵌套

### 最终的结果：

1. 成功的输出pingpong
2. 成功的加载elf文件并运行
3. 完成了中断嵌套

### 代码修改位置

1. irqHandle.c里的timerHandle()
2. irqHandle.c的syscallExec()
3. irqHandle.c的syscallFork()
4. irqHandle.c的syscallSleep()
5. irqHandle.c的syscallExit()
6. syscall.c里对应上述的四个函数
7. kvm.c里的loadElf函数

## 3. 实验思路

---

### 实现库函数

库函数可以看作系统的调用的前一等级的封装

库函数就是在调用系统调用也就是irq\_Handle，所以这里只需要小心的设置好参数传递的顺序。

接下来的工作就交给系统调用了

### 处理时钟中断

时钟中断有若干的功能

1. 对正在运行的进程已用时间片加一
2. 对处于阻塞状态的进程休眠时间减一
3. 判断当前进程是否时间片用完以及是否处于阻塞或者是死亡状态，或者是当前运行IDLE
4. 若上述条件成立则进入调度阶段，寻找RUNABLE的进程进行替换
5. 替换的核心思路就是换内核栈的栈顶信息，然后利用iret从内核态顺利返回用户态实现切换

对于IDLE的考虑就是只要当前有程序可运行就运行可运行程序，否则进入IDLE

也就是一个等待程序，等待下一个时钟中断被唤醒并做同样的检查（找可运行程序）

## 系统调用实现

### Fork

fork是从当前进程拷贝出新的进程

首先是寻找dead状态的进程块进行替换，没有的则fork失败

找到之后应该拷贝数据段和代码段的内容，然后拷贝部分寄存器的值eflags, eip, cs

还有一部分的值直接计算得到，最后设置各自的返回值，返回值的设置是通过pcb块的里的tf的eax设置的，到了这里fork就实现好了

### sleep

sleep的处理思路比较简单，就是通过之前约定好的参数位置，得到了对应的睡眠时间的参数

然后设置好值之后设置当前进程为阻塞，然后调用时钟中断，就会实现进程新的调度

### exit

exit是退出进程，也就是设置当前的进程为dead，然后调用时钟中断就可以了

时钟中断会进行新的进程的调度

### exec

这个系统调用是为了加载新的程序替换原来的程序

实现相对的复杂，一个是对输入的字符串指针进行处理，然后传入loadelf函数对文件进行解析

在loadelf函数里要负责打开和读取文件，还需要拷贝对应的信息

读入字符串的功能是参考了sysprint里的，由于这个时候是用户态转向内核态，原先的字符串指针所指的位置并不是所需要的位置，所以需要数据段的选择子进行处理，然后在进行进一步的读取。

得到的字符串作为地址传入了loadelf，这里就调用先前熟悉的readInode读取文件的信息，然后进一步的基于信息去读取文件。这里设置了一个缓冲数组，读入数组，然后开始根据elf头的内容决定是否加载和加载的位置，然后拷贝即可

最后返回的时候需要设置对应的entry，如果load的过程顺利，那么就可以直接修改eip实现程序的替换

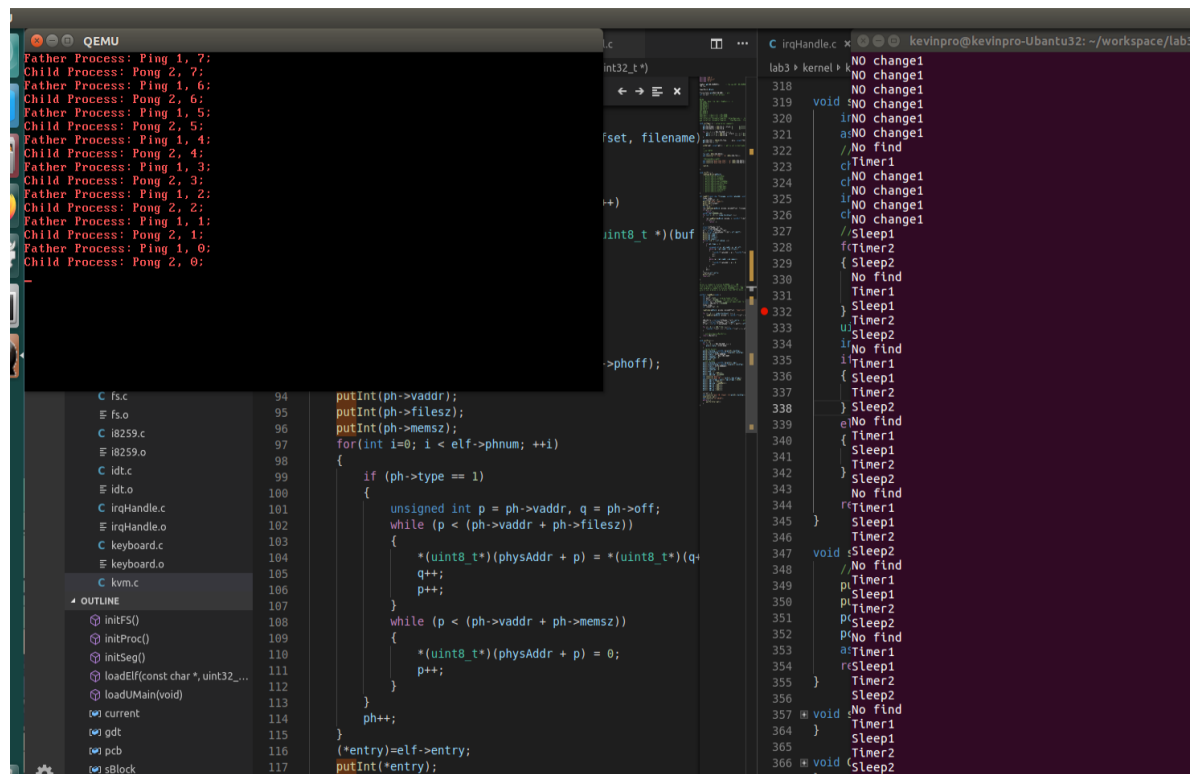
## 中断嵌套

中断嵌套就是在fork里，fork已经是在中断的状态下了，那么此时进行时钟中断并进行处理

## 4. 实验结果

### pingpong

右边是实验过程中的调试输出（基本的语义就是在执行sleep或者是timehandle的时候打印出操作的信息，这个在初期实现的时候调错帮助很大）

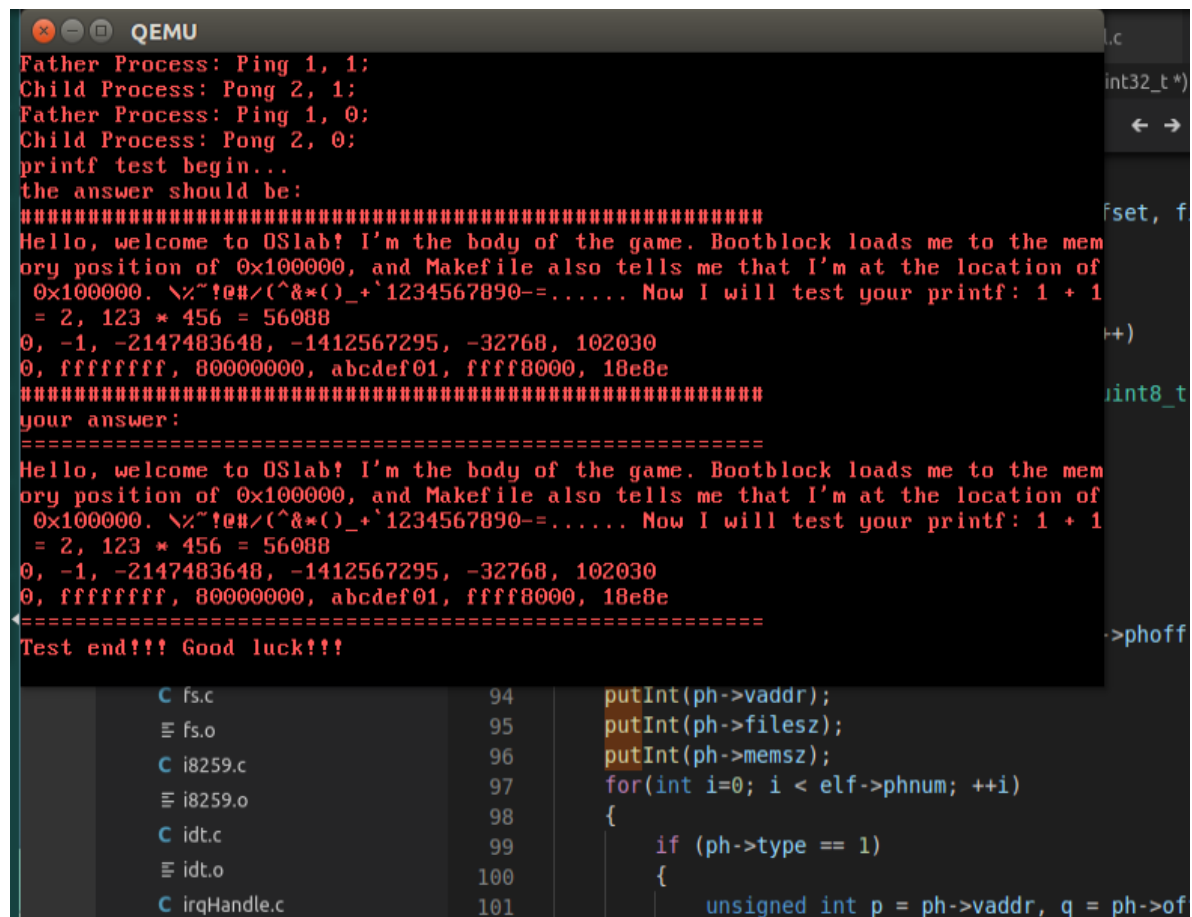


```
Father Process: Ping 1, 7;
Child Process: Pong 2, 7;
Father Process: Ping 1, 6;
Child Process: Pong 2, 6;
Father Process: Ping 1, 5;
Child Process: Pong 2, 5;
Father Process: Ping 1, 4;
Child Process: Pong 2, 4;
Father Process: Ping 1, 3;
Child Process: Pong 2, 3;
Father Process: Ping 1, 2;
Child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

C fs.c 94 putInt(ph->vaddr);
C fs.o 95 putInt(ph->filesz);
C i8259.c 96 putInt(ph->memsz);
C i8259.o 97 for(int i=0; i < elf->phnum; ++i)
C idt.c 98 {
C idt.o 99     if (ph->type == 1)
C irqHandle.c 100     {
C irqHandle.o 101         unsigned int p = ph->vaddr, q = ph->off;
C keyboard.c 102         while (p < (ph->vaddr + ph->filesz))
C keyboard.o 103         {
C kvm.c 104             *(uint8_t*)(physAddr + p) = *(uint8_t*)(q);
C kvm.c 105             q++;
C kvm.c 106             p++;
C kvm.c 107         }
C kvm.c 108         while (p < (ph->vaddr + ph->memsz))
C kvm.c 109         {
C kvm.c 110             *(uint8_t*)(physAddr + p) = 0;
C kvm.c 111             p++;
C kvm.c 112         }
C kvm.c 113     }
C kvm.c 114     ph++;
C kvm.c 115 }
C kvm.c 116 (*entry)=elf->entry;
C kvm.c 117 putInt(*entry);

lab3 > kernel > k
318 NO change1
319 NO change1
320 void NO change1
321 NO change1
322 NO change1
323 Timer1
324 NO change1
325 NO change1
326 NO change1
327 Sleep1
328 Timer2
329 Sleep2
330 No find
331 Timer1
332 Sleep1
333 Timer2
334 Sleep2
335 No find
336 Timer1
337 Sleep1
338 Timer2
339 Sleep2
340 No find
341 Timer1
342 Sleep1
343 Timer2
344 Sleep2
345 No find
346 Timer1
347 Sleep1
348 Timer2
349 Sleep2
350 No find
351 Timer1
352 Sleep1
353 Timer2
354 Sleep2
355 No find
356 Timer1
357 Sleep1
358 Timer2
359 Sleep2
360 No find
361 Timer1
362 Sleep1
363 Timer2
364 Sleep2
365 No find
366 Timer1
367 Sleep1
368 Timer2
369 Sleep2
370 No find
371 Timer1
372 Sleep1
373 Timer2
374 Sleep2
375 No find
376 Timer1
377 Sleep1
378 Timer2
379 Sleep2
380 No find
381 Timer1
382 Sleep1
383 Timer2
384 Sleep2
385 No find
386 Timer1
387 Sleep1
388 Timer2
389 Sleep2
390 No find
391 Timer1
392 Sleep1
393 Timer2
394 Sleep2
395 No find
396 Timer1
397 Sleep1
398 Timer2
399 Sleep2
400 No find
401 Timer1
402 Sleep1
403 Timer2
404 Sleep2
405 No find
406 Timer1
407 Sleep1
408 Timer2
409 Sleep2
410 No find
411 Timer1
412 Sleep1
413 Timer2
414 Sleep2
415 No find
416 Timer1
417 Sleep1
418 Timer2
419 Sleep2
420 No find
421 Timer1
422 Sleep1
423 Timer2
424 Sleep2
425 No find
426 Timer1
427 Sleep1
428 Timer2
429 Sleep2
430 No find
431 Timer1
432 Sleep1
433 Timer2
434 Sleep2
435 No find
436 Timer1
437 Sleep1
438 Timer2
439 Sleep2
440 No find
441 Timer1
442 Sleep1
443 Timer2
444 Sleep2
445 No find
446 Timer1
447 Sleep1
448 Timer2
449 Sleep2
450 No find
451 Timer1
452 Sleep1
453 Timer2
454 Sleep2
455 No find
456 Timer1
457 Sleep1
458 Timer2
459 Sleep2
460 No find
461 Timer1
462 Sleep1
463 Timer2
464 Sleep2
465 No find
466 Timer1
467 Sleep1
468 Timer2
469 Sleep2
470 No find
471 Timer1
472 Sleep1
473 Timer2
474 Sleep2
475 No find
476 Timer1
477 Sleep1
478 Timer2
479 Sleep2
480 No find
481 Timer1
482 Sleep1
483 Timer2
484 Sleep2
485 No find
486 Timer1
487 Sleep1
488 Timer2
489 Sleep2
490 No find
491 Timer1
492 Sleep1
493 Timer2
494 Sleep2
495 No find
496 Timer1
497 Sleep1
498 Timer2
499 Sleep2
500 No find
```

### exec elf



```
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. \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!!!

C fs.c 94 putInt(ph->vaddr);
C fs.o 95 putInt(ph->filesz);
C i8259.c 96 putInt(ph->memsz);
C i8259.o 97 for(int i=0; i < elf->phnum; ++i)
C idt.c 98 {
C idt.o 99     if (ph->type == 1)
C irqHandle.c 100     {
C irqHandle.o 101         unsigned int p = ph->vaddr, q = ph->of
```

## 中断嵌套

由于完全的中断太慢了。设置了个模运算（模数少了个0，应该是16次）（原版的也可以运行）

```
    if(pos==-1)
    {
        pcb[current].regs.eax = -1;
        return;
    }
    else
    {
        putString("Fork");
        putInt(pos);
        enableInterrupt();
        for (int j = 0; j < 0x100000; j++)
        {
            *(uint8_t*)(j + (pos + 1) * 0x100000) = *(uint8_t*)(j + (current + 1) * 0x100000);
            if(j%0x10000==0)
                asm volatile("int $0x20"); //XXX Testing irqTimer during syscall
        }
        disableInterrupt();
        /*
        for (int j = 0; j < 0x100000; j++)
        {
            *(uint8_t*)(j + (pos + 1) * 0x100000) = *(uint8_t*)(j + (current + 1) * 0x100000);
        }
        */
        for(int i=0;i<MAX_STACK_SIZE;i++)
```

## 5.自由报告

### 问题1

为什么在exec的时候都已经提供了文件名指针了，还要做处理？

因为二者所处的特权级不同，需要对数据段进行设置之后再进行读取，才能得到正确的结果

### 问题2

为什么出现了父进程sleep的时候进入子进程，子进程也执行了fork得到了第三个进程？

因为部分寄存器设置错误，导致子进程重复执行了fork，出现了多个子进程

### 问题3

为什么fork的时候返回值一直是0，无法标识出是父进程

因为fork函数写错了，一直返回的0。应该返回sysfork返回的pid作为最后的结果