

**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

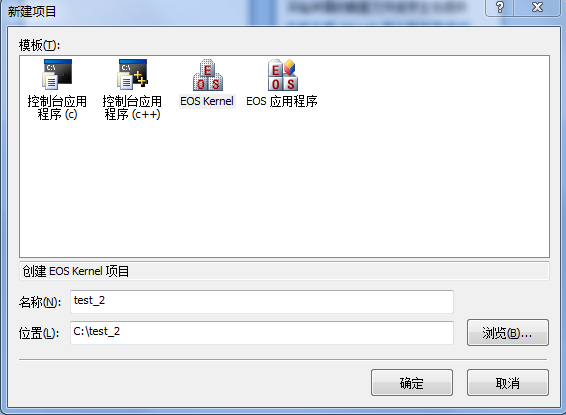
**完成日期**  11.07

**合肥工业大学 计算机与信息学院**

* 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。
  1. **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虛拟机软件的特点，Bochs的调试命令。

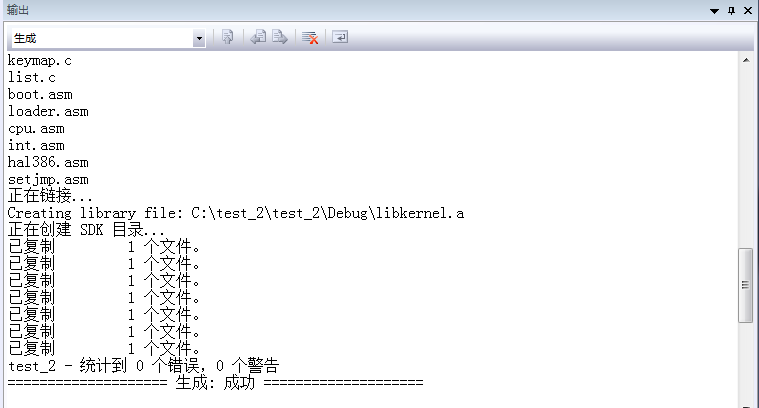
* 1. **实验内容**

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。

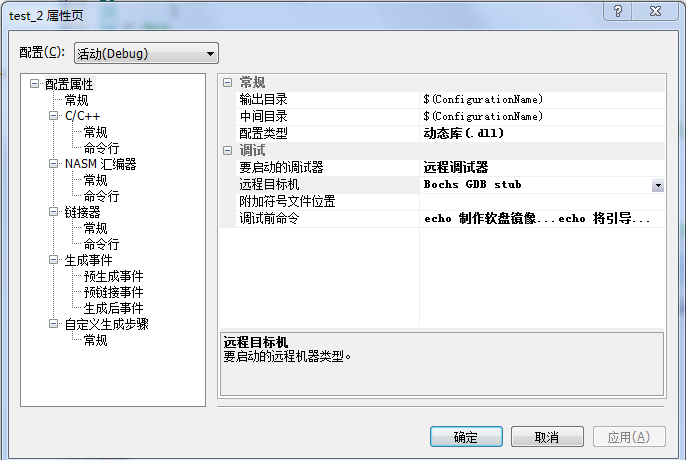
生成完成后，使用 Windows 资源管理器打开项目文件夹中的 Debug 文件夹。找到由 boot.asm 文 件生成的软盘引导扇区程序 boot.bin 文件，该文件的大小一定为 512 字节（与软盘引导扇区的 大小一致）。找到由 loader.asm 生成的加载程序 loader.bin 文件，记录下此文件的大小 1566 字 节，在下面的实验过程中会用到。找到由其它源文件生成的 EOS 操作系统内核文件 kernel.dll。

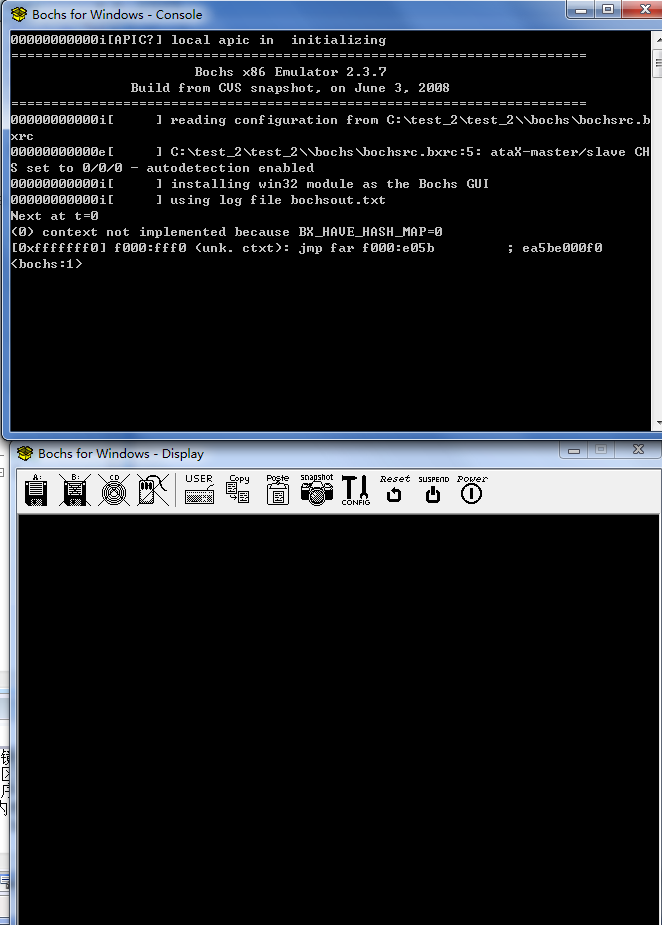
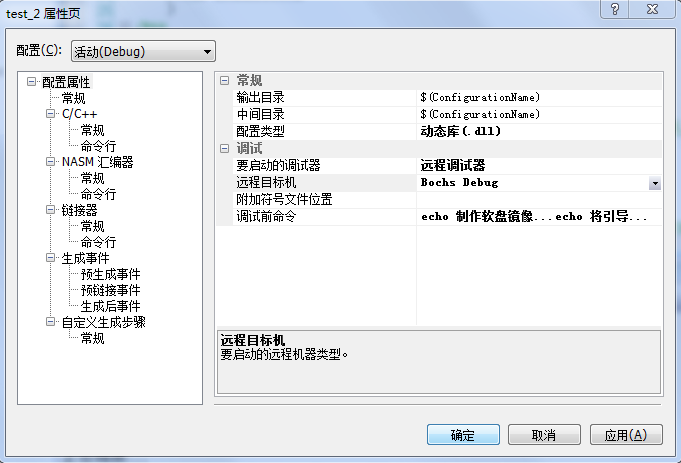
情况如下：

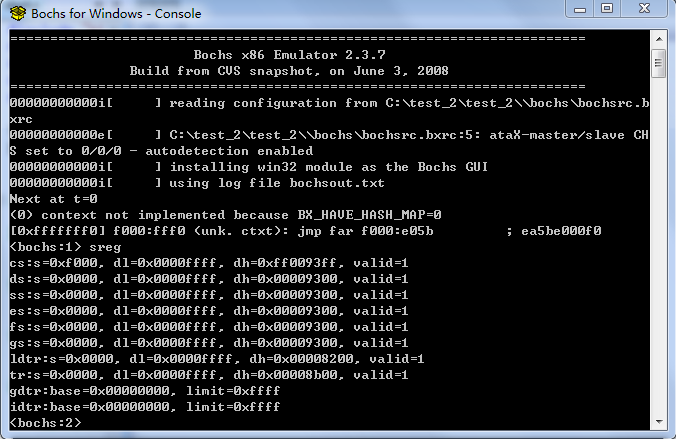
并且找到软盘引导扇区程序如下

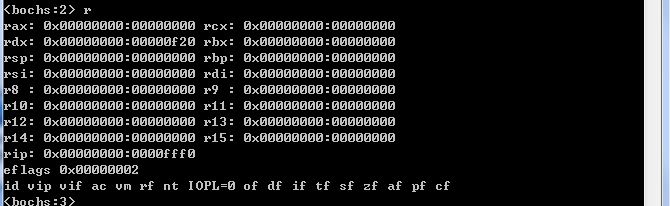
Loader.asm生成的加载程序

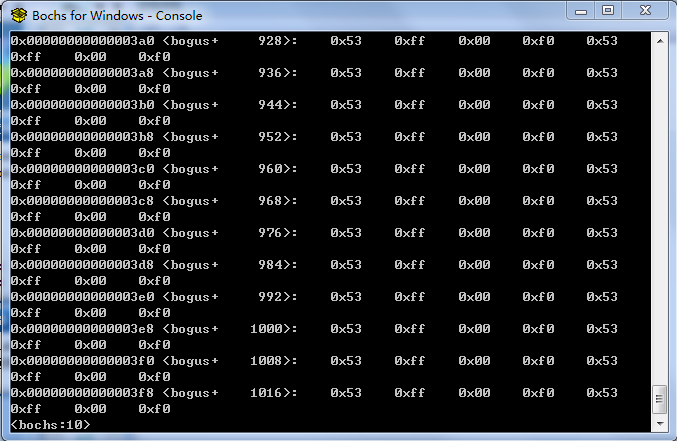
EOS操作系统内核文件

下面使用Bochs Debug做目标远程机

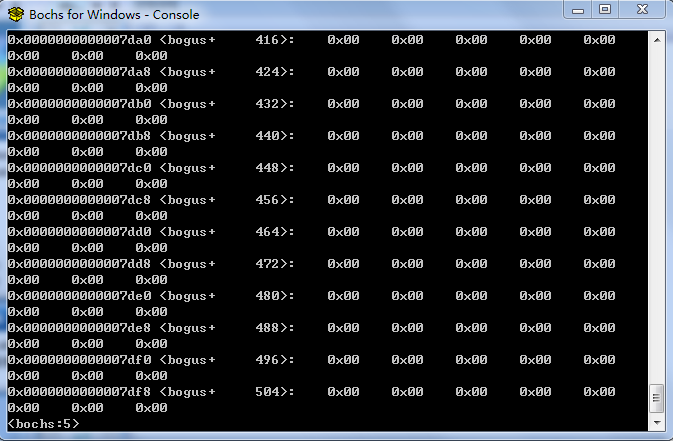
按F5调试，此时会弹出两个Bochs窗口。标题为“Bochs for windows-Display”的窗口相当于计算机的显示器，用于显示操作系统的输出。标题为“Bochs for windows-Console”的窗口是Bochs的控制台，用来输入调试命令，输出各种调试信息。

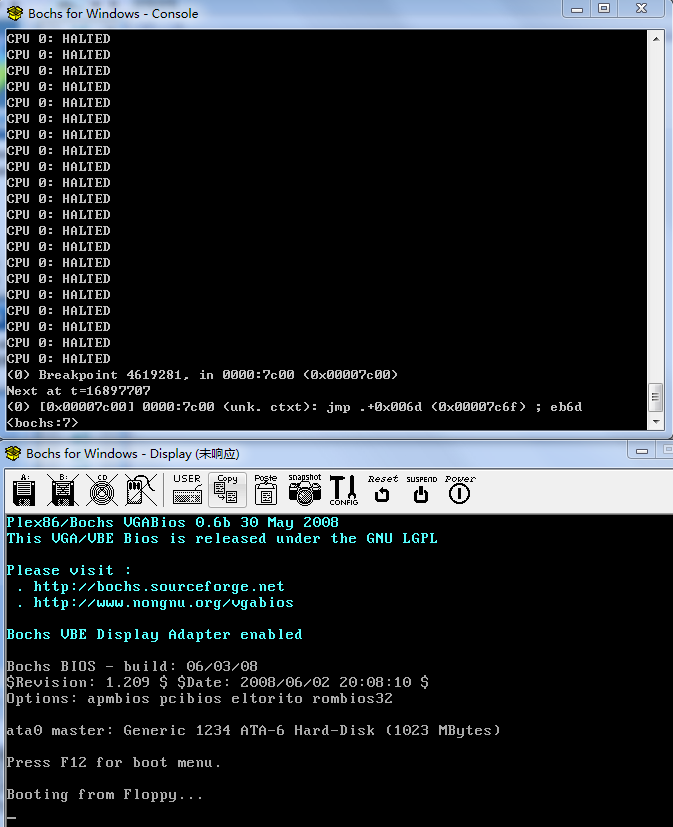
在 Console 窗口中输入调试命令 sreg 后按回车，显示当前 CPU 中各个段寄存器的值，如图 10-2。 其中 CS 寄存器信息行中的“s=0xf000”表示 CS 寄存器的值为 0xf000。

输入调试命令 xp /1024b 0x0000

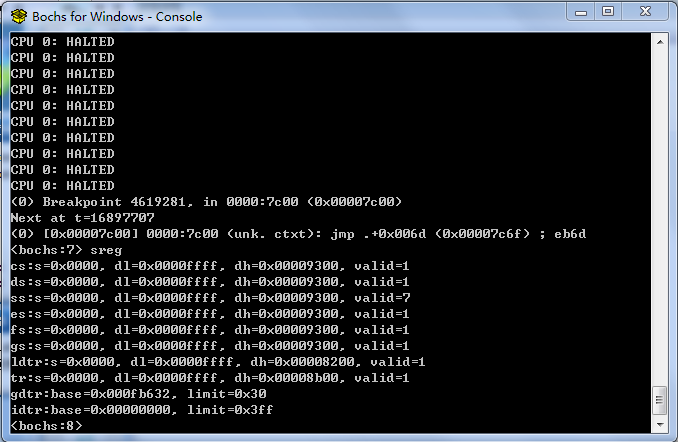
输入调试命令 xp /512b 0x7c00，

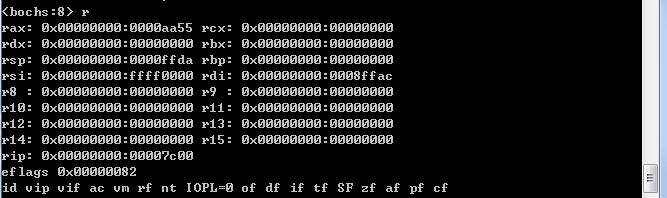
调试软盘引导扇区程序 BIOS 在执行完自检和初始化工作后，会将软盘引导扇区（512 字节）加载到物理地址 0x7c00-0x7dff 位置，并从 0x7c00 处的指令开始执行引导程序，所以接下来练习从 0x7c00 处调试软盘引导扇区程序：

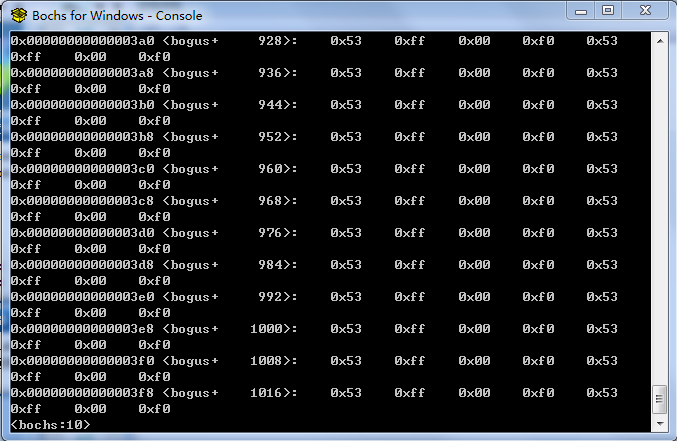
1. 输入调试命令 vb 0x0000:0x7c00，这样就在逻辑地址 0x0000:0x7c00（相当于物理地址 0x7c00） 处添加了一个断点。

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

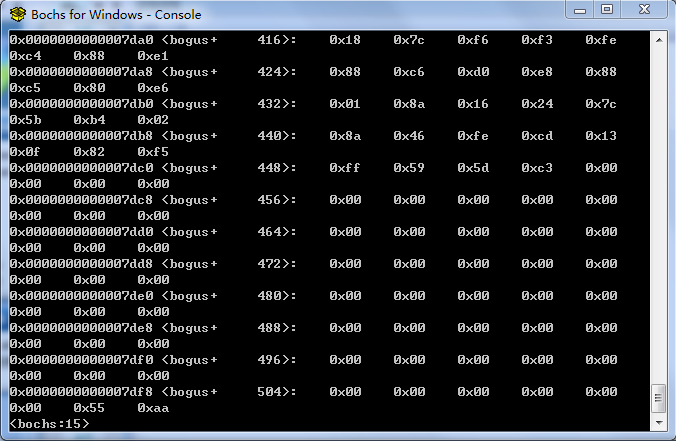
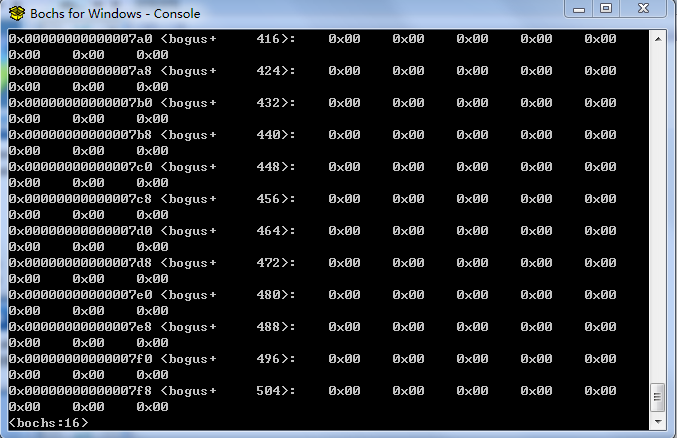
3. 为了方便后面的使用，读者可以先在纸上分别记录下此条指令的字节码（eb6d）和此条指令要跳 转执行的下一条指令的地址（括号中的 0x00007c6f）。

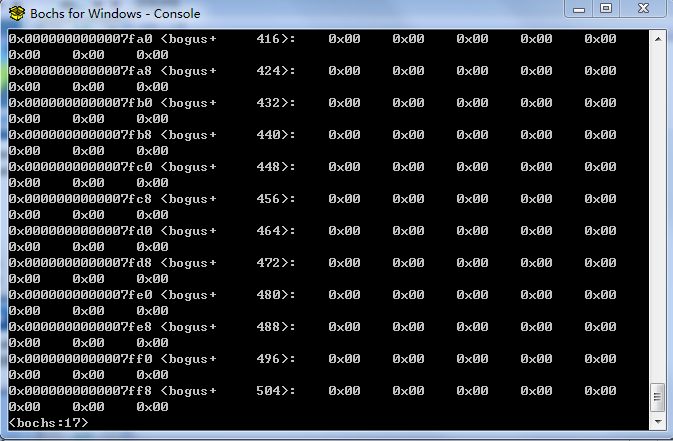
4.输入调试命令 sreg 验证 CS 寄存器（0x0000）的值。

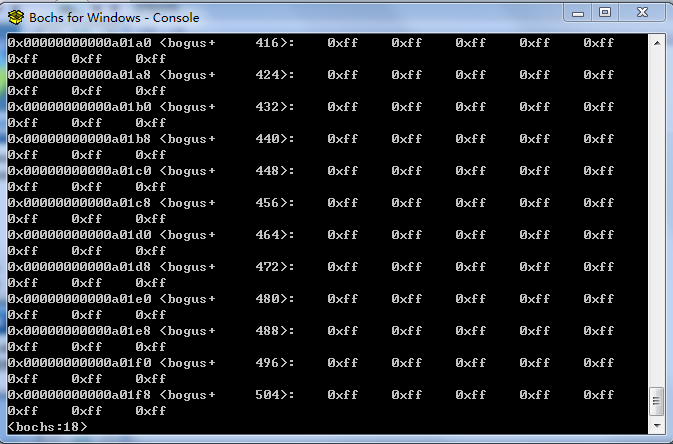
5. 输入调试命令 r 验证 IP 寄存器（0x7c00）的值。

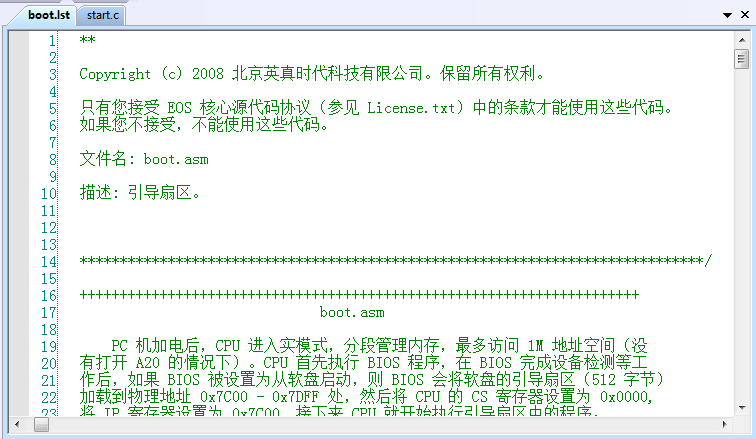
6. 由于 BIOS 程序此时已经执行完毕，输入调试命令 xp /1024b 0x0000 验证此时 BIOS 中断向量表已经被载入。

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）。

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。

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。

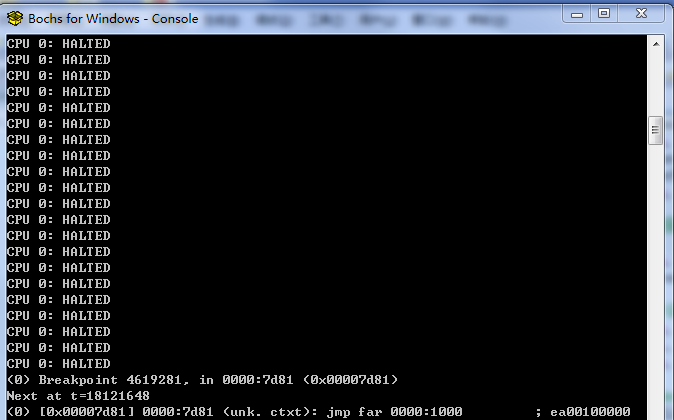
10. 输入调试命令 xp /512b 0xa0000 验证图 3-2 中上位内存已经被系统占用。

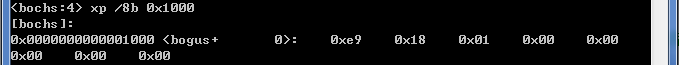
NASM 汇编器在将 boot.asm 生成为 boot.bin 的同时，会生成一个 boot.lst 列表文件，帮助开发者调 试 boot.asm 文件中的汇编代码。按照下面的步骤查看 boot.lst 文件：

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）

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）

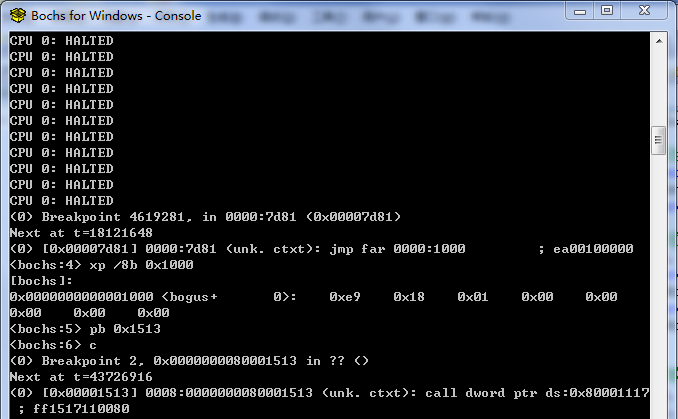
输入调试命令vb 0x0000:0x7d81 添加一个断点。输入调试命令 c 继续执行，到断点处中断。在 Console 窗口中显示

按照打开boot.lst文件的方法打开 loader.lst 文件，并在此文件中查找到 loader 程序的第一条 指令（第 33 行）

输入调试命令xp /8b 0x1000查看内存 0x1000 处的数据，验证此块内存的前三个字节和 loader.lst 文件中的第一条指令的字节码是相同的。说明 loader 程序已经加载到从地址 0x1000 开始的内存中了。根据之前记录的 loader.bin 文件的大小，自己设计一个查看内存的调试命令，查看内存中 loader 程序结束位置的字节码，并与 loader.lst 文件中最后指令的字节码比较。

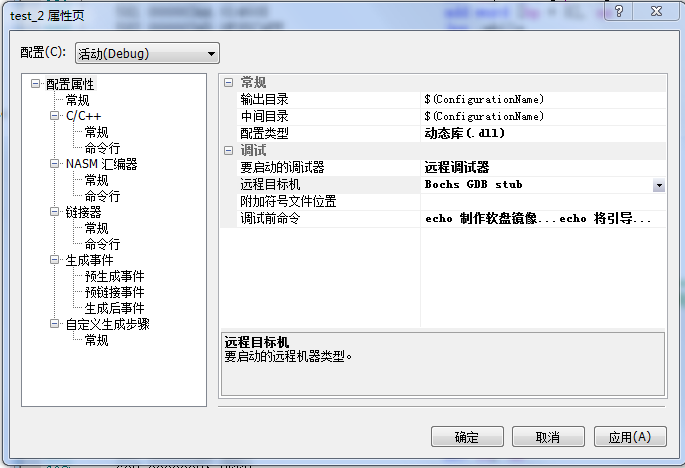
在loader.lst文件中查找到准备进入 EOS 操作系统内核执行的指令（第 755 行）

下面引用的代码是 loader.lst 文件中第一个节的最后一条指令（第 593 行）

****使用添加物理地址断点的调试命令 pb 0x1513 添加一个断点。 4. 输入调试命令 c 继续执行，在刚刚添加的断点处中断。在 Console 窗口中显示要执行的下一条指令（注意，此时指令中访问的地址均为逻辑地址）：

****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

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：

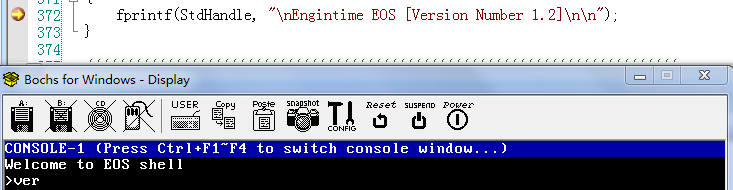
1. 在“项目管理器”窗口中，右键点击项目节点，在弹出的快捷菜单中选择“属性”。

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”。

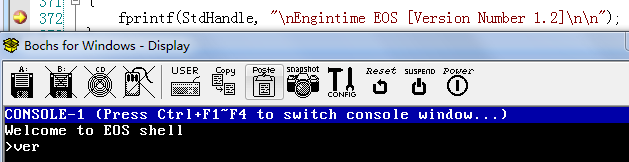
3. 点击“确定”按钮关闭“属性页”对话框。

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。

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

6. 按 F5 启动调试，会在刚刚添加的断点处中断。

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。

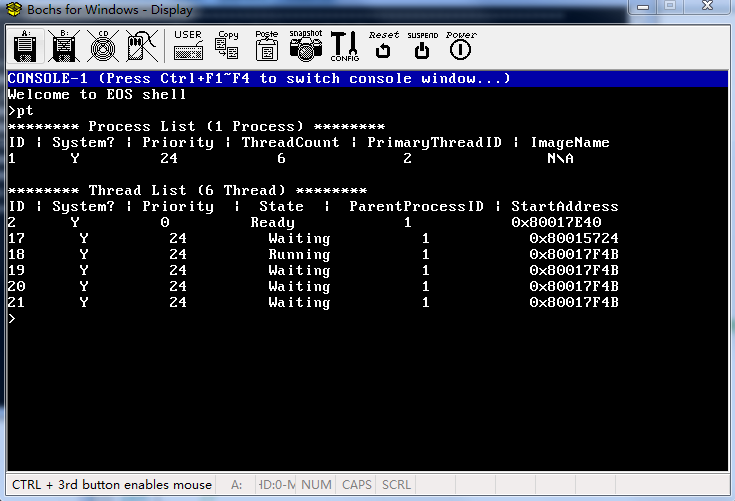
9. 选择“调试”菜单中的“删除所有断点”菜单项，删除之前添加的所有断点。

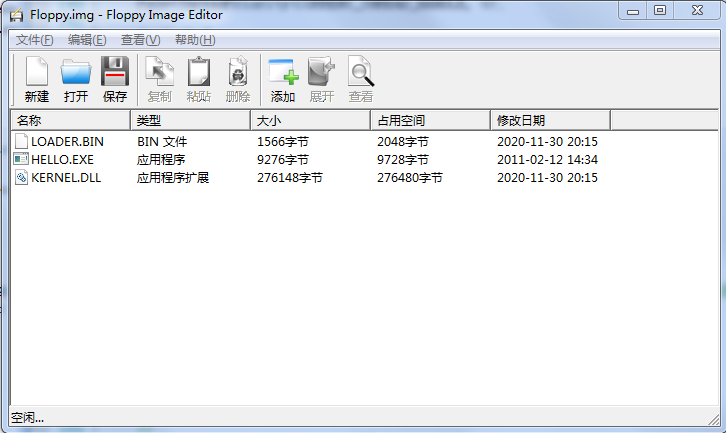
10. 选择“调试”菜单中的“停止调试”菜单项，停止调试。

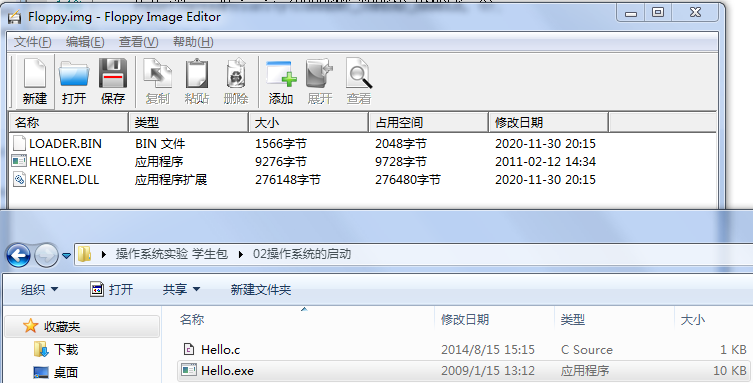
EOS 启动后的状态和行为

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。

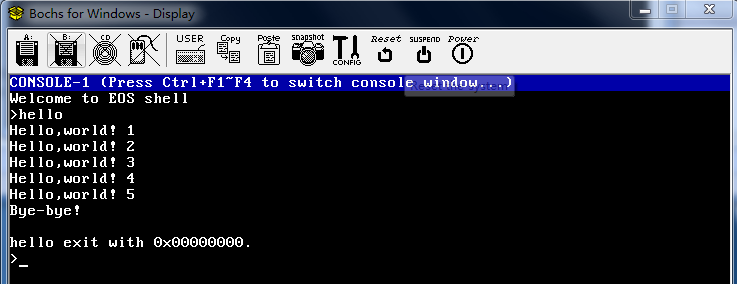
观察其它线程的状态。

使用 pt 命令查看进程和线程的信息

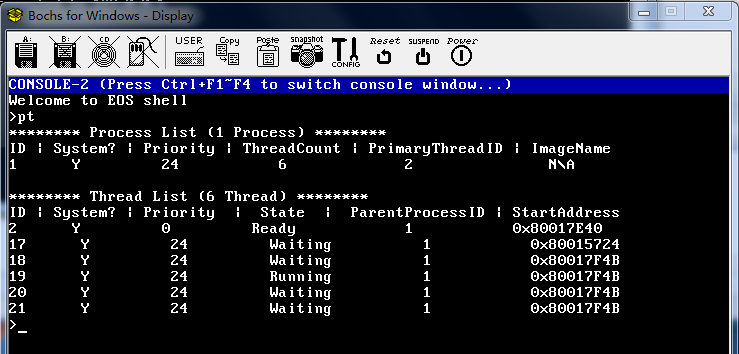
使用FloppyImageEditor工具打开此软盘镜像文件。

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. d11三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BI0S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loder.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：

* 1. **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