

# 合肥工业大学

## 操作系统实验报告

实验题目	实验 2 操作系统的启动
学生姓名	孙淼
学 号	2018211958
专业班级	计算机科学与技术 18-2 班
指导教师	田卫东
完成日期	11.07

## 1. 实验目的和任务要求

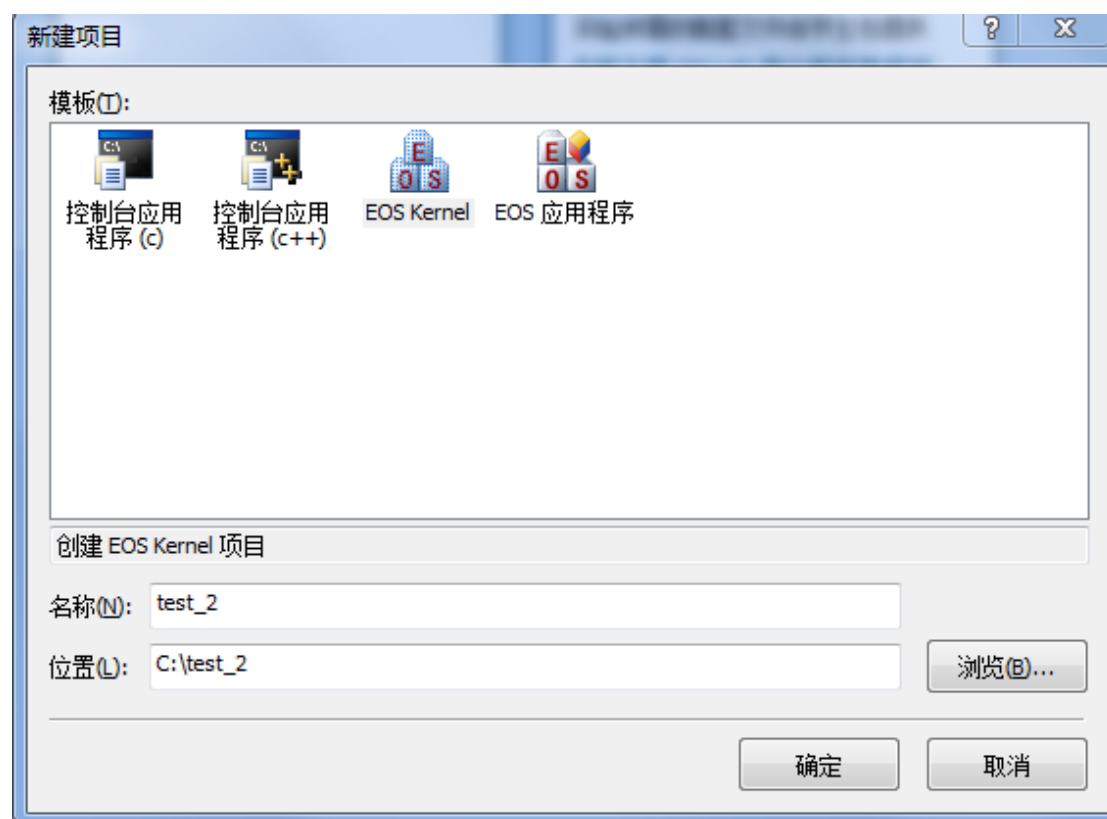
- 跟踪调试 EOS 在 PC 机上从加电复位到成功启动的全过程，了解操作系统的启动过程。
- 查看 EOS 启动后的状态和行为，理解操作系统启动后的工作方式。

## 2. 实验原理

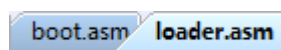
实验原理涉及 EOS 操作系统的启动过程，汇编语言的相关知识，NASM 汇编代码的相关知识以及 Bochs 虚拟机软件的特点，Bochs 的调试命令。

## 3. 实验内容

新建一个 EOS Kernel



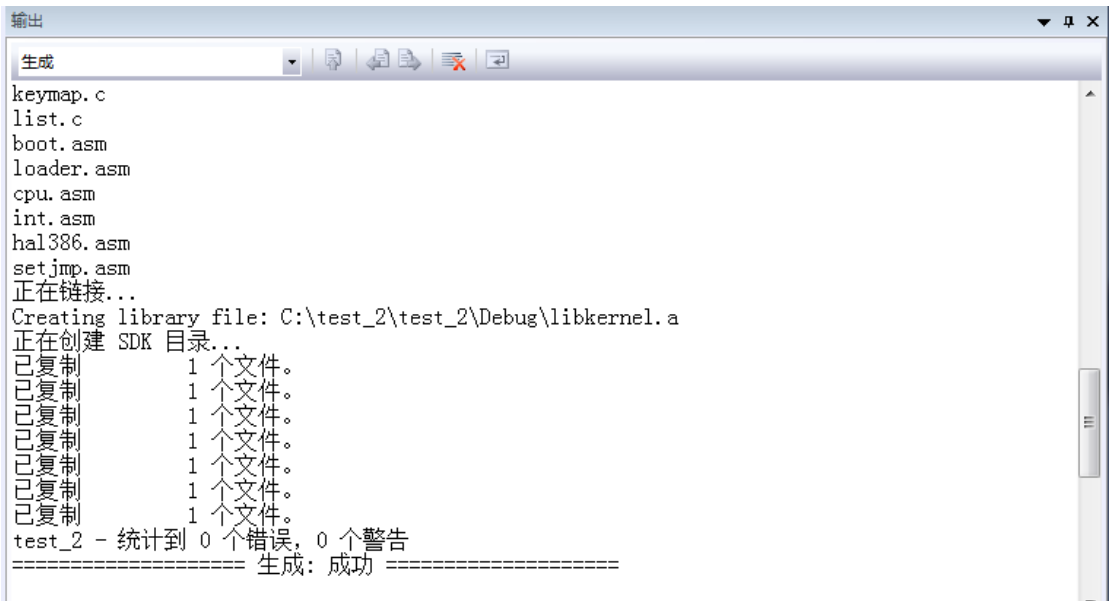
在“项目管理器”窗口中打开 boot/boot.asm 和 boot/loader.asm 两个汇编文件。boot.asm 是软盘引导扇区程序的源文件，loader.asm 是加载程序的源文件。简单阅读一下这两个文件中的 NASM 汇编代码和注释。



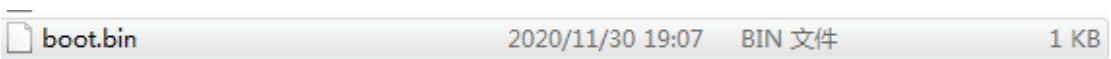
按 F7 生成项目。

生成完成后，使用 Windows 资源管理器打开项目文件夹中的 Debug 文件夹。找到由 boot.asm 文件生成的软盘引导扇区程序 boot.bin 文件，该文件的大小一定为 512 字节（与软盘引导扇区的大小一致）。找到由 loader.asm 生成的加载程序 loader.bin 文件，记录下此文件的大小 1566 字节，在下面的实验过程中会用到。找到由其它源文件生成的 EOS 操作系统内核文件 kernel.dll。

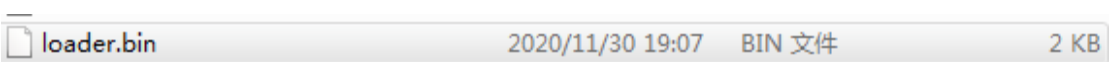
情况如下：



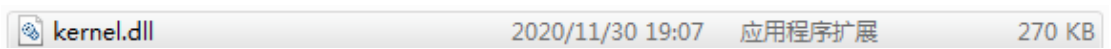
并且找到软盘引导扇区程序如下



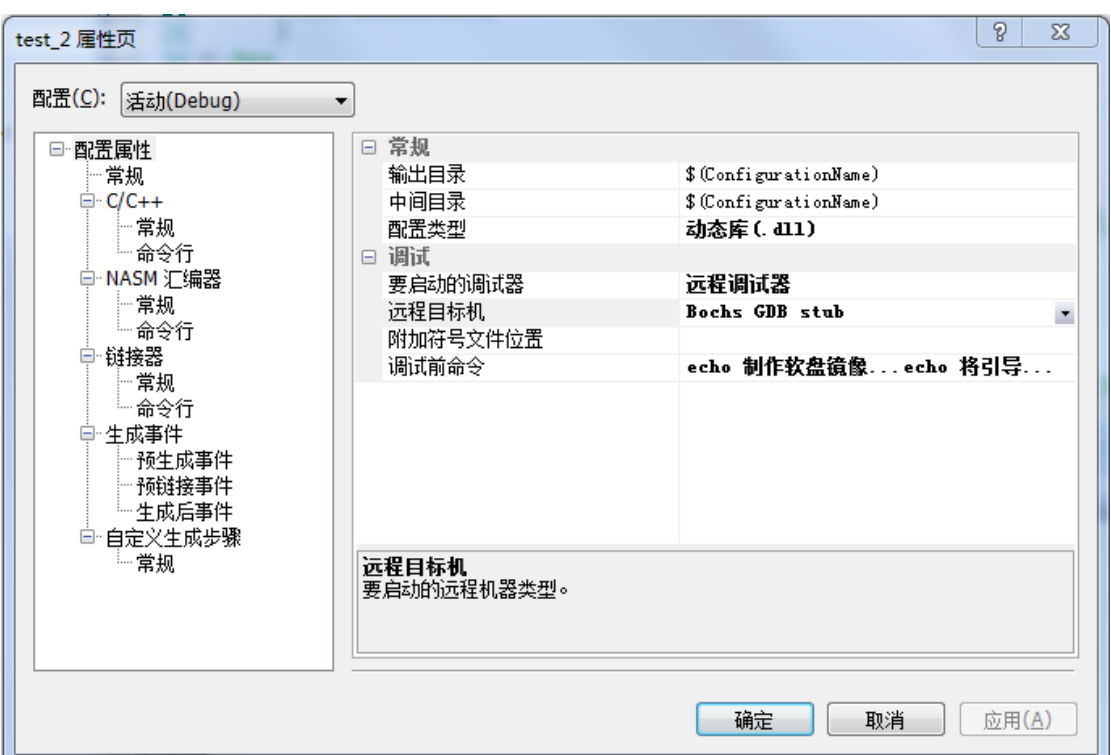
Loader.asm 生成的加载程序

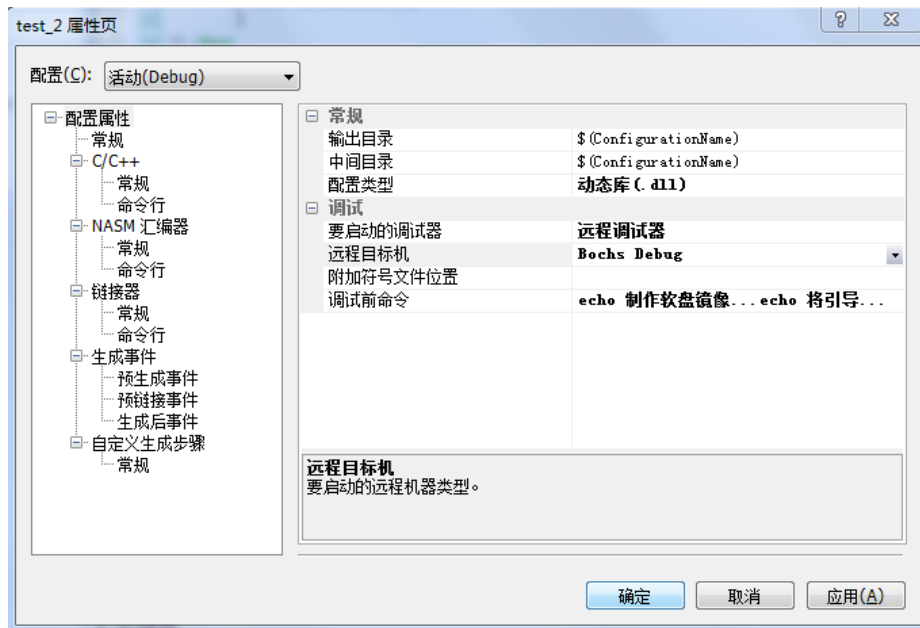


EOS 操作系统内核文件

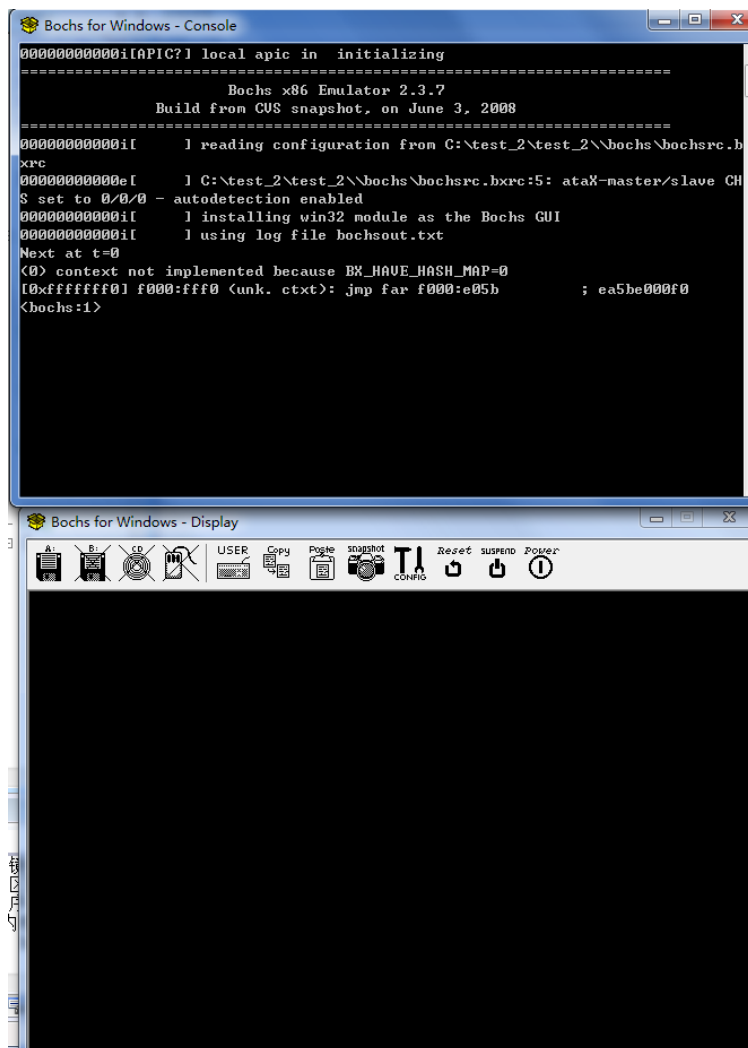


下面使用 Bochs Debug 做目标远程机

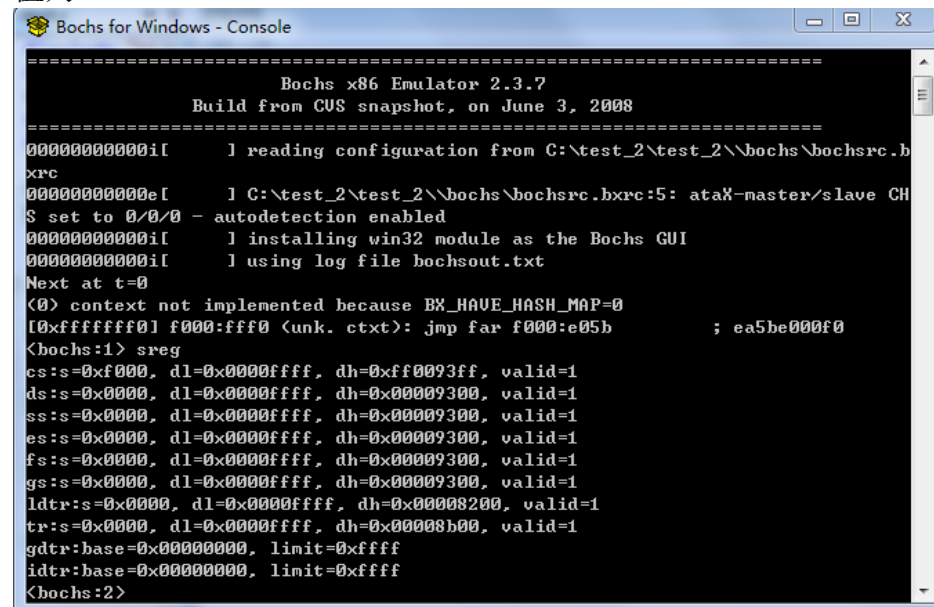




按 F5 调试, 此时会弹出两个 Bochs 窗口。标题为 “Bochs for windows-Display” 的窗口相当于计算机的显示器, 用于显示操作系统的输出。标题为 “Bochs for windows-Console” 的窗口是 Bochs 的控制台, 用来输入调试命令, 输出各种调试信息。



在 Console 窗口中输入调试命令 `sreg` 后按回车, 显示当前 CPU 中各个段寄存器的值, 如图 10-2。其中 CS 寄存器信息行中的“s=0xf000”表示 CS 寄存器的值为 0xf000。

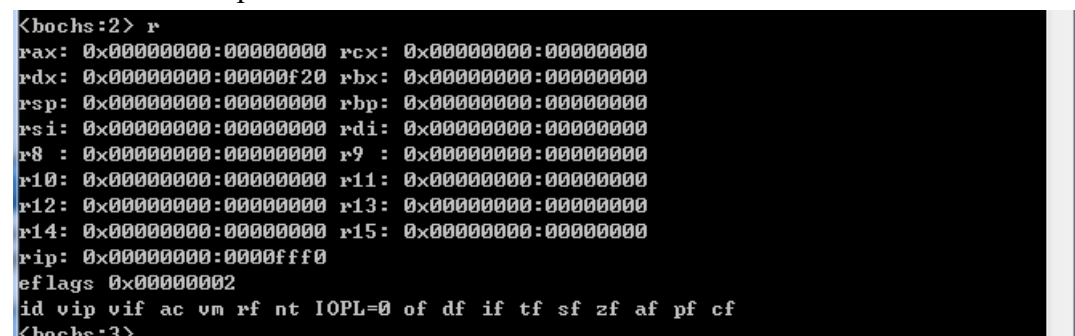


```

=====
Bochs x86 Emulator 2.3.7
Build from CVS snapshot, on June 3, 2008
=====
0000000000i|      | reading configuration from C:\test_2\test_2\bochs\bochsrc.b
xrc
0000000000e|      | C:\test_2\test_2\bochs\bochsrc.bxrc:5: ataX-master/slave CH
S set to 0/0/0 - autodetection enabled
0000000000i|      | installing win32 module as the Bochs GUI
0000000000i|      | using log file bochsout.txt
Next at t=0
<0> context not implemented because BX_HAVE_HASH_MAP=0
[0xffffffff] 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>

```

输入调试命令 `xp /1024b 0x0000`

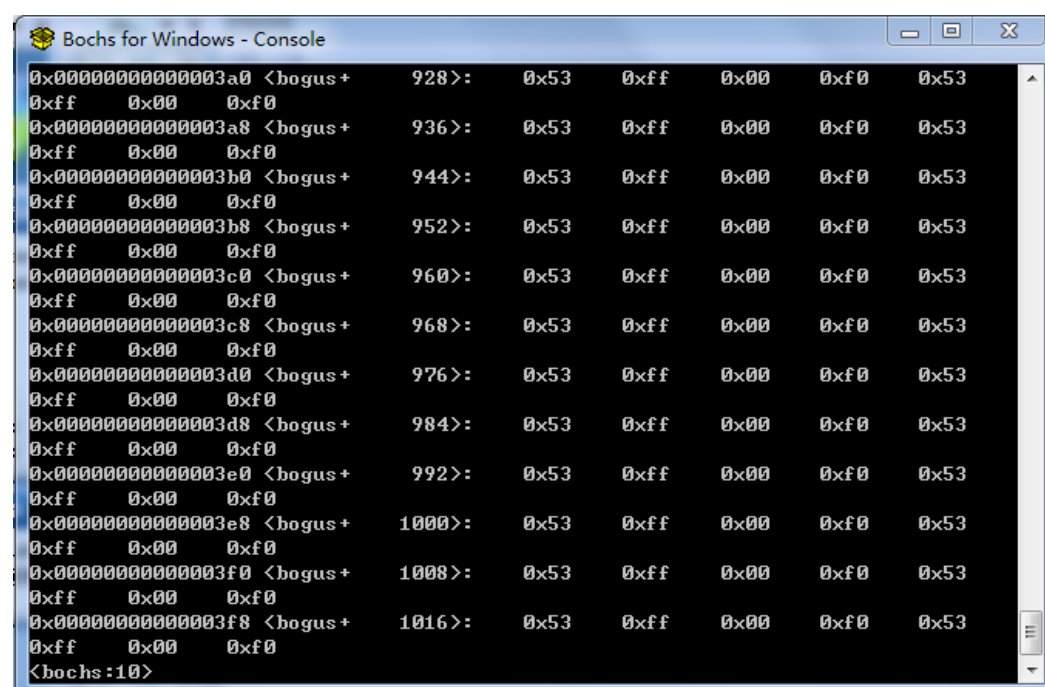


```

<bochs:2> xp /1024b 0x0000
rax: 0x00000000:00000000 rcx: 0x00000000:00000000
rdx: 0x00000000:00000020 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: 0x00000000:0000ffff
eflags 0x00000002
id vip vif ac vm rf nt IOPL=0 of df if tf sf zf af pf cf
<bochs:3>

```

输入调试命令 `xp /512b 0x7c00,`



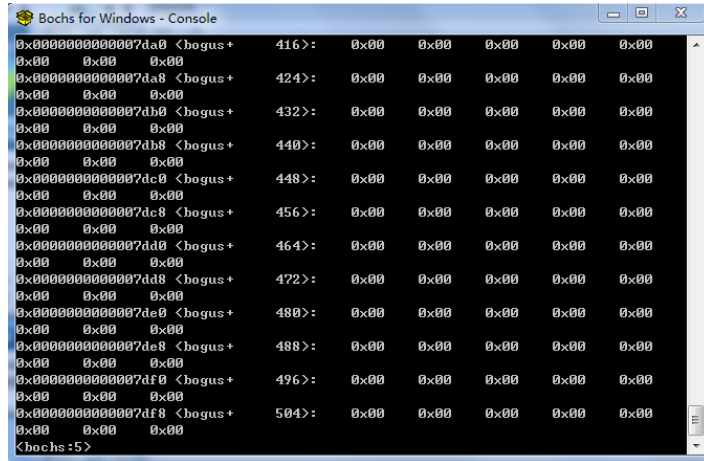
```

Bochs for Windows - Console
0x000000000000003a0 < bogus + 928>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003a8 < bogus + 936>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003b0 < bogus + 944>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003b8 < bogus + 952>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003c0 < bogus + 960>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003c8 < bogus + 968>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003d0 < bogus + 976>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003d8 < bogus + 984>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003e0 < bogus + 992>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003e8 < bogus + 1000>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003f0 < bogus + 1008>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x000000000000003f8 < bogus + 1016>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
<bochs:10>

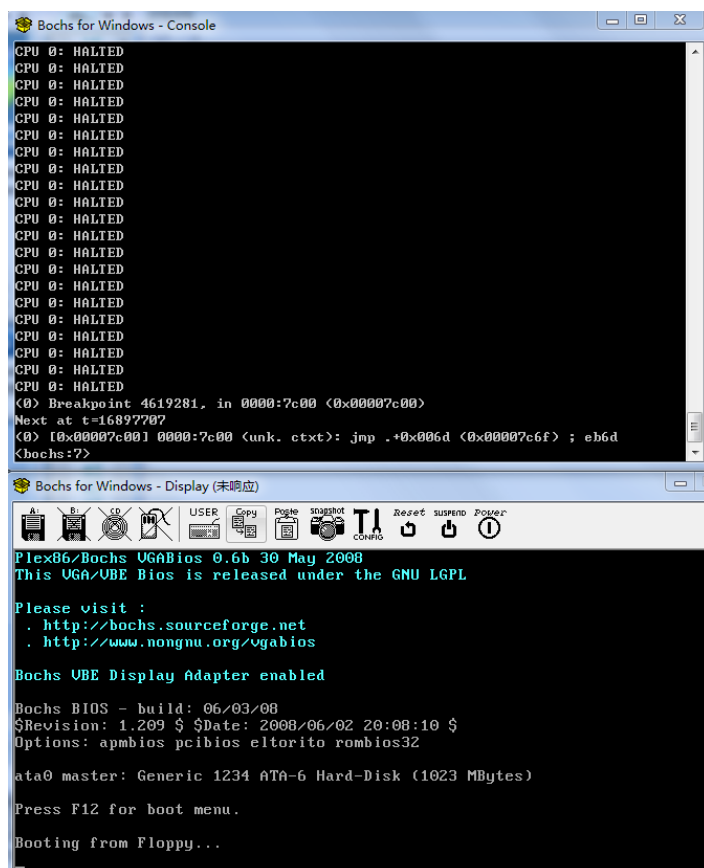
```

调试软盘引导扇区程序 BIOS 在执行完自检和初始化工作后，会将软盘引导扇区（512 字节）加载到物理地址 0x7c00-0x7dff 位置，并从 0x7c00 处的指令开始执行引导程序，所以接下来练习从 0x7c00 处调试软盘引导扇区程序：

1. 输入调试命令 `vb 0x0000:0x7c00`，这样就在逻辑地址 `0x0000:0x7c00`（相当于物理地址 `0x7c00`）处添加了一个断点。

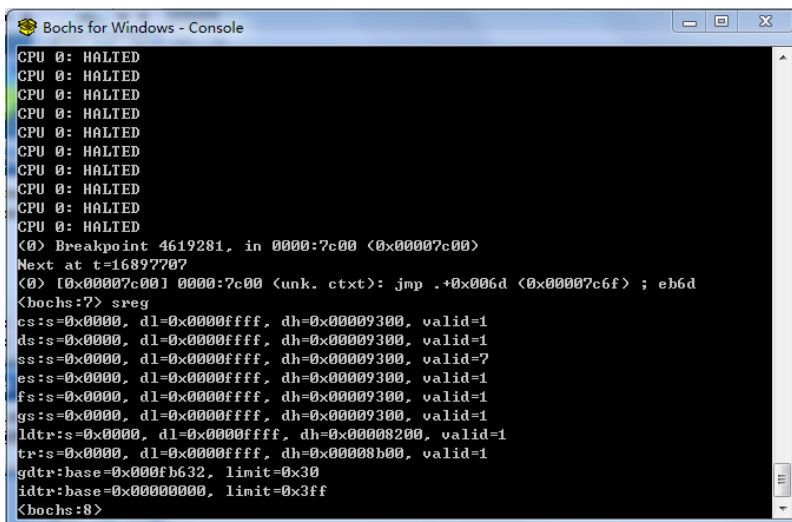


2. 输入调试命令 `c` 继续执行，在 `0x7c00` 处的断点中断。中断后会在 Console 窗口中输出下一个要执行的指令，即软盘引导扇区程序的第一条指令，如下 (0)
- ```
[0x00007c00] 0000:7c00 (unk. ctxt): jmp .+0x006d (0x00007c6f); eb6d
```



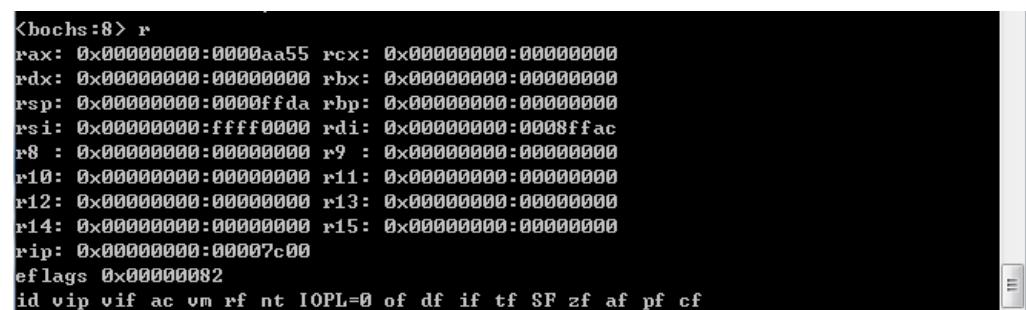
3. 为了方便后面的使用,读者可以先在纸上分别记录下此条指令的字节码(**eb6d**)和此条指令要跳 转执行的下一条指令的地址(括号中的 **0x00007c6f**)。

4. 输入调试命令 `sreg` 验证 CS 寄存器 (0x0000) 的值。



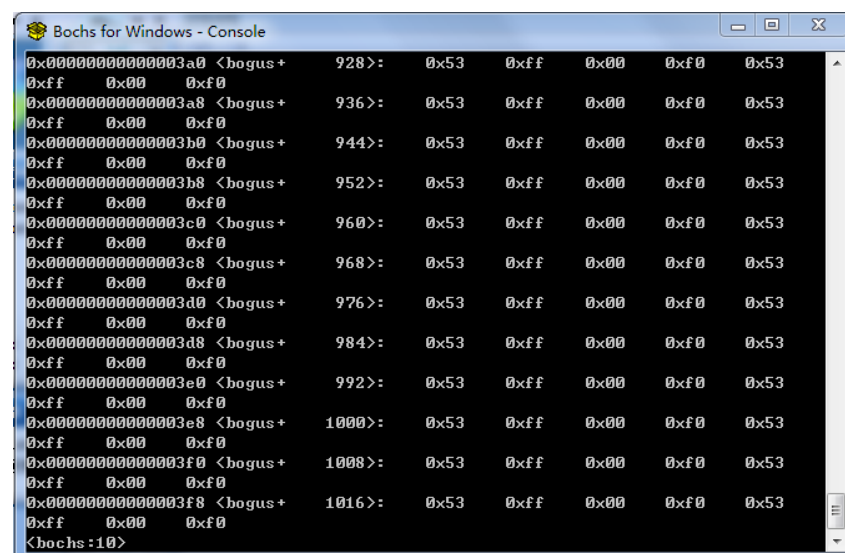
```
Bochs for Windows - Console
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
CPU 0: HALTED
(0) Breakpoint 4619281, in 0000:7c00 (0x00007c00)
Next at t=16897707
(0) [0x00007c00] 0000:7c00 (unk. ctxt): jmp .+0x006d (0x00007c6f) ; eb6d
<bochs:7> sreg
cs:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ds:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=1
ss:s=0x0000, dl=0x0000ffff, dh=0x00009300, valid=7
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
gdt:base=0x000fb632, limit=0x30
idt:base=0x00000000, limit=0x3ff
<bochs:8>
```

5. 输入调试命令 `r` 验证 IP 寄存器 (0x7c00) 的值。



```
<bochs:8> r
rax: 0x00000000:0000aa55 rcx: 0x00000000:00000000
rdx: 0x00000000:00000000 rbx: 0x00000000:00000000
rsp: 0x00000000:0000ffda rbp: 0x00000000:00000000
rsi: 0x00000000:ffffff00 rdi: 0x00000000:0008ffac
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: 0x00000000:00007c00
eflags 0x00000082
id vip vif ac vm rf nt IOPL=0 of df if tf SF zf af pf cf
```

6. 由于 BIOS 程序此时已经执行完毕, 输入调试命令 `xp/1024b 0x0000` 验证此时 BIOS 中断向量表已经被载入。



```
Bochs for Windows - Console
0x00000000000003a0 <bogus+ 928>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003a8 <bogus+ 936>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003b0 <bogus+ 944>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003b8 <bogus+ 952>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003c0 <bogus+ 960>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003c8 <bogus+ 968>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003d0 <bogus+ 976>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003d8 <bogus+ 984>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003e0 <bogus+ 992>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003e8 <bogus+ 1000>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003f0 <bogus+ 1008>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
0x00000000000003f8 <bogus+ 1016>: 0x53 0xff 0x00 0xf0 0x53
0xff 0x00 0xf0
<bochs:10>
```

7. 输入调试命令 `xp/512b 0x7c00` 显示软盘引导扇区程序的所有字节码。观察此块内存最开始的两个字节分别为 0xeb 和 0x6d, 这和引导程序第一条指令的字节码 (eb6d) 是相同的。此块内存最后的两个字节分别为 0x55 和 0xaa, 表示引导扇区是激活的, 可以用来引导操作系统, 这两个字节是 boot.asm 中最后一行语句 `dw 0xaa55` 定义的 (注意, Intel 80386 CPU 使用 little-endian 字节顺序, 参见附录 B)。

```

Bochs for Windows - Console
0x0000000000007da0 <hogus + 416>: 0x18 0x7c 0xf6 0xf3 0xfe
0xc4 0x88 0xe1
0x0000000000007da8 <hogus + 424>: 0x88 0xc6 0xd0 0xe8 0x88
0xc5 0x80 0xe6
0x0000000000007db0 <hogus + 432>: 0x01 0x8a 0x16 0x24 0x7c
0x5b 0xb4 0x02
0x0000000000007db8 <hogus + 440>: 0x8a 0x46 0xfe 0xcd 0x13
0xf 0x82 0xf5
0x0000000000007dc0 <hogus + 448>: 0xff 0x59 0x5d 0xc3 0x00
0x00 0x00 0x00
0x0000000000007dc8 <hogus + 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dd0 <hogus + 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007dd8 <hogus + 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007de0 <hogus + 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007de8 <hogus + 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007df0 <hogus + 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007df8 <hogus + 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x55 0xaa
<bochs:15>

```

8. 输入调试命令 `xp /512b 0x0600` 验证图 3-2 中第一个用户可用区域是空白的。

```

Bochs for Windows - Console
0x0000000000007a0 <hogus + 416>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007a8 <hogus + 424>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007b0 <hogus + 432>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007b8 <hogus + 440>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007c0 <hogus + 448>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007c8 <hogus + 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007d0 <hogus + 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007d8 <hogus + 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007e0 <hogus + 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007e8 <hogus + 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007f0 <hogus + 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007f8 <hogus + 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:16>

```

9. 输入调试命令 `xp /512b 0x7e00` 验证图 3-2 中第二个用户可用区域是空白的。

```

Bochs for Windows - Console
0x0000000000007fa0 <hogus + 416>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fa8 <hogus + 424>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fb0 <hogus + 432>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fb8 <hogus + 440>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fc0 <hogus + 448>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fc8 <hogus + 456>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fd0 <hogus + 464>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fd8 <hogus + 472>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fe0 <hogus + 480>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007fe8 <hogus + 488>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007ff0 <hogus + 496>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
0x0000000000007ff8 <hogus + 504>: 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00
<bochs:17>

```



10. 输入调试命令 `xp /512b 0xa0000` 验证图 3-2 中上位内存已经被系统占用。

Bochs for Windows - Console

|                                   |      |      |      |      |      |
|-----------------------------------|------|------|------|------|------|
| 0x000000000000a01a <bogus + 416>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a01b <bogus + 424>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a01c <bogus + 432>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a01d <bogus + 440>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a01e <bogus + 448>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a01f <bogus + 456>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a020 <bogus + 464>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a021 <bogus + 472>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a022 <bogus + 480>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a023 <bogus + 488>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a024 <bogus + 496>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| 0x000000000000a025 <bogus + 504>: | 0xff | 0xff | 0xff | 0xff | 0xff |
| <bochs:18>                        |      |      |      |      |      |

NASM 汇编器在将 `boot.asm` 生成 `boot.bin` 的同时，会生成一个 `boot.lst` 列表文件，帮助开发者调试 `boot.asm` 文件中的汇编代码。按照下面的步骤查看 `boot.lst` 文件：

```
boot.lst start.c
1  **
2
3  Copyright (c) 2008 北京英真时代科技有限公司。保留所有权利。
4
5  只有您接受 EOS 核心源代码协议 (参见 License.txt) 中的条款才能使用这些代码。
6  如果您不接受, 不能使用这些代码。
7
8  文件名: boot.asm
9
10 描述: 引导扇区。
11
12
13
14 *****/
15
16 ++++++
17
18 boot.asm
19
20 PC 机加电后, CPU 进入实模式, 分段管理内存, 最多访问 1M 地址空间 (没
21 有打开 A20 的情况下)。CPU 首先执行 BIOS 程序, 在 BIOS 完成设备检测等工
22 作后, 如果 BIOS 被设置为从软盘启动, 则 BIOS 会将软盘的引导扇区 (512 字节)
23 加载到物理地址 0x7C00 - 0x7DFF 处, 然后将 CPU 的 CS 寄存器设置为 0x0000,
24 将 IP 寄存器设置为 0x7C00。接下来, CPU 就开始执行引导扇区中的程序。
```

在 boot.lst 中查找到软盘引导扇区程序第一条指令所在的行（第 73 行）

```
73 73 00000000 EB6D jmp short Start
```

在 `boot.lst` 文件中查找到加载完毕 `loader.bin` 文件后要跳转到 `loader` 程序中执行的指令（第 278 行）

```
278: 278 00000181 EA00100000      jmp 0:LOADER_ORG
```

输入调试命令 `vb 0x0000:0x7d81` 添加一个断点。输入调试命令 `c` 继续执行，到断点处中断。在 **Console** 窗口中显示

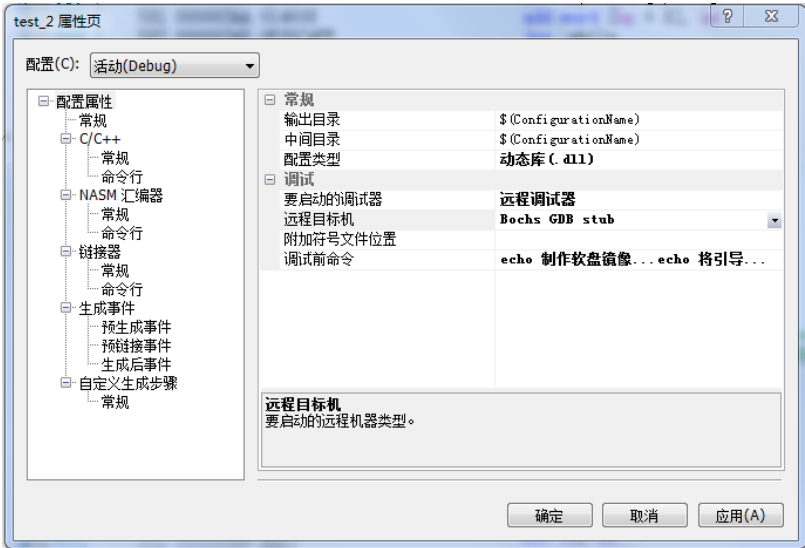


使用查看虚拟内存的调试命令 `x /1wx ds:0x80001117` 查看内存中保存的 32 位函数入口地址

```
<bochs:7> x /1wx ds:0x80001117
[bochs]:
0x0000000080001117 <bogus+      0>:      0x80017de0
```

继续调试 EOS 操作系统的内核，验证从加载程序进入内核入口点函数的过程。步骤如下：

1. 在“项目管理器”窗口中，右键点击项目节点，在弹出的快捷菜单中选择“属性”。



2. 在弹出的“属性页”对话框右侧的属性列表中找到“远程目标机”属性，将此属性值修改为“Bochs GDB stub”。
3. 点击“确定”按钮关闭“属性页”对话框。

```
372 | fprintf(StdHandle, "\nEngintime EOS [Version Number 1.2]\n\n");
```

4. 在“项目管理器”窗口中打开 ke 文件夹中的 start.c 文件，此文件中只定义了一个函数，就是操作系统内核的入口点函数 KiSystemStartup。

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

一个函数，就是操作系统内核的入口点函数 KiSystemStartup。

5. 在 KiSystemStartup 函数中的代码行（第 52 行）KiInitializePic(); 添加一个断点。

```
52 | KiInitializePic();
```

6. 按 F5 启动调试，会在刚刚添加的断点处中断。

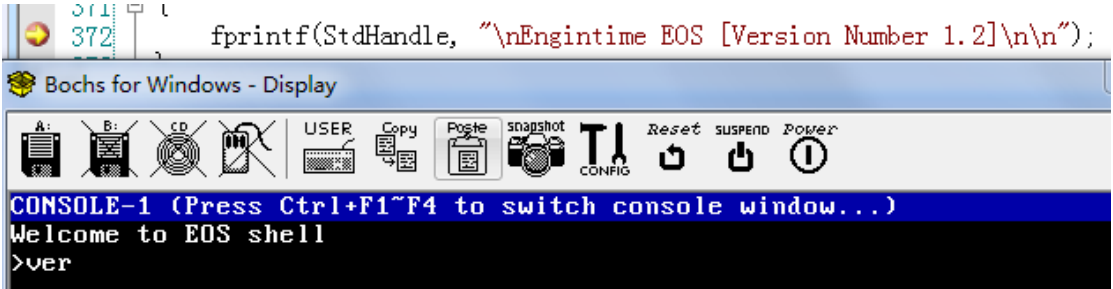
```
372 | fprintf(StdHandle, "\nEngintime EOS [Version Number 1.2]\n\n");
373 | }
374 |
```



7. 在 start.c 源代码文件中的 KiSystemStartup 函数名上点击鼠标右键，在弹出的快捷菜单中选择“添加监视”，KiSystemStartup 函数就被添加到了“监视”窗口中。在“监视”窗口中可以看到 此函数地址为 {void (PVOID)} 0x800\*\*\*\*\* 与之

前记录的在内存 ds:x80001117 处保存的函数入口地址相同，说明的确是由 Loader 程序进入了操作系统内核。

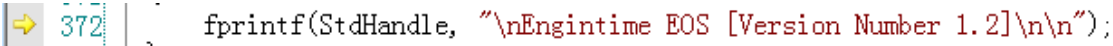
8. 按 F5 继续执行 EOS 操作系统内核，在 Display 窗口中显示 EOS 操作系统已经启动，并且控制台程序已经开始运行了。



9. 选择“调试”菜单中的“删除所有断点”菜单项，删除之前添加的所有断点。

10. 选择“调试”菜单中的“停止调试”菜单项，停止调试。

### EOS 启动后的状态和行为



删除所有断点后，在 ke/sysproc.c 文件的第 143 行添加一个断点。读者会注意到这是在一个死循环中添加了一个断点，没错，当没有其它线程运行时，空闲线程总是会不停的执行这个死循环，直到有中断发生，或者有更高优先级的线程抢占了处理器。



观察其它线程的状态。

对象类型

数据源: SINGLE\_LIST\_ENTRY ObpTypeListHead

源文件: ob\obtype.c

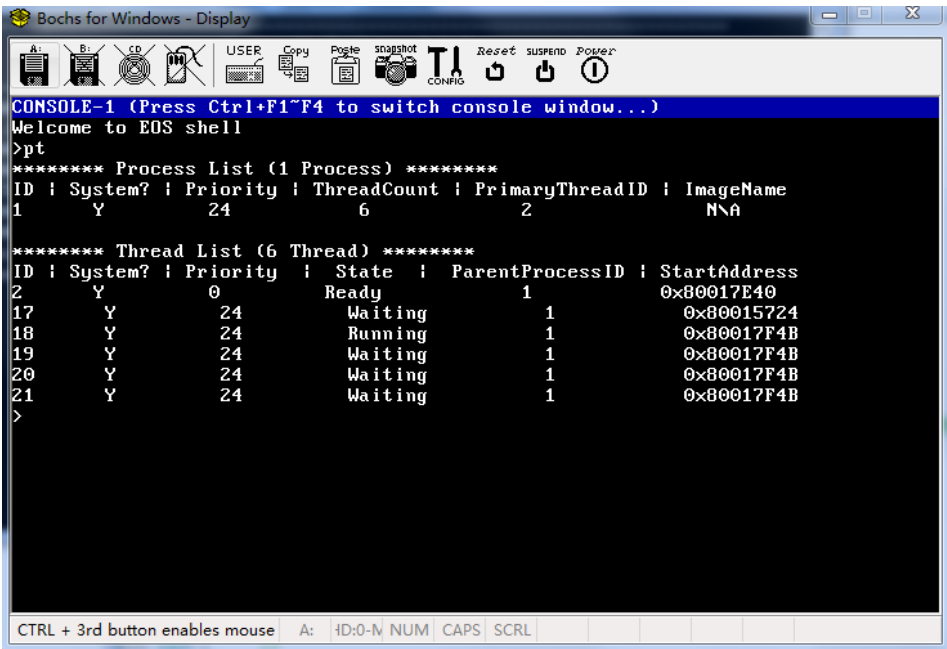
类型链表

| TypeListEntry |                                  |
|---------------|----------------------------------|
| Name          | "FILE"                           |
| ObjectCount   | 1                                |
| Create        | NULL                             |
| Delete        | 0x80015db1<br>IopCloseFileObject |
| Wait          | NULL                             |
| Read          | 0x80015e3c<br>IopReadFileObject  |
| Write         | 0x80015f14<br>IopWriteFileObject |

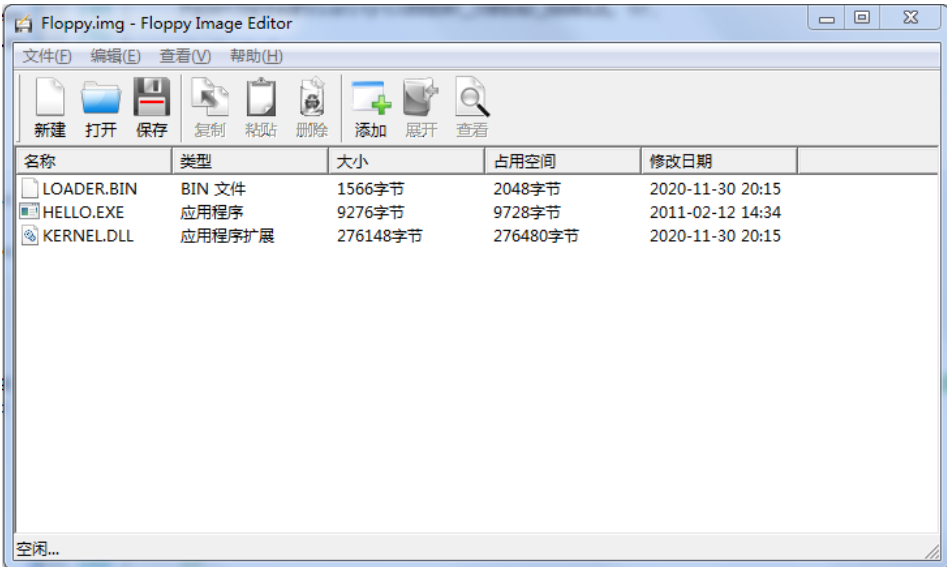
↓

| TypeListEntry |                                     |
|---------------|-------------------------------------|
| Name          | "CONSOLE"                           |
| ObjectCount   | 4                                   |
| Create        | NULL                                |
| Delete        | NULL                                |
| Wait          | NULL                                |
| Read          | 0x80015253<br>IopReadConsoleInput   |
| Write         | 0x80015190<br>IopWriteConsoleOutput |

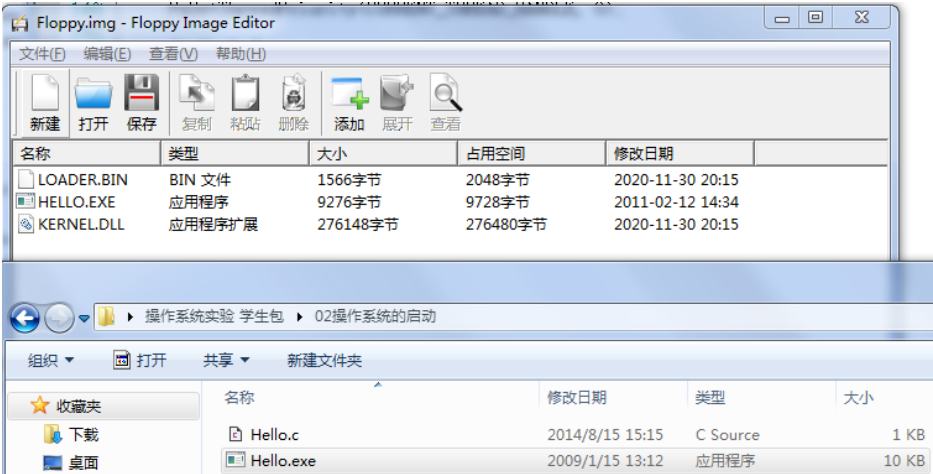
使用 pt 命令查看进程和线程的信息



使用 FloppyImageEditor 工具打开此软盘镜像文件。



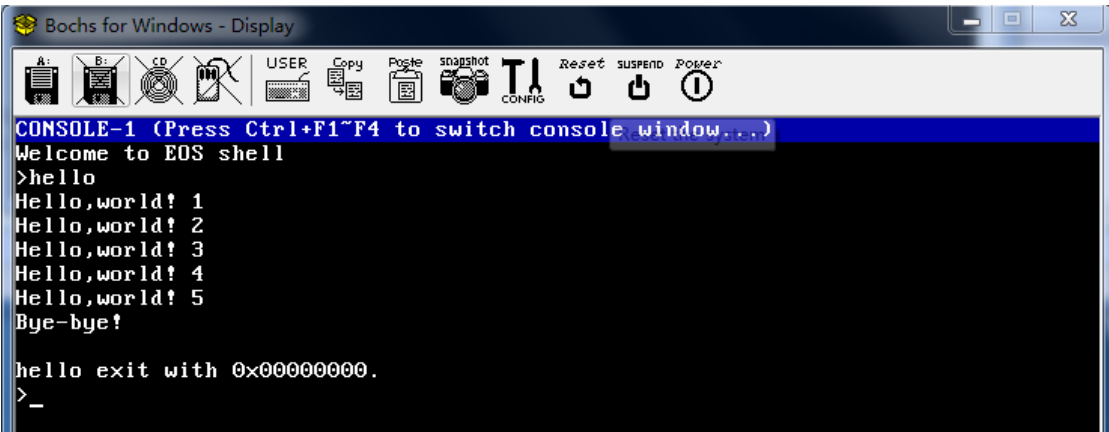
将本实验文件夹中的 Hello.exe 文件拖动到 FloppyImageEditor 工具窗口的文件列表中释放



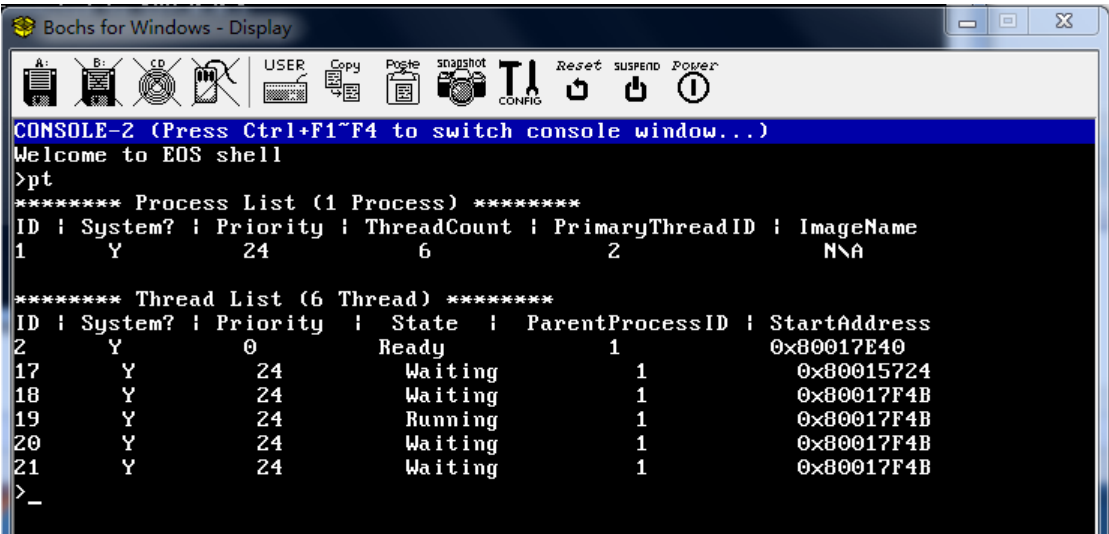
按 F5 启动调试。  
待 EOS 启动完毕，在 EOS 控制台中输入命令“hello”后按回车。此时在软盘中



的 EOS 应用程序 Hello.exe 就会开始运行。



迅速按 Ctrl+F2 切换到控制台  
并输入命令“pt”后按回车。输出的进程和线程信息如图 10-7 所示。



#### 4. 实验的思考与问题分析

(1) 为什么 EOS 操作系统从软盘启动时要使用 boot.bin 和 loader.bin 两个程序？使用一个可以吗？它们各自的主要功能是什么？如果将 loader.bin 的功能移动到 boot.bin 文件中，则 boot.bin 文件的大小是否仍然能保持小于 512 字节？

答：

在 IDE 环境启动执行 EOS 操作系统的时候，会把 boot. bin、loader. bin 和 kernel. dll 三个二进制文件写到软盘镜像文件中，然后让虚拟机来执行软盘里的 EOS 操作系统。仅使用其中的一个是不能运行的。

Boot.bin 程序的功能是:在 Boot.bin 程序执行的过程中, CPU 始终处于实模式状态。Boot. bin 程序利用 BIOS 提供的 int 0x13 中断服务程序读取软盘 FAT12 文件系统的根目录, 在根目录中搜寻 loader.bin 文件。

Loader. bin 程序的功能是: Loader. bin 程序的任务和 Boot. bin 程序很相似, 同样是将其它的程序加载到物理内存中, 但这次加载的是 EOS 内核。除此之外, Loader. bin 程序还负责检测内存大小, 为内核准备保护模式执行环境等工作。如果把 loader. bin 功能移动到 boot. bin 程序中, 就会导致程序规模的扩大, 可能使其大小大于 512 字节。

(2) 软盘引导扇区加载完毕后内存中有两个用户可用的区域, 为什么软盘引导扇区程序选择 loader.bin 加载到第一个可用区域的 0x1000 处呢? 这样做有什么好处? 这样做会对 loader.bin 文件的大小有哪些限制。

答:

用户只有两个可用区域, 加载的位置只能在这两个区域内选择。第一个用户可用区域是低地址区, 并且空间是比较小的, 适合容纳较小的文件, 所以我们选择把占用空间小的 loader.bin 加载到第一用户区。

好处: 由低地址开始, 方便检索查找。小文件占用较小的空间, 能够节约资源。

限制: loader. bin 文件必须小于 1c00k 才能放到第一用户区。

(3) 练习使用 Bochs 单步调试 BIOS 程序、软盘引导扇区程序和 loader 程序, 加深对操作系统启动过程的理解。

答:

(4) EOS 空闲线程在其死循环中不停的执行 i++, 从效率和节能的角度来说, 这种方式都是不可取的。请读者尝试使用内联汇编将 i++ 替换为停机指令 “HLT”。

答:

## 4. 总结和感想体会

通过这次实验, 我锻炼了自己的动手能力, 进一步熟悉了软件的应用, 对课本上的知识加深了理解。对系统的进程, 线程以及存储等知识有了一定的了解, 直观的感受让我印象深刻。通过对命令的键入和设计, 我对于常用操作系统编程命令有了一定的熟悉。

## 参考文献

- [1] 北京英真时代科技有限公司 [DB/CD]. <http://www.engintime.com>.
- [2] Bochs. [DB/CD]. <http://bochs.sourceforge.net>
- [3] NASM. [DB/CD]. <http://www.nasm.us>