

**操作系统实验报告**

**实验题目**  操作系统的启动

**学生姓名**  付炎平

**学 号**  2019217819

**专业班级**  物联网工程19-2班

**指导教师**  田卫东

**完成日期**  2021.11.9

**合肥工业大学 计算机与信息学院**

**实验 2 操作系统的启动**

# 实验目的和任务要求

（1）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。

（2）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。

# 实验原理

（1）EOS 操作系统如何启动。

（2）汇编语言相关知识。

（3）Bochs 虚拟机软件的特点和 Bochs 的调试命令。

# 实验内容

## 准备实验

* + 1. 启动 OS Lab。
    2. 新建一个 EOS Kernel 项目。
    3. 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。
    4. 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。

## 调试 EOS 操作系统的启动过程

* + 1. **使用 Bochs Debug 做为远程目标机**

按照下面的步骤将调试时使用的远程目标机修改为 Bochs Debug：

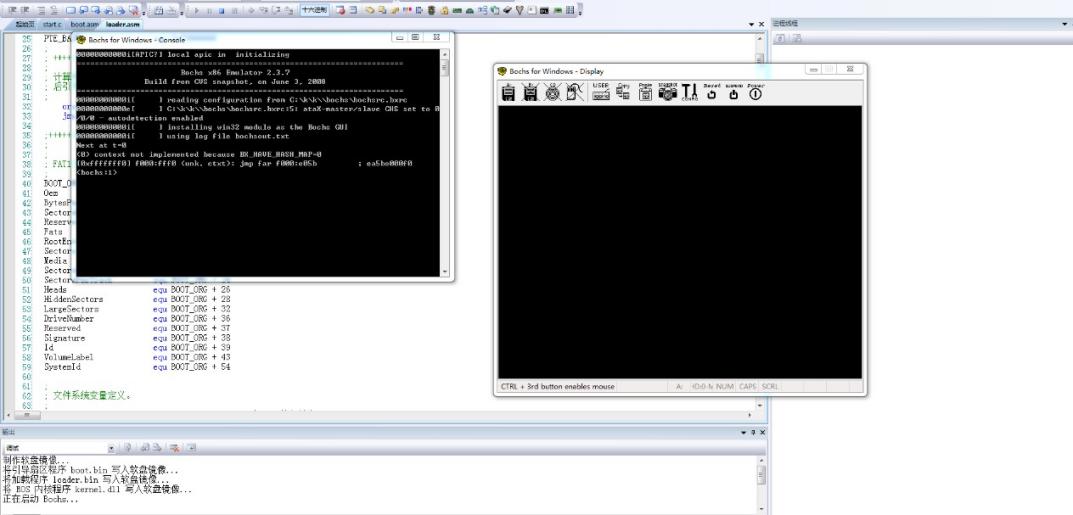
1. 在“项目管理器”窗口中，右键点击项目节点，在弹出的快捷菜单中选择“属性”。
2. 在弹出的“属性页”对话框右侧的属性列表中找到“远程目标机”属性， 将此属性值修改为“Bochs Debug”（此时按 F1 可以获得关于此属性的帮助）。
3. 点击“确定”按钮关闭“属性页”对话框。接下来就可以使用 Bochs 虚拟机提供的调试功能单步调试 BIOS 程序和 EOS 操作系统的软盘引导扇区程序了。

## 调试 BIOS 程序

按 F5 启动调试，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。

启动调试后，Bochs 会在 CPU 要执行的第一条指令（即 BIOS 的第一条指令）处中断。 此时，Display 窗口还没有显示任何内容，Console 窗口会显示将要执行的 BIOS 第一条指令的相关信息，并等待用户输入调试命令。



根据 Console 窗口显示的内容，读者可以获得关于 BIOS 第一条指令的如下信息：

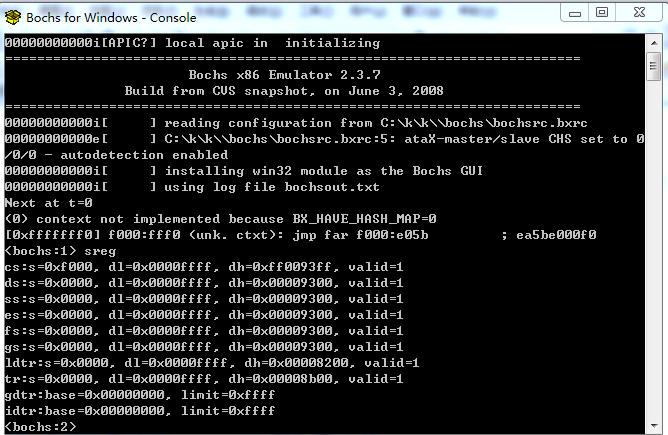
行首的[0xfffffff0]表示此条指令所在的物理地址。

f000:fff0 表示此条指令所在的逻辑地址（段地址:偏移地址），如果将其转换为物理地址，则得到的物理地址与行首显示的物理地址是一致的。

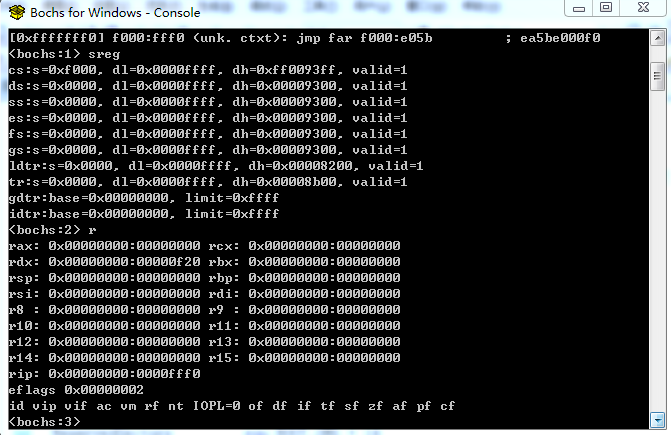
jmp far f000:e05b 是此条指令的反汇编代码。

行尾的 ea5be000f0 是此条指令的十六进制字节码，可以看出此条指令长度为 5 个字节。

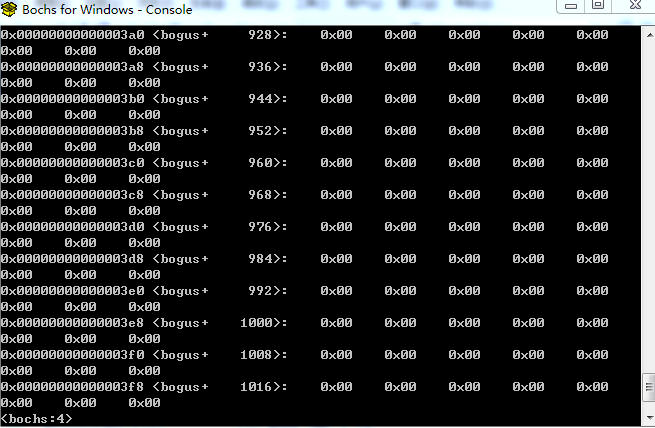
接下来可以按照下面的步骤，查看 CPU 在没有执行任何指令之前主要寄存器和内存中的数据：

1. 在 Console 窗口中输入调试命令 g sreg 后按回车，显示当前 CPU 中各个段寄存器的值.,其中 CS 寄存器信息行中的“s=0xf000”表示 CS 寄存器的值为 0xf000。
2. 输入调试命令 r r 后按回车，显示当前 CPU 中各个通用寄存器的值， 如图 10-3。其中“rip:

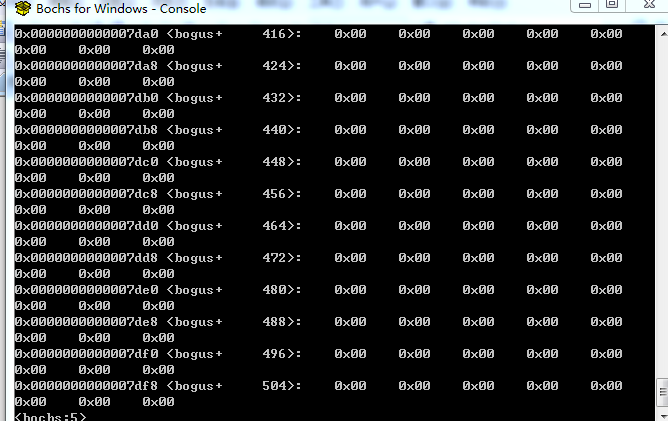
0x00000000:0000fff0”表示 IP 寄存器的值为 0xfff0。



1. 输入调试命令 xp /1024b 0x0000，查看开始的 1024 个字节的物理内存。在 Console 中输出的这 1K 物理内存的值都为 0，说明 BIOS 中断向量表还没有被加载到此处。



1. 输入调试命令 xp /512b 0x7c00，查看软盘引导扇区应该被加载到的内存位置。输出的内存值都为 0，说明软盘引导扇区还没有被加载到此处。

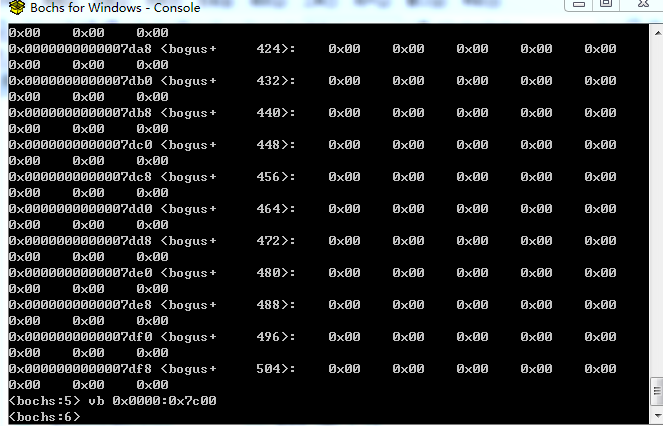


通过以上的调试步骤可以验证 BIOS 第一条指令所在逻辑地址中的段地址和 CS 寄存器值是一致的，偏移地址和 IP 寄存器的值是一致的。由于此时还没有执行任何指令，内存也还没有被使用，所以其中的值都为 0。

## 调试软盘引导扇区程序

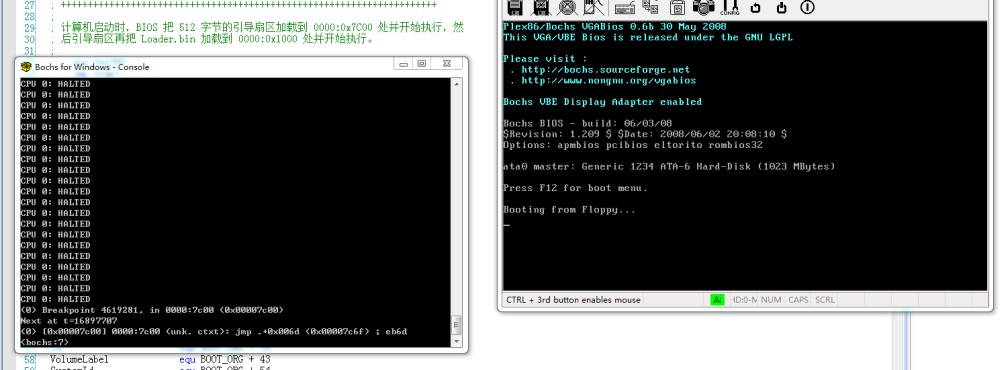
BIOS 在执行完自检和初始化工作后，会将软盘引导扇区（512 字节）加载到物理地址 0x7c00-0x7dff 位置，并从 0x7c00 处的指令开始执行引导程序， 所以接下来练习从 0x7c00 处调试软盘引导扇区程序：

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。

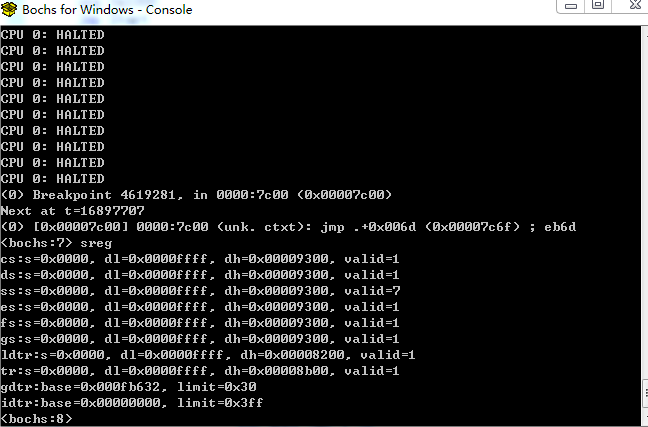
1. 输入调试命令 c 继续执行，在 0x7c00 处的断点中断。中断后会在Console 窗口中输出下一个要执行的指令，即软盘引导扇区程序的第一条指令， 如下

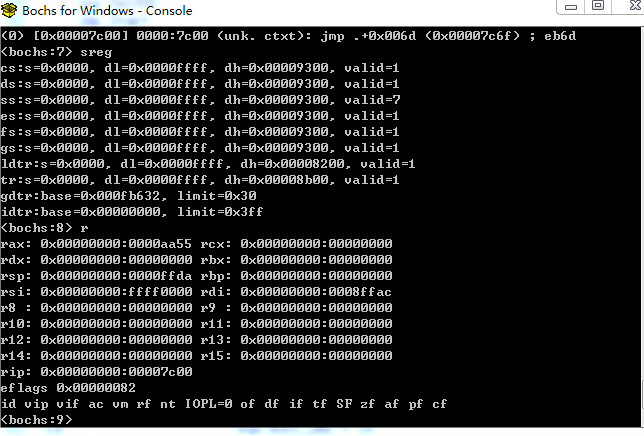
(0) [0x00007c00] 0000:7c00 (unk. ctxt): jmp .+0x006d (0x00007c6f) ; eb6d



1. 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

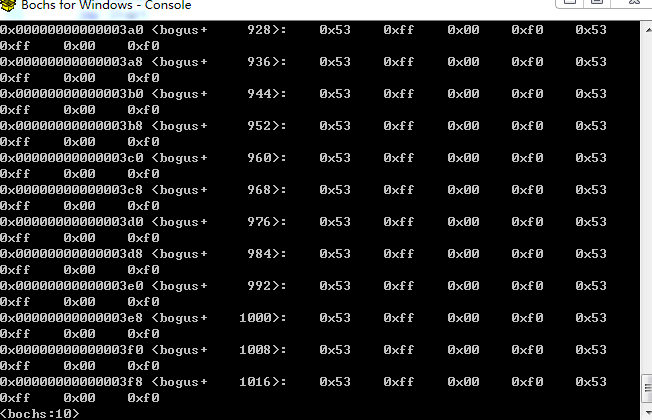
（eb6d）和此条指令要跳转执行的下一条指令的地址（括号中的 0x00007c6f）。

1. 输入调试命令 sreg 验证 CS 寄存器（0x0000）的值。
2. 输入调试命令 r 验证 IP 寄存器（0x7c00）的值。

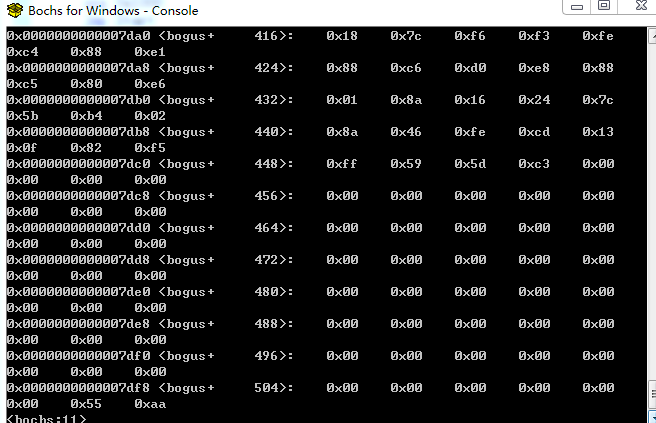


1. 由于 BIOS 程序此时已经执行完毕，输入调试命令 0 xp /1024b 0x0000

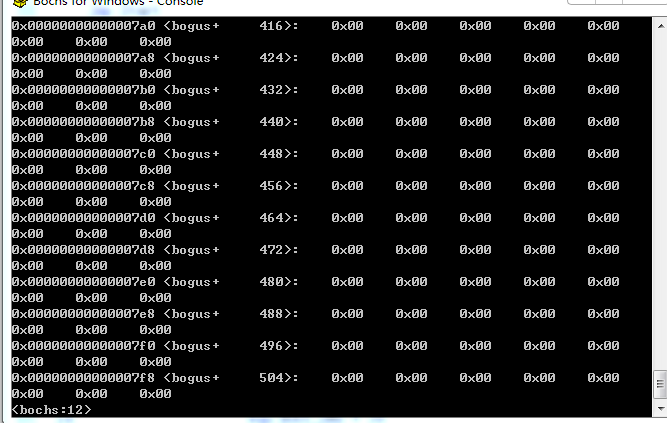
验证此时 BIOS 中断向量表已经被载入。



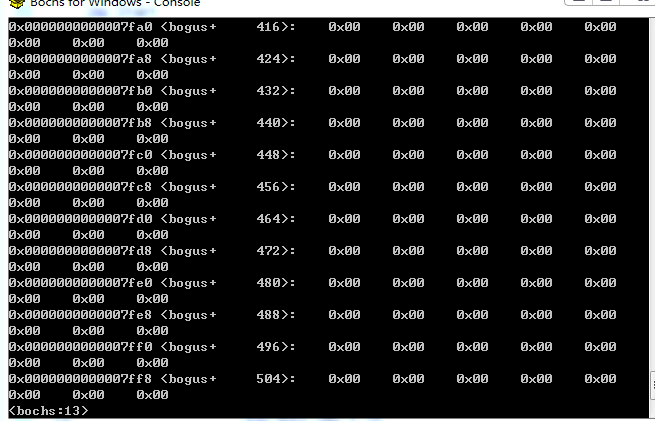
1. 输入调试命令 0 xp /512b 0x7c00 显示软盘引导扇区程序的所有字节码。观察此块内存最开始的两个字节分别为 0xeb 和 0x6d，这和引导程序第一条指令的字节码（eb6d）是相同的。此块内存最后的两个字节分别为 0x55 和0xaa，表示引导扇区是激活的，可以用来引导操作系统，这两个字节是 boot.asm 中最后一行语句 dw 0xaa55 定义的（注意，Intel 80386 CPU 使用 little- endian 字节顺序，参见附录 B）。



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

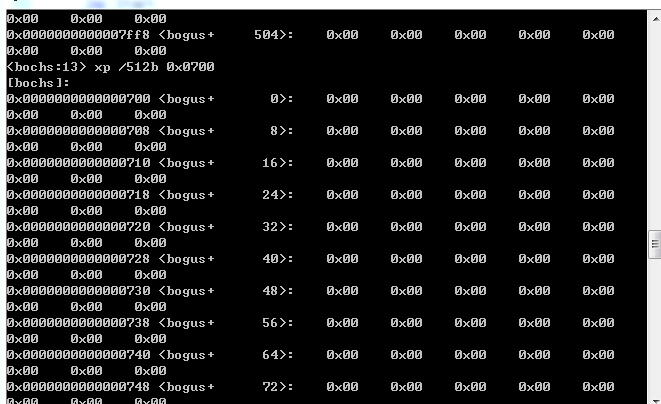


1. 输入调试命令 0 xp /512b 0x7e00 验证图 3-2 中第二个用户可用区域是空白的。

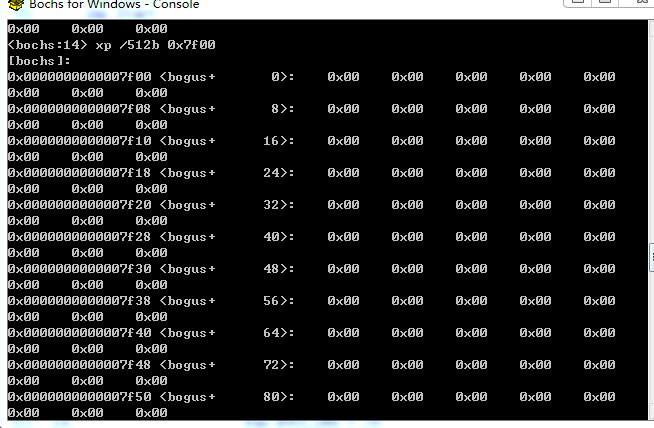


1. 自己设计两个查看内存的调试命令，分别验证这两个用户可用区域的高地址端也是空白的。

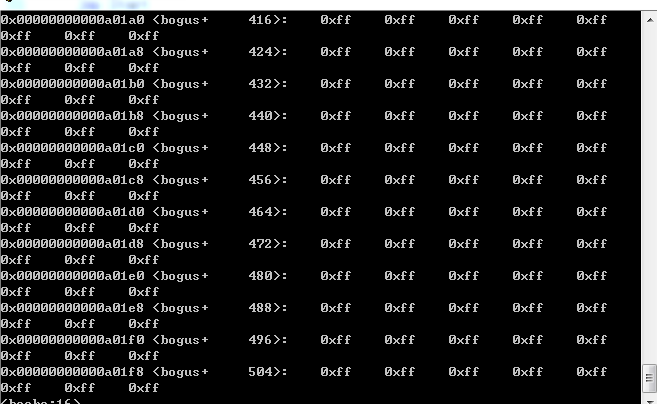
设计指令：xp /512b 0x0700



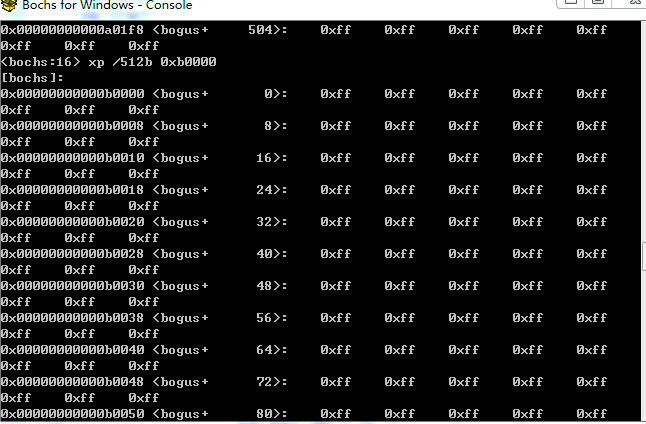
设计指令：xp /512b 0x7f00:



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

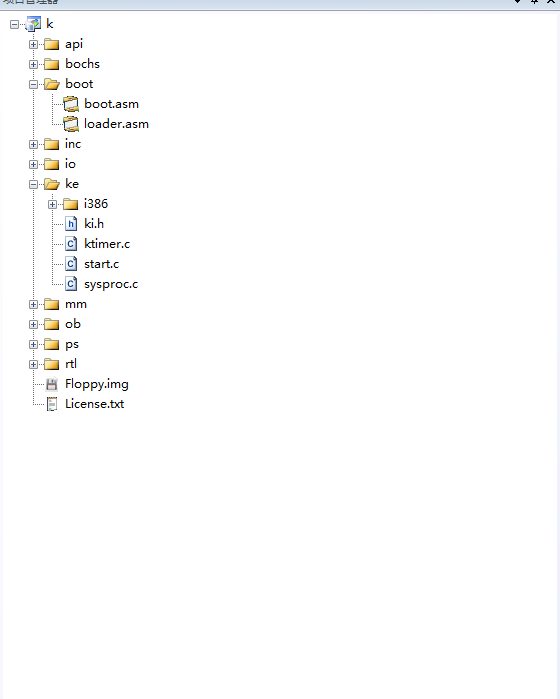


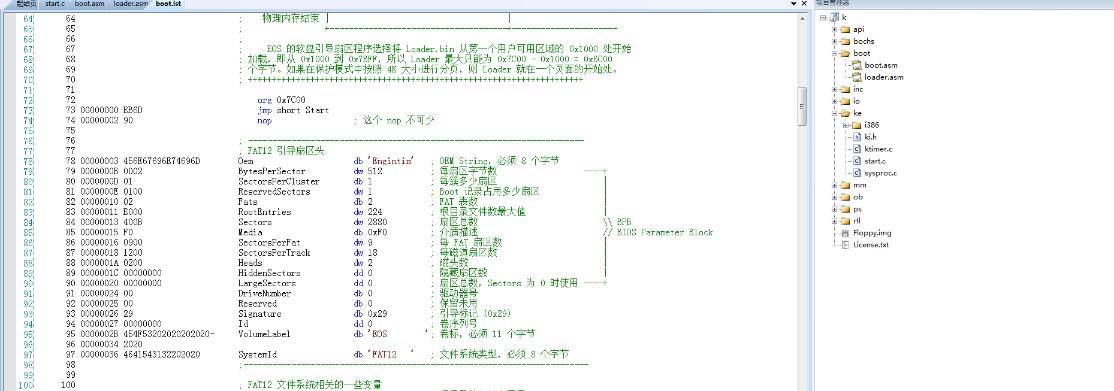
1. 自己设计一个查看内存的调试命令，验证上位内存的高地址端已经被系统占用。

设计指令 xp /512b 0xb0000：

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：

1. 选择“视图”菜单中的“项目管理器”，打开项目管理器窗口。



1. 在“项目管理器”窗口中，右键点击“boot”文件夹中的 boot.asm 文件。
2. 在弹出的快捷菜单中选择“打开生成的列表文件”，在源代码编辑器中就会打开文件 boot.lst。
3. 将 boot.lst 文件和 boot.asm 文件对比可以发现，此文件包含了

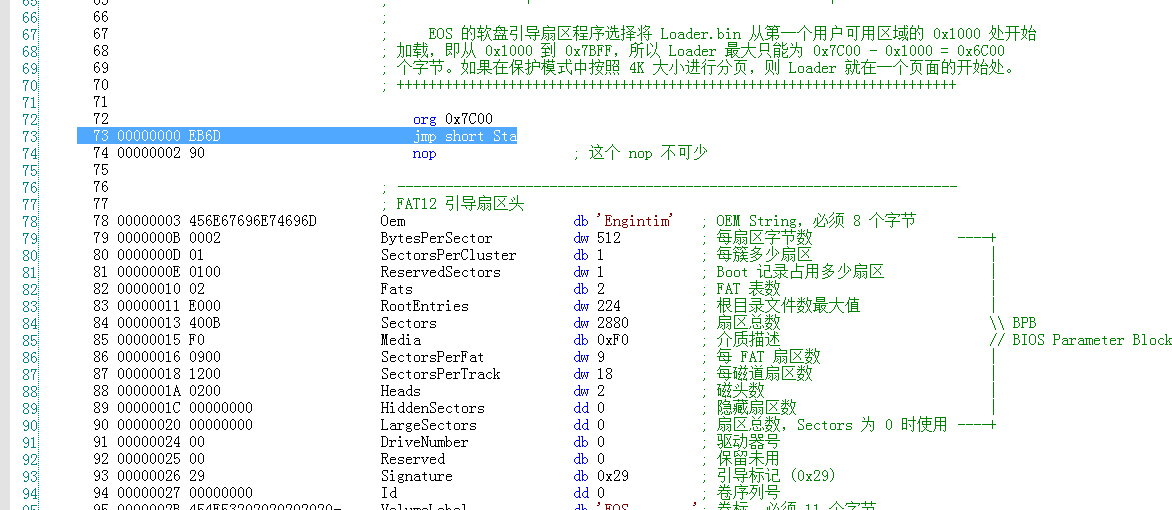
boot.asm 文件中所有的汇编代码，同时在代码的左侧又添加了更多的信息。

1. 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 00000000 EB6D jmp short Start

此行包含的信息有：73 是行号。 00000000 是此条指令相对于程序开始位置的偏移（第一条指令应该为 0）。 EB6D 是此条指令的字节码，和之前记录下来的指令字节码是一致的。

该条指令如下：

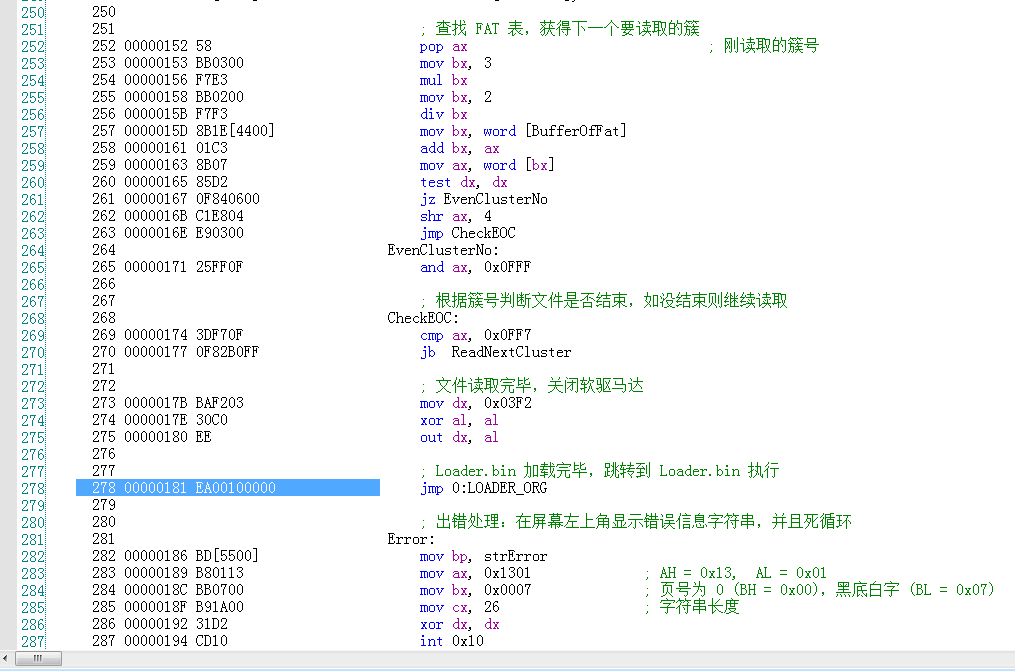


软盘引导扇区程序的主要任务就是将软盘中的 loader.bin 文件加载到物理内存的 0x1000 处，然后跳转到 loader 程序的第一条指令（物理地址 0x1000 处的指令）继续执行 loader 程序。按照下面的步骤调试此过程：

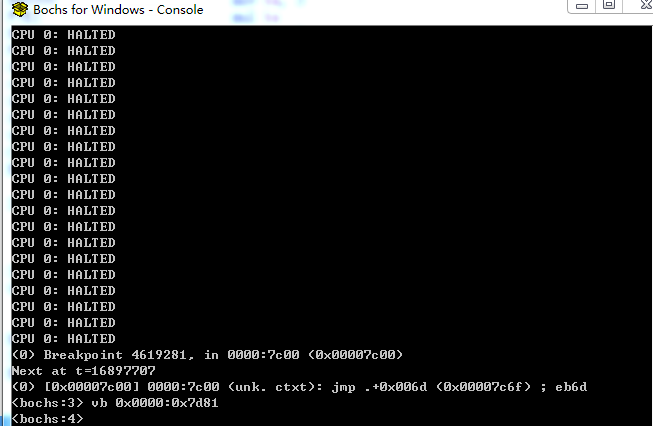
1. 在 boot.lst 文件中查找到加载完毕 loader.bin 文件后要跳转到

loader 程序中执行的指令（第 278 行）

278 00000181 EA00100000 jmp 0:LOADER\_ORG

根据此指令相对于程序开始（0x7C00）的偏移 （0x0181）可以得到此指令的逻辑地址为 0x0000:7D81。

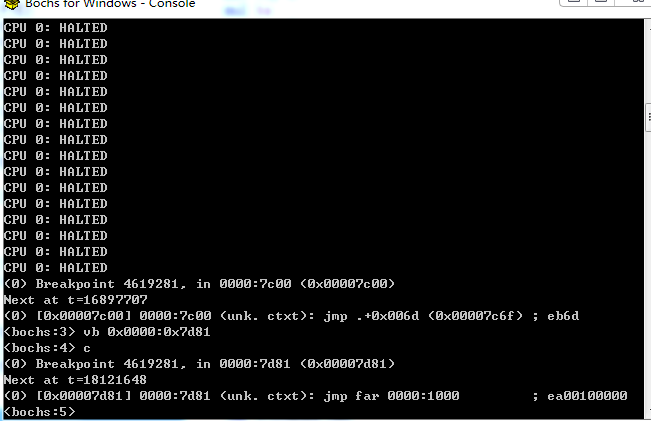
1. 输入调试命令 1 vb 0x0000:0x7d81 添加一个断点。



1. 输入调试命令 c 继续执行，到断点处中断。在 Console 窗口中显示

(0) [0x00007d81] 0000:7d81 (unk. ctxt): jmp far 0000:1000 ; ea00100000

此条指令会跳转到物理内存 0x1000 处（即 Loader 程序的第一条指令）继续执行。

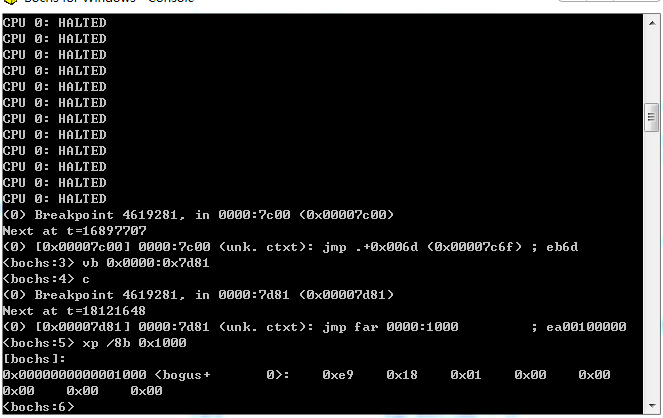


1. 按照打开 boot.lst 文件的方法打开 loader.lst 文件，并在此文件中查找到 loader 程序的第一条指令（第 33 行）

33 00000000 E91801 jmp Start

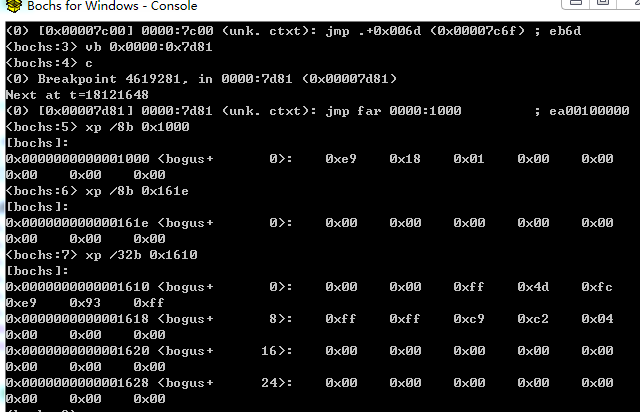


1. 输入调试命令 xp /8b 0x1000 查看内存 0x1000 处的数据，验证此块内存的前三个字节和 loader.lst 文件中的第一条指令的字节码是相同的。说明loader 程序已经加载到从地址 0x1000 开始的内存中了。



1. 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。

设计指令 1：xp /8b 0x161e ；可以发现，太短，无法看出。设计指令 2：xp /32b 0x1610 ；成功看到。



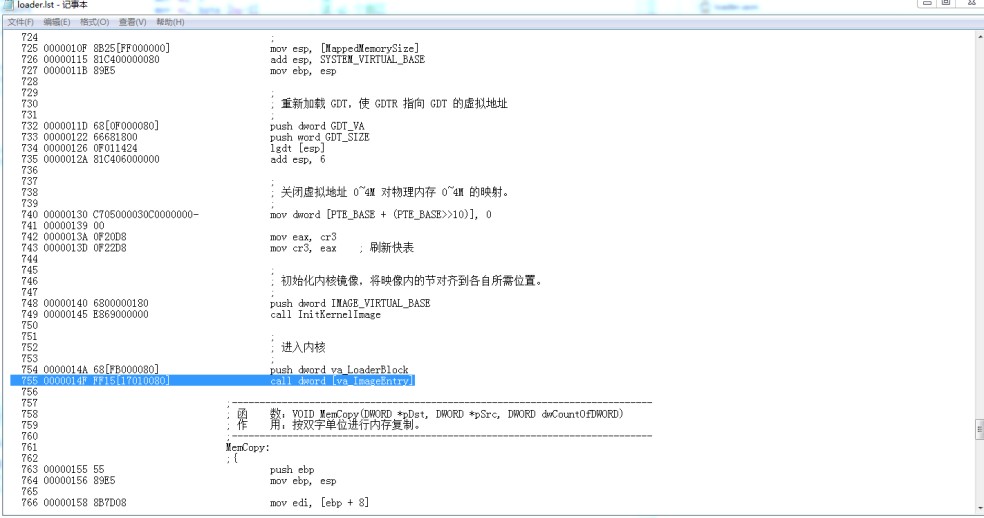
## 调试加载程序

Loader 程序的主要任务是将操作系统内核文件（kernel.dll 文件）加载到内存中，然后让 CPU 进入保护模式并且启用分页机制，最后进入操作系统内核开始执行（调用 kernel.dll 的入口点函数）。按照下面的步骤调试上述过程：

1. 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）

755 0000014F FF15[17010080] call dword [va\_ImageEntry]



1. 计算此条指令物理地址的过程要复杂一些。首先，其偏移地址 0x14f 是相对于节（节 SECTION 是 NASM 汇编中的概念）开始的。由于在 boot.asm 程序中只有一个节，所以之前计算的结果都是正确的，但是在 loader.asm 程序中有两个节，并且此条指令是在第二个节（32 位的保护模式代码）。

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）

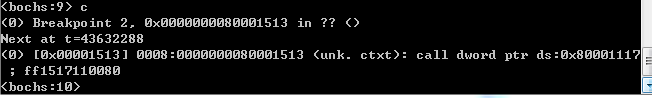
593 000003C0 C20600 ret 6

因为第一个节中最后一条指令的偏移为 0x03c0，并且此条指令长度为 3 个字节（字节码为 C20600），所以在进入第二个节之前的物理地址为 0x13c3

（0x1000+0x03c0+0x3）,由于第二个节是 32 位汇编程序，因此需要补一个字节确保四字节对齐，所以第二个节的起始物理地址是 0x13c4,从而可以计算出进入内核执行的指令所在的物理地址为 0x1513（0x13c4+0x14f）。

1. 使用添加物理地址断点的调试命令 pb 0x153 13 添加一个断点。
2. 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）：

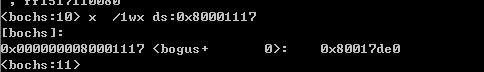
(0) [0x00001513] 0008:0000000080001513 (unk. ctxt): call dword ptr ds:0x80001117; ff1517110080

由于这里使用了函数指针的概念，所以，根据反汇编指令可以确定内核入口点函数的地址就保存在逻辑地址 ds:0x8000117 处的四个字节中。

1. 使用查看虚拟内存的调试命令 x /1wx ds:7 0x80001117 查看内存中保存的 32 位函数入口地址，

在 Console 窗口中会输出类似下面的内容：

0x0000000080001117 <bogus+ 0>: 0x800\*\*\*\*\*

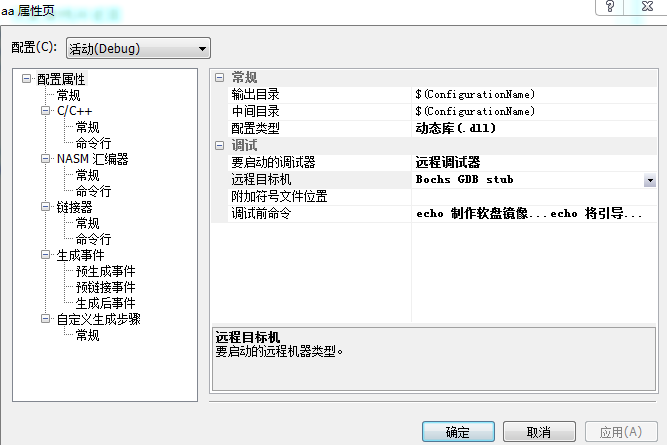
记录下此块内存中保存的函数地址，后面的实验会验证内核入口点函数的地址与此地址是一致的。

1. 选择“调试”菜单中的“停止调试”菜单项，停止调试。

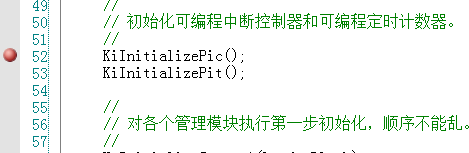
## 调试内核

通过前面的练习，读者使用 Bochs 虚拟机提供的调试功能了解了 EOS 操作系统的引导和加载过程，接下来读者可以继续调试 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”。



1. 点击“确定”按钮关闭“属性页”对话框。
2. 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。
3. 在 KiSystemStartup 函数中的代码行（第 52 行）KiInitializePic(); 添加一个断点。

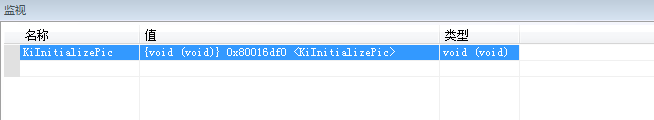


1. 按 F5 启动调试，会在刚刚添加的断点处中断。
2. 在 start.c 源代码文件中的 KiSystemStartup 函数名上点击鼠标右键，在弹出的快捷菜单中选择“添加监视”，KiSystemStartup 函数就被添加到了“监视”窗口中。在“监视”窗口中可以看到此函数地址为

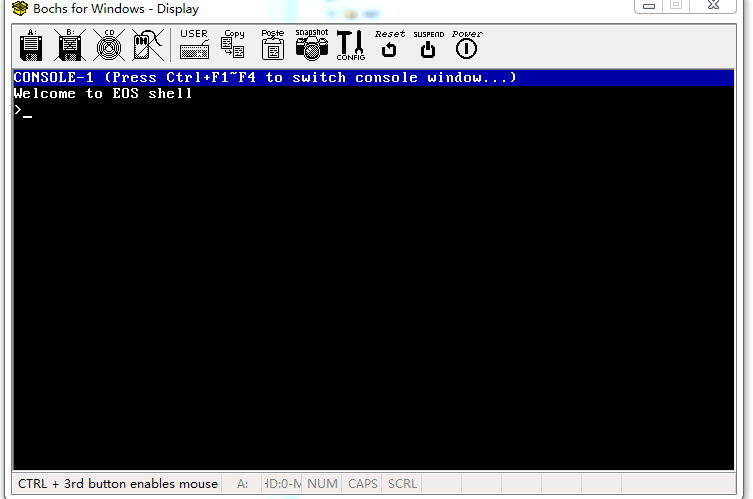
{void (PVOID)} 0x800\*\*\*\*\* <KiSystemStartup>

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。

查看“监视”窗口如下：



1. 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。



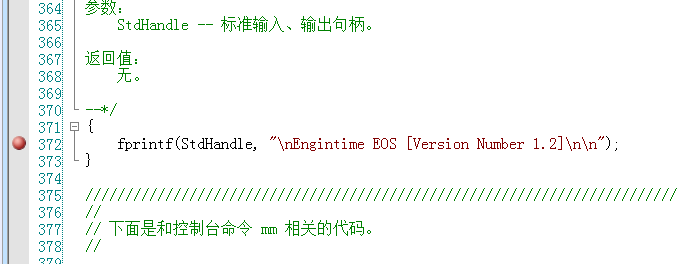
1. 选择“调试”菜单中的“删除所有断点”菜单项，删除之前添加的所有断点。
2. 选择“调试”菜单中的“停止调试”菜单项，停止调试。

## EOS 启动后的状态和行为

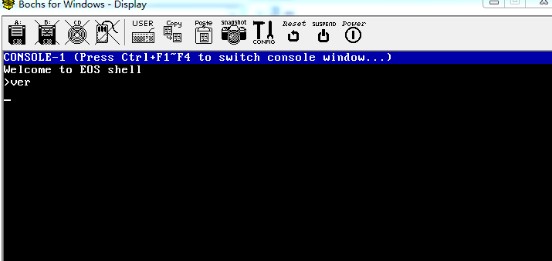
至此，读者已经完整学习了 EOS 操作系统的启动过程。当 EOS 启动完毕进入内核后，会进一步完成一系列的内核初始化工作，直到 EOS 操作系统可以接收用户输入的命令为止。因为内核初始化阶段涉及到的知识较多，不太适合刚刚接触操作系统原理的读者深入学习，所以，读者可以在全面掌握了操作系统概念后再回过头来仔细研究 EOS 内核的初始化过程，这对于读者深入理解操作系统原理也是有很大帮助的。但是，读者还是需要在这里了解一下 EOS 在完成内核初始化后，处于什么样的状态，具有什么样的行为，这对于读者顺利完成后续的实验是很重要的。

请读者按照下面的步骤进行实验：

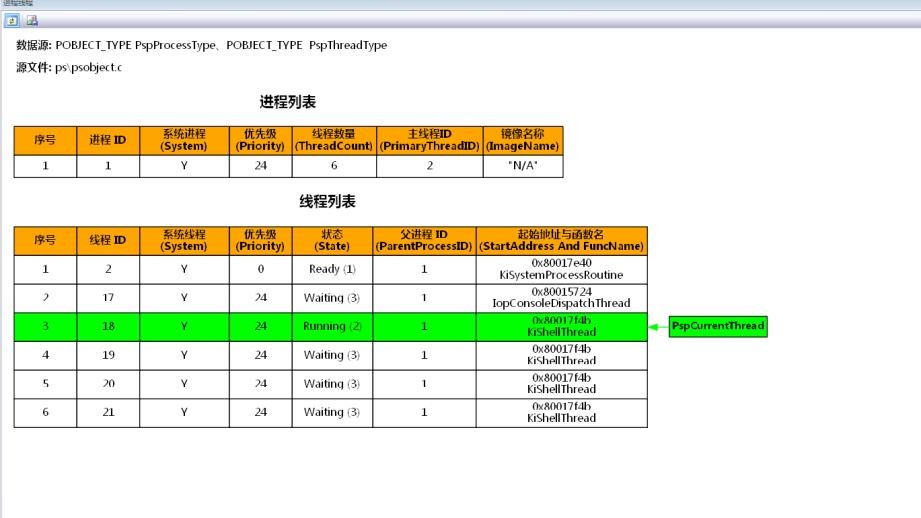
1. 在 ke/sysproc.c 文件的第 372 行，也就是“ver”命令函数中添加一个断点。



1. 按 F5 启动调试。
2. 待 EOS 启动完成后，在控制台中输入命令“ver”后按回车，会在刚刚添加的断点处中断。



1. 刷新进程线程窗口，会得到如图 10-5 所示的内容。刷新进程线程窗口后：



有必要对图 10-5 中显示的进程和线程信息进行一下详细的说明。此时，在进程列表中只有一个 ID 为 1 的系统进程，其优先级为 24，包含有 6 个线程， 其中 ID 为 2 的线程是该进程的主线程，系统进程是没有映像名称的，或者可以认为“kernel.dll”就是系统进程的镜像名称。在线程列表中有 6 个线程，

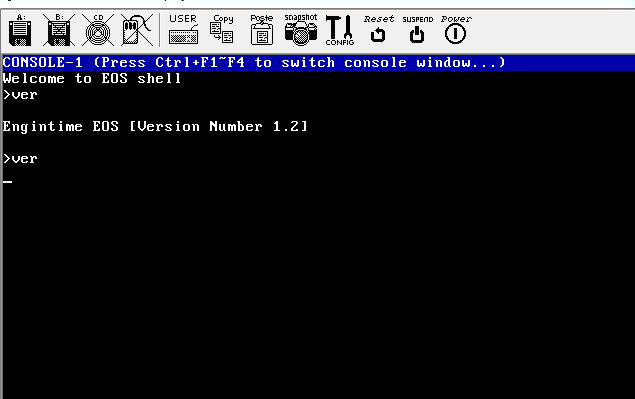
它们都是系统线程。其中优先级为 0 的是空闲线程，当没有优先级大于 0 的线程占用处理器时，空闲线程就会在处理器上运行并处于运行状态（Running）， 否则就处于就绪状态（Ready），随时准备继续在处理器上运行。

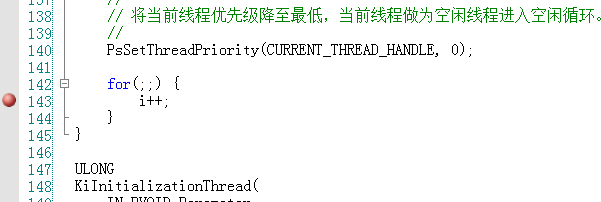
ID 为 17 的线程是控制台派遣线程，用于将键盘事件派遣到活动的控制台线程，所以在没有键盘事件发生的时间里，该线程总是处于阻塞状态（Waiting）， 等待键盘事件的到来。余下的四个线程都是控制台线程，分别对应于四个控制台， 由于它们执行的是同一个控制台线程函数（ ke/sysproc.c 文件中的KiShellThread 函数），所以它们的起始地址都是相同的。

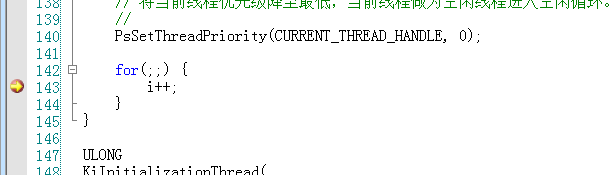
控制台线程只有在执行控制台命令的时候才会处于运行状态，其它时间它们都在等待控制台派遣线程为它们分配键盘事件，会处于阻塞状态。由于本次是在控制台 1 中执行的 ver 命令，所以控制台 1 对应的 ID 号为 18 的线程会处

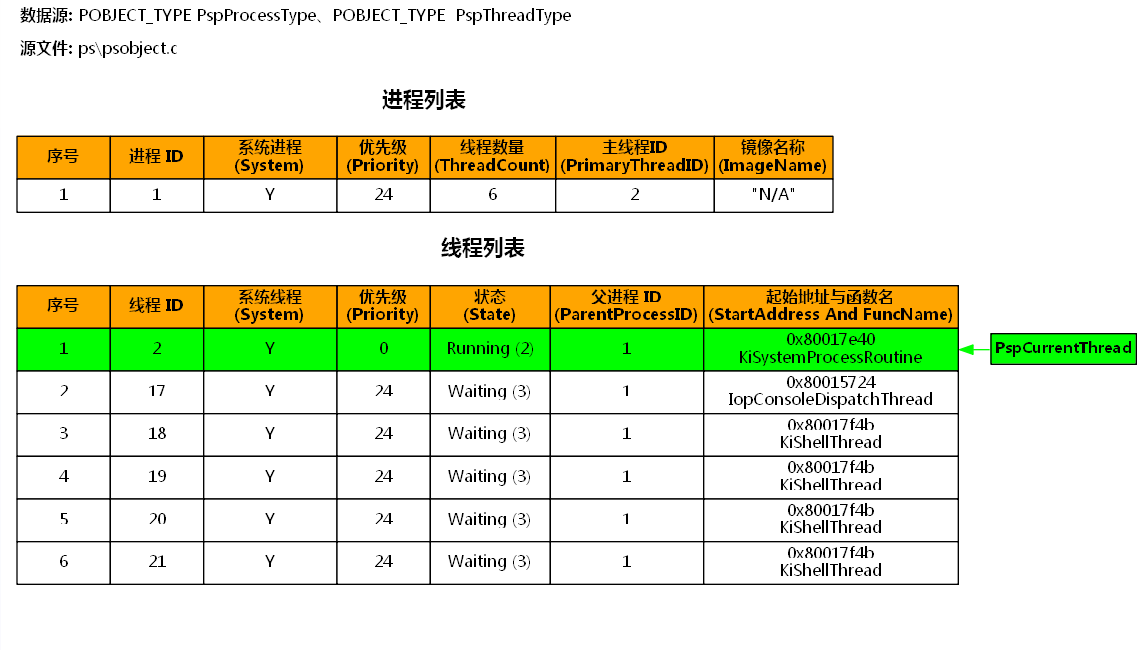
于运行状态（使用绿色底色，并且由当前线程指针 PspCurrentThread 指向）， 而其它的三个控制台线程都处于阻塞状态。

读者可以按照下面的步骤进行验证：

1. 在控制台 1 中执行“ver”命令，命中之前添加的断点。
2. 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。



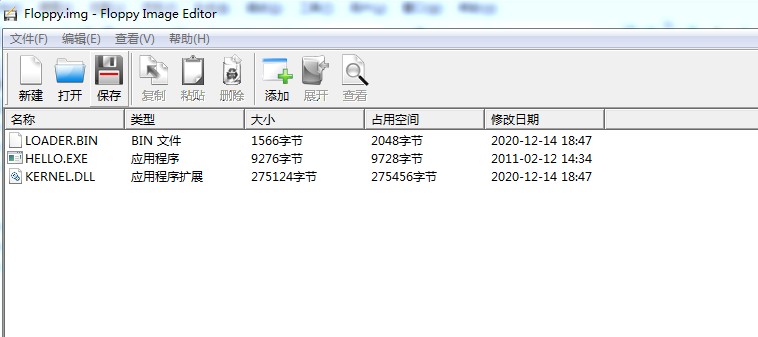
1. 按 F5 继续运行，会命中刚刚添加的断点。
2. 刷新进程线程窗口，验证空闲线程此时处于运行状态，并注意观察其它线程的状态。



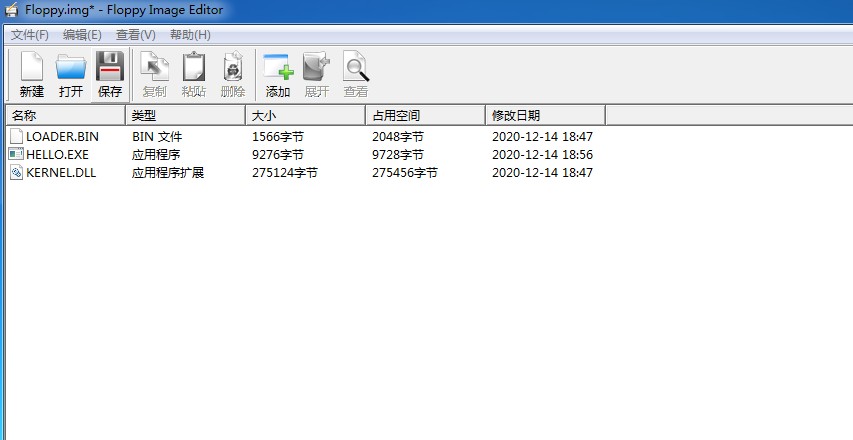
之前都是让 EOS 运行一个内核命令，进而查看进程线程信息。接下来，读者可以按照下面的步骤查看

当一个 EOS 应用程序正在运行时的进程和线程信息：

1. 在 OS Lab 中选择“调试”菜单中的“停止调试”，结束之前的调试。
2. 删除所有断点。
3. 在 OS Lab “项目管理器”窗口中双击 Floppy.img 文件，使用

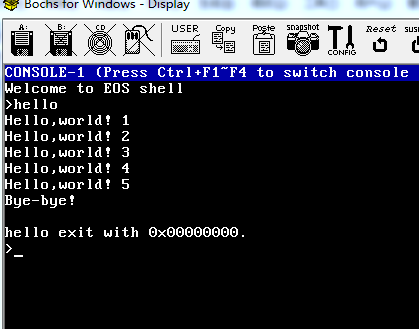
FloppyImageEditor 工具打开此软盘镜像文件。

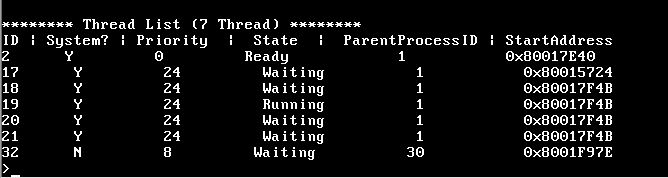
1. 打开配套资源“学生包”，在其中找到本实验对应的文件夹。可以在本书前言部分找到“学生包”的下载地址。
2. 将本实验文件夹中的 Hello.exe 文件拖动到 FloppyImageEditor 工具窗口的文件列表中释放，Hello.exe 文件即被添加到软盘镜像文件中。Hello.exe 一个 EOS 应用程序，其源代码可以参见本实验文件夹中的 Hello.c 源文件。



1. 在 FloppyImageEditor 中选择“ 文件” 菜单中的“ 保存” 后关闭

FloppyImageEditor。

1. 按 F5 启动调试。
2. 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。
3. 迅速按 Ctrl+F2 切换到控制台 2，并输入命令“pt”后按回车。控制台 2 输出的进程和线程信息如下图所示：



最后请读者按照实验 1 第 3.6 节中的内容自己新建一个 EOS 应用程序项目，在 main 函数中添加一个断点后启动 EOS，待命中断点后刷新进程线程窗口， 显示内容：



# 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：boot.bin 是软盘引导扇区程序，大小为 512 字节，由于 EOS 从软盘启动的过程比较复杂，必然会造成引导程序过大，显然不能将引导程序全部放入软盘引导扇区中，这样就需要在 loader.bin 程序中完成部分功能。

boot.bin 程序的主要功能是将软盘根目录中的 loader.bin 文件加载到物理内存 0x1000 处，然后跳转执行 loader.bin 程序的第一条指令，继续启动 EOS。Loader.bin 程序的主要功能是将软盘根目录中 kernel.dll 文件加载到内存中， 然后启动保护模式和分页机制，最后跳转到 kernel.dll 的入口点函数执行。

* 1. 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将 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：软盘引导扇区程序执行时，CPU 处于实模式状态，CS 段寄存器为 0，不改变段寄存器（全为 0）值的情况下，只能访问 0x0000-0xFFFF 的地址空间，再加上程序受到 512 字节限制，不希望出现修改段寄存器代码的情况，最好将loader.bin 文件加载到 0x0000-0xFFFF 的地址空间中。

为了保持 4K 地址对齐，要从 0x1000 处开始加载，这样在后面启动分页机制后，loader 程序就是从一页的 0 偏移处开始的。

* 1. 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：loder 程序结束位置字节码的调试命令：如果 loder.bin 文件的大小为

1566 个字节，转换为十六进制就是 61E，程序最后的 8 个字节就在物理内存

0x1616-0x161D 的位置，查看这 8 个字节的调试命令为 xp /8b 0x1616。若loder.bin 文件的大小发生变化，调试命令也要随之变化。

# 总结和感想体会

（1）通过在OSLab上新建EOS项目并跟踪调试程序让我明白了如何启动系统并在进程线程窗口进行查看，通过调试hs Debug 远程目标机和 BIOS 程序，让我对EOS系统的调试有了一个更加深入的理解。

（2）通过观察进程进行的调试窗口，验证老师在操作系统课上对进程的讲解，也进一步巩固了我在上课学到的知识，这次实验让我熟悉了OSlab软件的使用，并且在启动系统中调试的过程中让我对进程有了一个更深入的理解，这次实验让我学到了很多，开阔了我的视野。