第8课 进入C语言的世界

《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

思考

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

TSS

- 寄存器状态数据

PCB

- 进程标识数据（进程ID等）
- 进程状态数据（等价于TSS的寄存器状态数据）
- 进程控制数据（进程状态，优先级等）

Linux的进程切换

- 并没有使用TSS，而是采用PCB，使用软件层面切换寄存器的状态数据。这样一是提升性能，二是便于移植（如AMD64不支持硬件进程切换）。

为什么要使用C语言开发OS？

- 提升开发效率
- 便于移植
- 便于代码理解

C代码：

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

int main(int argc, char* argv[])
{
    if (fork() == 0) {
        printf("I'm son.\n");
    } else {
        printf("I'm dad.\n");
    }
    return 0;
}
```

汇编代码：

```
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, 0x1fffff ;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, 0x1fffff ;delay a while
433     .t1:
434     loop .t1
435     jmp .do_son
436     jmp task0 ;never arrived
```

如何将C代码与汇编链接到一起？

- 方案1：都编译为中间码，然后链接到一起。
- 方案2：将C代码编译为ELF格式，再抽取出机器码，使用脚本将其与其他机器指令放到一起。（采用）

代码1：

```
1  #include "system.h"
2
3  int my_entrance()
4  {
5      if (fork_syscall() == 0) {
6          // son
7          while(1) {
8              print_char('S');
9          }
10     } else {
11         // dad
12         while(1) {
13             print_char('D');
14         }
15     }
16     return 0;
17 }
```

代码2：

```
 1 #ifndef _SYSTEM_H
 2 #define _SYSTEM_H
 3
 4 // 打印字符
 5 #define print_char(character) ({ \
 6 __asm__ volatile ("int $0x81"::"a" (character) ); \
 7 })
 8
 9 int fork_syscall();
10
11 #endif // _SYSTEM_H
```

代码3：

```
1 int fork_syscall()
2 {
3     int ret = -1;
4     __asm__ volatile ("int $0x82": "=a" (ret): );
5
6     return ret;
7 }
```

编译+链接

```
3 i586-pc-linux-gcc -c usermode.c system.c
4 i586-pc-linux-ld -Ttext=2200 -emy_entrance usermode.o system.o -o usermode
```

可执行链接格式

- ELF：Executable and Linkable Format。在UNIX/Linux下，编译或链接后的文件为ELF格式。（下节课详细介绍）

ld链接器

- -Ttext = 2200：指定机器指令的起始地址是0x2200。
- -emy_entrance：指定代码的入口函数是my_entrance，而不是main。

readelf工具

```
sangwfdembp:cls8 sangwf$ i586-pc-linux-readelf -h usermode
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x2200
```

objdump工具

```
sangwfdembp:cls8 sangwf$ i586-pc-linux-objdump -d usermode

usermode:     file format elf32-i386


Disassembly of section .text:

00002200 <my_entrance>:
    2200:       55                      push   %ebp
    2201:       89 e5                   mov    %esp,%ebp
    2203:       83 ec 08                sub    $0x8,%esp
    2206:       e8 19 00 00 00          call   2224 <fork_syscall>
    220b:       85 c0                   test   %eax,%eax
    220d:       75 09                   jne    2218 <my_entrance+0x18>
    220f:       b8 53 00 00 00          mov    $0x53,%eax
    2214:       cd 81                   int    $0x81
    2216:       eb f7                   jmp    220f <my_entrance+0xf>
    2218:       b8 44 00 00 00          mov    $0x44,%eax
    221d:       cd 81                   int    $0x81
    221f:       eb f7                   jmp    2218 <my_entrance+0x18>
    2221:       66 90                   xchg   %ax,%ax
    2223:       90                      nop
```

objdump工具（续）

```
00002224 <fork_syscall>:
    2224:       55                          push   %ebp
    2225:       89 e5                       mov    %esp,%ebp
    2227:       83 ec 10                    sub    $0x10,%esp
    222a:       c7 45 fc ff ff ff ff        movl   $0xffffffff,-0x4(%ebp)
    2231:       cd 82                       int    $0x82
    2233:       89 45 fc                    mov    %eax,-0x4(%ebp)
    2236:       8b 45 fc                    mov    -0x4(%ebp),%eax
    2239:       c9                          leave
    223a:       c3                          ret
```

代码4：

```
450 task0:
451     jmp C_ENTER
452     hlt
453
454 times 8192 - ($ - $$) db 0
455 C_ENTER:
```

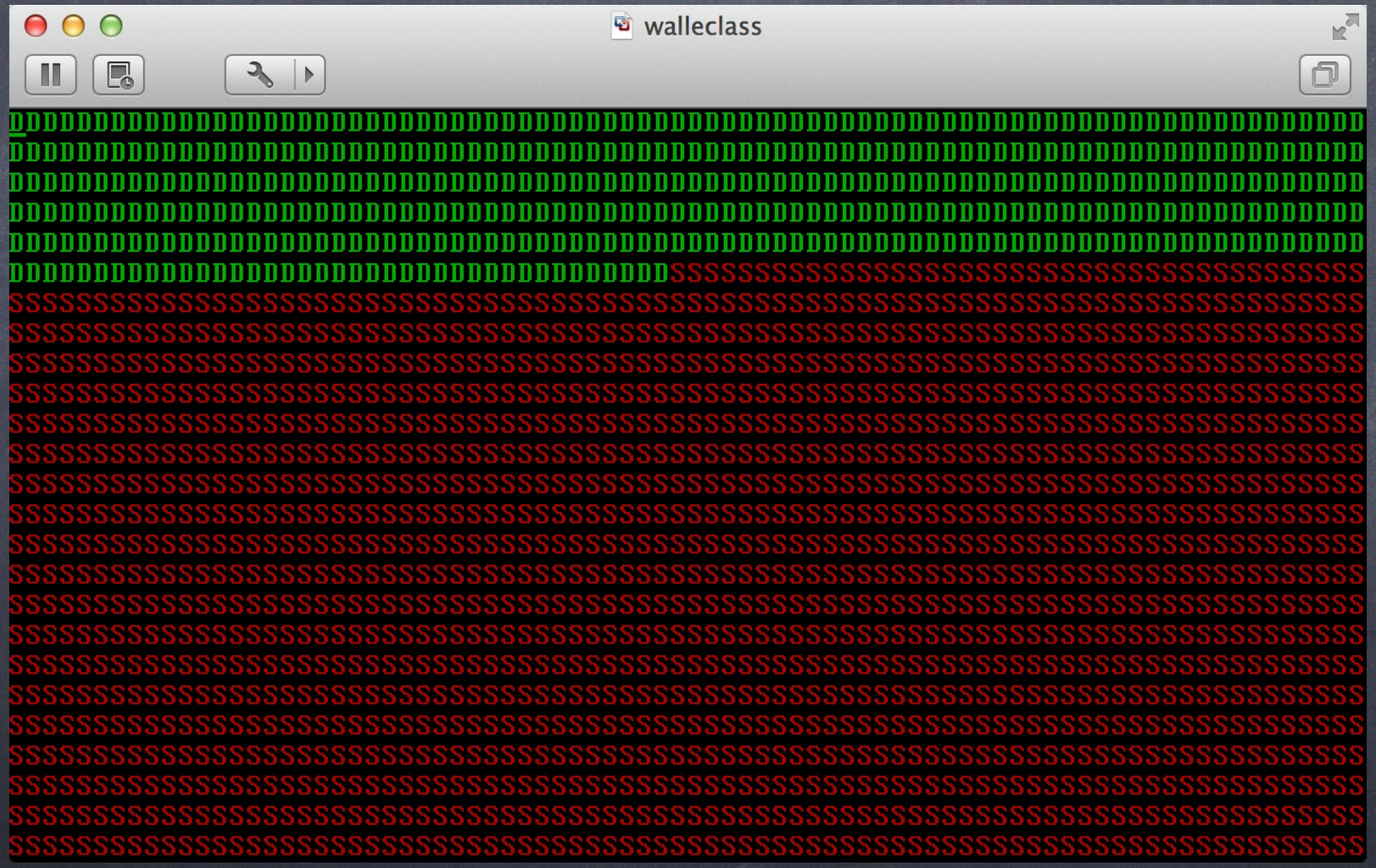
编译与链接指令

```
nasm -f bin boot.s -o boot.bin
nasm -f bin print_ab.s -o print_ab.bin
i586-pc-linux-gcc -c usermode.c system.c
i586-pc-linux-ld -Ttext=2200 -emy_entrance usermode.o system.o -o usermode
i586-pc-linux-objcopy -O binary usermode usermode.bin

cat boot.bin print_ab.bin usermode.bin > system.bin
dd conv=sync if=system.bin of=print_ab.img bs=1440k count=1

rm -f boot.bin print_ab.bin system.bin usermode.o system.o
```

运行结果:



思考

- 操作系统如何执行应用程序的？如双击一个应用，到底发生了什么？

谢谢！