

第7课 真正的多任务实现

《跟着瓦利哥学写OS》

联系方式

- 自己动手写操作系统QQ群：82616767。
- 申请考试邮箱：sangwf@gmail.com
- 所有课程的代码都在Github：
- <https://github.com/sangwf/walleclass>

问题出在哪里?

```
11 bits 32 ;这是32位的指令
12
13 mov eax, DATA_SEL ;先设置数据段
14 mov ds, eax
15 mov gs, eax
16
17 ;初始化栈
18 lss esp, [init_stack]
```


真相大白！

- 使用`gs`之前，都给它专门赋值过。似乎`gs`会被某个地方隐含的调用到。
- 问题出在VMWare的BIOS实现，当开始执行`0x7C00`的内核代码时，`gs`的值为`0xf000`，这样在`pop gs`时，CPU会检测`gs`的有效性，结果超出了GDT的有效索引范围，触发Tripple Fault，导致系统重启。

思考

- Linux下，是怎么创建一个新进程的？
- 答案：`fork`系统调用
- `$ man fork`

fork说明

NAME

fork -- create a new process

SYNOPSIS

```
#include <unistd.h>
```

```
pid_t  
fork(void);
```


fork说明

DESCRIPTION

Fork() causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process) except for the following:

- o The child process has a unique process ID.
- o The child process has a different parent process ID (i.e., the process ID of the parent process).
- o The child process has its own copy of the parent's descriptors. These descriptors reference the same underlying objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an `lseek(2)` on a descriptor in the child process can affect a subsequent read or write by the parent. This descriptor copying is also used by the shell to establish standard input and output for newly created processes as well as to set up pipes.
- o The child processes resource utilizations are set to 0; see `setrlimit(2)`.

fork说明

RETURN VALUES

Upon successful completion, **fork()** returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and the global variable errno is set to indicate the error.

实验1：

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main(int argc, char* argv[])
5 {
6     if (fork() == 0) {
7         printf("I'm son.\n");
8     } else {
9         printf("I'm dad.\n");
10    }
11    return 0;
12 }
```


实验1：

```
sangwfdembp:cls7 sangwf$ gcc forktest.c -o forktest  
sangwfdembp:cls7 sangwf$ ./forktest  
I'm dad.  
I'm son.
```


一个进程的构成

- GDT中的两个描述符：TSS和LDT
- TSS：任务状态段，包括各种寄存器信息
- LDT：局部描述符表，包括代码段、数据段的信息
- 代码段
- 数据段
- 内核栈
- 用户栈

fork的实现原理（简化）

- 1, 响应系统中断。
- 2, 在GDT中创建新的TSS和LDT的描述符
- 3, 创建TSS和LDT, 基本是复制父进程的。
- 3, 代码段和数据段都指向父进程的。
- 4, 分别设置父进程的TSS的eax为子进程PID, 子进程的eax为0。
- 5, 返回（父进程继续正常执行, 等到时钟中断时, 就会将子进程参与调度）

实验2：

- 实现一个伪的fork，子进程的TSS事先已经创建好，只需要修改EIP、EAX。
- 只能fork一次。
- 父进程打印A，子进程打印B。

关键代码1：

```
418 task0:
419     int 0x82 ;fork
420     cmp eax, 0 ;eax存放 fork的返回值， 为 0时表示子进程， 非 0时表示父进程
421     je .do_son
422     .do_father:
423     mov al, 'A'
424     int 0x81
425     mov ecx, 0x1fffffff ;delay a while
426     .t0:
427     loop .t0
428     jmp .do_father
429     .do_son:
430     mov al, 'B'
431     int 0x81
432     mov ecx, 0x1fffffff ;delay a while
433     .t1:
434     loop .t1
435     jmp .do_son
436     jmp task0 ;never arrived
```


关键代码2：

[illegible]

关键代码3：

```
180     mov [task1_eip], eax
181     mov dword [task1_eax], 0 ;对子进程，设置 eax 为 0
182
183     jmp .fork_finish
184     .fork_fail:
185     ;print something
186     mov al, 'F'
187     call func_write_char
188     .fork_finish:
189     pop eax
190     mov eax, [process_count] ;对父进程，设置 eax 为 process_count
191     pop ds
192     sti
193     iret
```


关键代码4：

```
129 int_timer: ;时钟中断处理函数
130     push ds
131     push eax
132
133     mov al, 0x20
134     out 0x20, al
135
136     mov eax, DATA_SEL
137     mov ds, ax
138
139     ;若只有一个进程，不需要切换
140     mov eax, [process_count]
141     cmp eax, 1
142     je .switch_finish
143
144     mov eax, 1
145     cmp [current], eax
146     je .switch_0
147     mov dword [current], 1
```


关键代码5：

```
148      jmp TSS1_SEL: 0 ;注意，执行这句后，马上保存了当前现场，
149                          ;并跳转到了任务1之前的现场去执行，也就
150                          ;下次切换回来时，是执行了下一句。是在内
151                          ;核态时，被切换出去了。
152      jmp .switch_finish
153      .switch_0:
154      mov dword [current], 0
155      jmp TSS0_SEL: 0
156      .switch_finish:
157      pop eax
158      pop ds
159      iret
```


运行结果:



The screenshot shows a terminal window titled "walleclass" with a standard macOS-style title bar (red, yellow, green buttons). Below the title bar is a toolbar with icons for pausing, copying, and a settings/next button. The main content area is filled with a dense, repeating pattern of red and green 'A' and 'B' characters, creating a complex, textured visual effect. The pattern appears to be a form of data visualization or a specific output from a program.

思考

- TSS (Task State Segment) 和PCB (Process Control Block) 是什么关系？

谢谢！