北京科技大学实验报告

学院：计算机与通信工程学院　专业：计算机科学与技术　班级：计 1703

姓名：　张宝丰　　学号：41724081　　实验日期：2019 年　12 月　1 日

实验名称：操作系统实验 1 操作系统启动（2 分）

实验目的：以一个教学型操作系统 EOS 为例，了解操作系统的启动过程，理解操作系统启动后的工作方式;能对核心源代码进行分析;训练分析问题、解决问题以及自主学习能力，逐步达到能独立对小型操作系统的功能进行分析、设计和实现。

实验环境：EOS 操作系统及其实验环境。

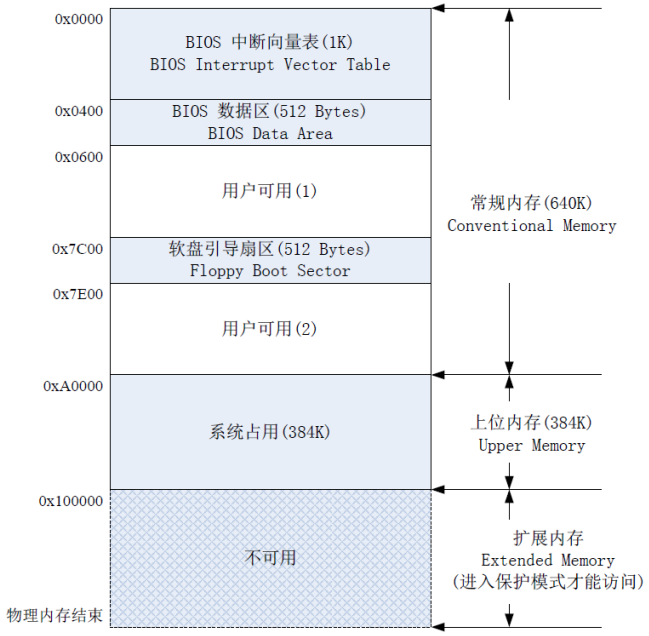
实验内容：

跟踪 EOS 成功启动的全过程，分析相关源代码；查看 EOS 启动后的状态和行为。

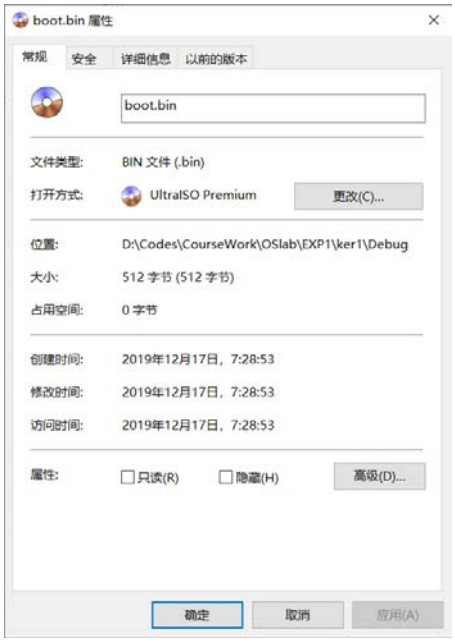
实验步骤：

1）EOS 操作系统启动过程的跟踪与源代码分析

（分析从引导到 EOS 内核加载的相关源代码;简要说明在本部分实验过程中完成的主要工作，包括 BIOS、引导程序、EOS 内核加载程序的跟踪等）



EOS 对于物理内存的功能分配如上图所示（图片来自 EOS 操作系统实验教程），BIOS 在进行完必要的自检和初始化之后，会自动搜索可引导的存储设备，在实验中它会找到虚拟软盘 img，并将其引导区的 boot.asm 引导文件（大小恰好为 512 字节）加载到 0x7C00（软盘引导扇区）。



可以看到，初始时段寄存器 CS 的值为 0xf000，寄存器 IP 的值为 0xfff0

```
Bochs for Windows - Console
Next at t=0
(0) context not implemented because BX_HAVE_HASH_MAP=0
[0xffffffff0] f000:fff0 (unk. ctxt): jmp far f000:e05b         ; ea5be000f0
<bochs:1> sreg
cs:s=0xf000, dl=0x0000ffff, dh=0xff0093ff, valid=1
ds:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ss:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
es:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
fs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
gs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ldtr:s=0x0000, dl=0x0000ffff, dh=0x00008200, valid=1
tr:s=0x0000, dl=0x0000ffff, dh=0x00008b00, valid=1
gdtr:base=0x00000000, limit=0xffff
idtr:base=0x00000000, limit=0xffff
<bochs:2> r
rax: 0x00000000:00000000 rcx: 0x00000000:00000000
rdx: 0x00000000:00000f20 rbx: 0x00000000:00000000
rsp: 0x00000000:00000000 rbp: 0x00000000:00000000
rsi: 0x00000000:00000000 rdi: 0x00000000:00000000
r8 : 0x00000000:00000000 r9 : 0x00000000:00000000
r10: 0x00000000:00000000 r11: 0x00000000:00000000
r12: 0x00000000:00000000 r13: 0x00000000:00000000
r14: 0x00000000:00000000 r15: 0x00000000:00000000
rip: 0x00000000:0000fff0
eflags 0x00000002
id vip vif ac vm rf nt IOPL=0 of df if tf sf zf af pf cf
<bochs:3>
```

接下来在 0x7c00 添加断点：vb 0x0000:0x7c00，再按下 c 继续执行:

```
(0) Breakpoint 4619281, in 0000:7c00 (0x00007c00)
Next at t=16897707
(0) [0x00007c00] 0000:7c00 (unk. ctxt): jmp .+0x006d (0x00007c6f) ; eb6d
```

从上图中看到，第一条指令是 jmp 0x7c6f，字节码是 eb6d。右键 boot.asm，打开生成的列表文件，可以看到 jmp 指令跳转到了一系列初始化操作，如下图所示：

```
                                    Start:
                                        ; 初始化 CPU 的段寄存器为 CS
0000006F 8CC8                           mov ax, cs
00000071 8ED8                           mov ds, ax
00000073 8EC0                           mov es, ax
00000075 8ED0                           mov ss, ax
00000077 31E4                           xor sp, sp
00000079 89E5                           mov bp, sp

                                        ; 初始化屏幕
0000007B B80006                         mov ax, 0x0600      ; AH = 0
0000007E BB0007                         mov bx, 0x0700      ; 黑底白
00000081 31C9                           xor cx, cx          ; 左上角
00000083 BA4F18                         mov dx, 0x184F      ; 右下角
00000086 CD10                           int 0x10
```

这说明 boot.asm 已经被加载到内存中并开始执行。boot.lst 中可以找到要跳转到 loader 程序中执行的指令：

```
jmp    0:LOADER_ORD
```

可以计算得出这条指令的逻辑地址是 0x7d81，在该处加上断点，并按 c 继续执行，可以看到这条指令实际的跳转目标是 0x0000:1000，即 Loader 程序的第一条指令。

```
<bochs:5> vb 0x0000:0x7d81
<bochs:6> c
(0) Breakpoint 4619281, in 0000:7d81 (0x00007d81)
Next at t=18121648
(0) [0x00007d81] 0000:7d81 (unk. ctxt): jmp far 0000:1000          ; ea00100000
```

接下来 Loader 程序会将操作系统内核文件 kernel.dll 加载到内存当中，然后让 CPU 进入保护模式并启用分页机制，最后进入操作系统内核开始执行，在物理地址 0x1513，loader 程序的指令 call dword ptr ds:0x80001117，于是利用 x /lwx ds:0x80001117 查看该地址的指针指向的函数入口地址，得知内核程序入口地址时 0x80017de0。

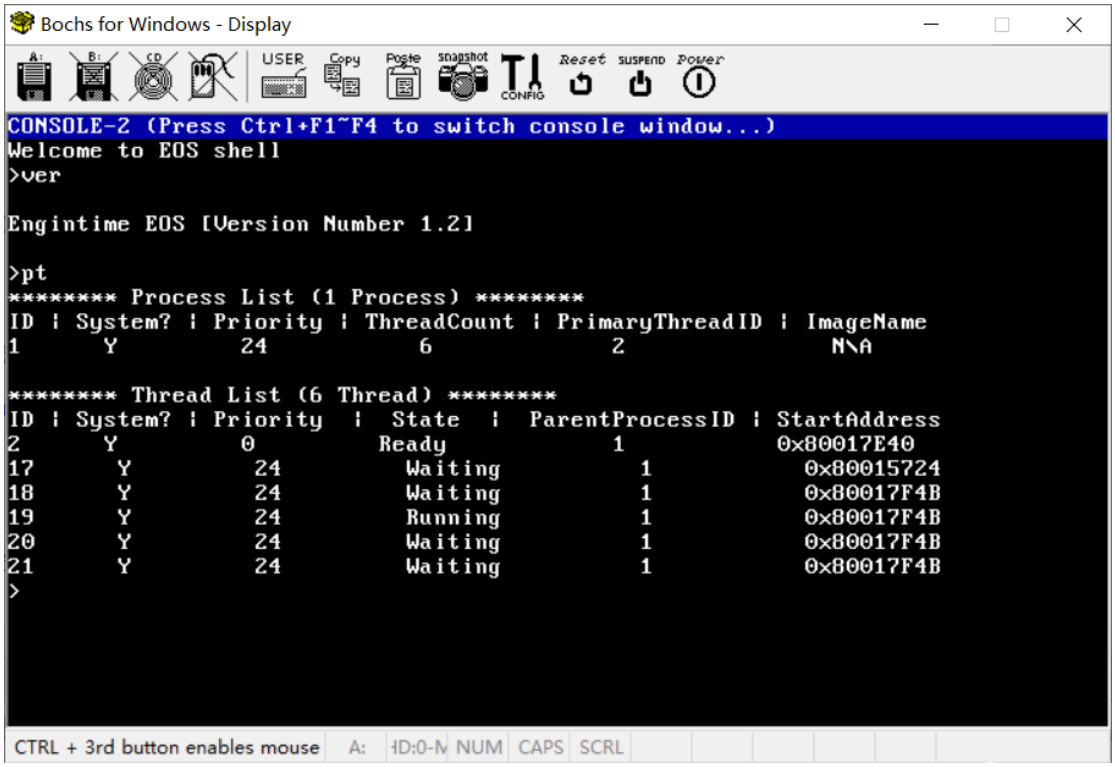
```
<bochs:7> pb 0x1513
<bochs:8> c
(0) Breakpoint 3, 0x0000000080001513 in ?? ()
Next at t=43915631
(0) [0x00001513] 0008:0000000080001513 (unk. ctxt): call dword ptr ds:0x80001117 ; ff1517110080
<bochs:9> x /1wx ds:0x80001117
[bochs]:
0x0000000080001117 <bogus+       0>:    0x80017de0
<bochs:10>
```

接着进行内核调试，查看 start.c 中的 KiSystemStartup 的地址是 0x80017de0，与之前看到的内核程序入口地址相同，说明确实是由 Loader 进入内核的。

名称	值	类型
KiSystemStartup	{void (PVOID)} 0x80017de0 <KiSystemStartup>	void (PVOID)

2）查看 EOS 启动后的状态和行为

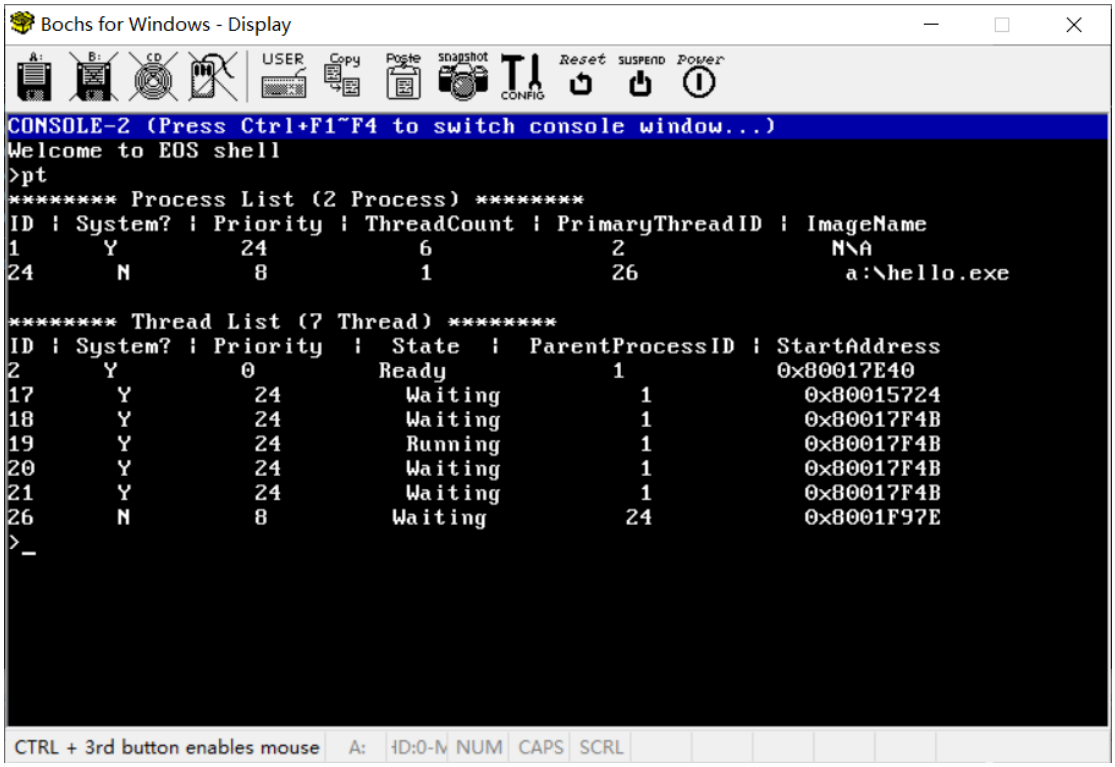
（给出在本部分实验过程中完成的主要工作）



在操作系统启动之后，可以看到只有一个系统进程正在运行，其 ID 为 2 空闲线程正在运行。在 ke/sysproc.c 的第 143 行可以看到，这是一个死循环，它作为空闲线程，在没有任何更高优先级的线程就绪时会不断运行；ID 为 18 的线程是控制台 2 的线程，由于这是在控制台 2 中进行的操作，所以上图中会显示 18 号线程正在运行。

```
138     //
139     // 将当前线程优先级降至最低，当前线程做为空闲线程进入空闲循环。
140     //
141     PsSetThreadPriority(CURRENT_THREAD_HANDLE, 0);
142
143     for(;;) {
144         i++;
145     }
}
```

最后，我在控制台中运行 Hello.exe，打印出下图的进程和线程信息：



从上图可以看到，运行 hello.exe 时，多了一个 ID=24 的非系统进程，这就是 hello.exe 创建的进程，而在线程列表中，也可以看到多了一个 ID=26 的线程，由于已经切换到控制台 2，hello.exe 对应的线程处在 Waiting 状态。

结果分析：

（对本实验所做工作及结果进行分析，包括结合 EOS，在理解的基础上总结操作系统的启动过程以及启动后的工作方式；对 EOS 启动过程的相关问题提出自己的思考；其他需要说明的问题）

操作系统的启动过程：

1. BIOS 进行自检和初始化，之后将 boot 程序加载到内存的物理地址 0x7c00 处，并跳转到那里，将 CPU 控制权交给 boot；

2. boot 程序的大小为 512 字节，其功能是将 loader 程序加载到内存的 0x1000 处，之后跳转到 0x1000，运行 loader；

3. loader 程序负责将系统内核文件 kernel.dll 加载到内存中，之后将 CPU 控制权交给操作系统内核，上述过程都已经在实验中得到验证。

在操作系统启动之后，系统会默认创建一个 ID=1 的系统进程，并且会创建 ID=0 的空闲线程、ID=17 的键盘响应线程、以及四个控制台线程。系统线程的优先级为 24，而用户程序的 hello.exe 创建的线程优先级为 8，低于系统线程。