第 14 章 实验环境设置与使用方法 1
14.1 概述 1
14.2 Bochs仿真系统 1
14.2.1 设置Bochs系统 2
14.2.2 配置文件bochsrc 2
14.2.3 调试功能 5
14.3 创建磁盘映象文件 5
14.3.1 利用Bochs软件自带的Image生成工具 6
14.3.2 在Linux系统下使用dd命令创建Image文件。 7
14.3.3 利用WinImage创建DOS格式的软盘Image文件 8
14.4 访问磁盘映象文件中的信息 8
14.4.1 使用WinImage工具软件 8
14.4.2 利用现有Linux系统 9
14.5 制作根文件系统 11
14.5.1 根文件系统和根文件设备 11
14.5.2 创建文件系统 11
14.5.3 Linux-0.11 的Bochs配置文件 13
14.5.4 在hdc.img上建立根文件系统 15
14.5.5 使用硬盘Image上的根文件系统 16
14.6 在Linux 0.11 系统上编译 0.11 内核 17
14.7 在Redhat 9 系统下编译Linux 0.11 内核 18
14.7.1 修改makefile文件 19
14.7.2 修改汇编程序中的注释 19
14.7.3 内存位置对齐语句align值的修改 19
14.7.4 修改嵌入宏汇编程序 20
14.7.5 c程序变量在汇编语句中的引用表示 20
14.7.6 保护模式下调试显示函数 20
14.8 利用bochs调试内核 21

第14章 实验环境设置与使用方法

14.1 概述

为了配合 Linux 0.11 内核工作原理的学习，本章介绍了利用 PC 机仿真软件和在实际计算机上运行 Linux 0.11 系统的方法。其中包括内核的编译过程、PC 仿真环境下文件的访问和复制、引导盘和根文件系统的制作方法以及 Linux 0.11 系统的使用方法。最后还说明了如何对内核代码作少量语法修改，使其在现有的 RedHat 9 系统（gcc 3.x）下能顺利通过编译。

在开始进行实验之前，首先准备好一些有用的工具软件。在 Windows 平台上，可以准备以下几个软件：

- Bochs 2.x 开放源代码的 PC 机仿真软件包。
- UltraEdit 超级编辑器。可用来编辑二进制文件。
- WinImage DOS 格式软盘映象文件读写软件。

运行 Linux 0.11 系统的最佳方法是使用 PC 仿真软件。目前世面上流行的 PC 仿真软件系统主要有 3 种：VMware 公司的 VMware Workstation、Connectix 公司的 Virtual PC（现在该软件已被微软收购）和开放源代码的 Bochs（发音与'box'相同）。这 3 种软件都虚拟或仿真了 Intel x86 硬件环境，可以让我们在运行这些软件的系统平台上运行多种其它的“客户”操作系统。

就使用范围和运行性能来说，这 3 个仿真软件还是有一定的区别。Bochs 仿真了 x86 的硬件环境及其外围设备，因此很容易被移植到很多操作系统上或者不同体系结构的平台上。由于主要使用了仿真技术，其运行性能和速度都要比其它两个软件要慢很多。Virtual PC 的性能则界于 Bochs 和 VMware Workstation 之间。它仿真了 x86 的大部分，而其它部分则采用虚拟技术来实现。VMware Workstation 仅仿真了一些 I/O 功能，而所有其它部分则是在 x86 实时硬件上直接执行。也就是说当客户操作系统在要求执行一条指令时，VMware 不是用仿真方法来模拟这条指令，而是把这条指令“传递”给实际系统的硬件来完成。因此 VMware 是 3 种软件中运行速度和性能最高的一种。有关这 3 种软件之间的具体区别和性能差异，请参考网上的一篇评论文章（http://www.osnews.com/story.php?news_id=1054）。

从应用方面来看，如果仿真环境主要是用于应用程序开发，那么 VMware Workstation 和 Virtual PC 可能是比较好的选择。但是如果需要开发一些低层系统软件（比如进行操作系统开发和调试、编译系统开发等），那么 Bochs 就是一个很好的选择。使用 Bochs，你可以知道被执行程序在仿真硬件环境中的具体状态和精确时序，而非实际硬件系统执行的结果。这也是为什么很多操作系统开发者更倾向于使用 Bochs 的原因。因此本章主要介绍利用 Bochs 仿真环境运行 Linux 0.11 的方法。目前，Bochs 网站名是 http://sourceforge.net/projects/bochs/。你可以从上面下载到最新发布的 Bochs 软件系统以及很多已经制作好的可运行磁盘映象文件。

14.2 Bochs 仿真系统

Bochs 是一个能完全仿真 Intel x86 计算机的程序。它可以被配置成仿真 386、486、Pentium 或以上的新型 CPU。在执行的整个过程，Bochs 仿真了所有的指令，并且含有标准 PC 机外设所有的设备模块。

由于 Bochs 仿真了整个 PC 环境，因此在其中执行的软件会“认为”它是在一个真实的机器上运行。这种完全仿真的方法使得我们能在 Bochs 下不加修改地运行大量的软件系统。

Bochs 是 Kevin Lawton 于 1994 年开始采用 C++语言开发的软件系统，被设计成能够在 Intel x86、PPC、Alpha、Sun 和 MIPS 硬件上运行。不管运行的主机采用的是何硬件平台，Bochs 仍然仿真 x86 的软件。这是其它两种仿真软件所不能做到的。为了在被模拟的机器上执行执行任何活动，Bochs 需要与主机操作系统进行交互。当在 Bochs 显示窗口中按下一键时，一个击键事件就会发送到键盘的设备处理模块中。当被模拟的机器需要从模拟的硬盘上执行读操作时，Bochs 就会从主机上的硬盘映象文件中执行读操作。

Bochs 软件的安装非常方便。你可以直接从 bochs.sourceforge.net 网站上下载到 Bochs 安装软件包。如果你所使用的计算机操作系统是 Windows，则其安装过程与普通软件完全一样。安装好后会在 C 盘上生成一个目录：'C:\Program Files\Bochs-2.1.1\'（其中版本号随不同的版本而不同）。如果你的系统是 RedHat 9 或其它 Linux 系统，你可以下载 Bochs 的 RPM 软件包并按如下方法来安装：

```
user$ su
Password:
root# rpm -i bochs-1.2.1.i386.rpm
root# exit
user$ _
```

安装时需要有 root 权限，否则你就得在自己的目录下重新编译 Bochs 系统。另外，Bochs 需要在 X11 环境下运行，因此系统中必须已经安装了 X Windows 系统才能使用 Bochs。在安装好后，可以先使用 Bochs 中自带的 Linux dlx 程序包来测试和熟悉一下系统。另外，也可以从 Bochs 网站上下载一些已经制作好的 Linux 磁盘映象文件。建议下载 Bochs 网站上的一个 SLS Linux 模拟系统：sls-0.99pl.tar.bz2 作为创建 Linux 0.11 模拟系统的辅助平台。在制作新的硬盘映象文件时需要借助这些系统对硬盘映象文件进行分区和格式化操作。

有关重新编译 Bochs 系统或把 Bochs 安装到其它硬件平台上的操作方法，请参考 Bochs 用户手册中的相关说明。

14.2.1 设置 Bochs 系统

为了在 Bochs 中运行一个操作系统，最少需要以下一些资源或信息：

- bochs 执行文件；
- bios 映象文件（通常称为'BIOS-bochs-latest'）；
- vga bios 映象文件（例如，'VGABIOS-lgpl-latest'）；
- 至少一个引导启动磁盘映象文件（软盘、硬盘或 CDROM 的映象文件）。

但是我们在使用过程中往往需要为运行系统预先设置一些参数。这些参数可以在命令行上传递给 Bochs 执行程序，但通常我们都使用一个配置文件（例如 Sample.bxrc）为专门的一个应用来设置。下面说明在 Windows 环境下配置文件的设置方法。

14.2.2 配置文件 bochsrc

Bochs 使用配置文件中的信息来寻找所使用的磁盘映象文件、运行环境外围设备的配置以及其它一些模拟机器的设置信息。每个被仿真的系统都需要设置一个相应的配置文件。若所安装的 Bochs 系统是 2.1 或以后版本，那么 Bochs 系统会自动识别后缀是'.bxrc'的配置文件，并且在双击该文件图标时就会自动启动 Bochs 系统运行。例如，我们可以把配置文件名取为'bochsrc-0.11.bxrc'。在 Bochs 安装的主

目录下有一个名称为'bochsrc-sample.txt'的样板配置文件，其中列出了所有可用的的参数设置，并带有详细的说明。下面简单介绍几个在实验中经常要修改的参数。

1. megs

用于设置模拟系统所含内存容量。默认值是 32MB。例如，如果要把模拟机器设置为含有 128MB 的系统，则需要在配置文件中含有如下一行信息：

```
megs: 128
```

2. floppya（floppyb）

floppya 表示第一个软驱，floppyb 代表第二个软驱。如果需要从一个软盘上来引导系统，那么 floppya 就需要指向一个可引导的磁盘。若想使用磁盘映象文件，就在后面写上磁盘映象文件的名称。在许多操作系统中，Bochs 可以直接读写主机系统的软盘驱动器。若要访问这些实际驱动器中的磁盘，就使用设备名称（Linux 系统）或驱动器号（Windows 系统）。还可以使用 status 来表明磁盘的插入状态。ejected 表示未插入，inserted 表示磁盘已插入。下面是几个例子，其中所有盘均为已插入状态。

```
floppya: 1_44=/dev/fd0, status=inserted                 # Linux 系统下直接访问 1.44MB A 盘。
floppya: 1_44=b:, status=inserted                       # win32 系统下直接访问 1.44MB B 盘。
floppya: 1_44=bootimage.img, status=inserted            # 指向磁盘映象文件 bootimage.img。
floppyb: 1_44=..\Linux\rootimage.img, status=inserted   # 指向上级目录 Linux/下 rootimage.img。
```

在配置文件中，若同时存在几行相同名称的参数，那么只有最后一行的参数起作用。

3. ata0、ata1、ata2、ata3

这 4 个参数名用来启动模拟系统中最多 4 个 ATA 通道。对于每个启用的通道，必须指明两个 IO 基地址和一个中断请求号。默认情况下只有 ata0 是启用的，并且参数默认为下面所示的值：

```
ata0: enabled=1, ioaddr1=0x1f0, ioaddr2=0x3f0, irq=14
ata1: enabled=1, ioaddr1=0x170, ioaddr2=0x370, irq=15
ata2: enabled=1, ioaddr1=0x1e8, ioaddr2=0x3e0, irq=11
ata3: enabled=1, ioaddr1=0x168, ioaddr2=0x360, irq=9
```

4. ata0-master（ata0-slave）

ata0-master 用来指明模拟系统中第 1 个 ATA 通道（0 通道）上连接的第 1 个 ATA 设备（硬盘或 CDROM 等）；ata0-slave 指明第 1 个通道上连接的第 2 个 ATA 设备。例子如下所示，其中，设备配置的选项含义如下表所示。

```
ata0-master: type=disk, path=hd.img, mode=flat, cylinders=306, heads=4, spt=17, translation=none
ata1-master: type=disk, path=2G.cow, mode=vmware3, cylinders=5242, heads=16, spt=50, translation=echs
ata1-slave:  type=disk, path=3G.img, mode=sparse, cylinders=6541, heads=16, spt=63, translation=auto
ata2-master: type=disk, path=7G.img, mode=undoable, cylinders=14563, heads=16, spt=63, translation=lba
ata2-slave:  type=cdrom, path=iso.sample, status=inserted
ata0-master: type=disk, path="hdc-large.img", mode=flat, cylinders=487, heads=16, spt=63
ata0-slave:  type=disk, path="..\hdc-0.11.img", mode=flat, cylinders=121, heads=16, spt=63
```

选项	说明	可取的值

type	连接的设备类型	[disk | cdrom]
path	映象文件路径名	
mode	映象文件类型，仅对 disk 有效	[flat | concat | external | dll | sparse | vmware3 | undoable | growing | volatile]
cylinders	仅对 disk 有效	
heads	仅对 disk 有效	
spt	仅对 disk 有效	
status	仅对 cdrom 有效	[inserted | ejected]
biosdetect	bios 检测类型	[none | auto],仅对 ata0 上 disk 有效 [cmos]
translation	bios 进行变换的类型(int13),仅对 disk 有效	[none | lba | large | rechs | auto]
model	确认设备 ATA 命令返回的字符串	

在配置 ATA 设备时，必须指明连接设备的类型 type，可以是 disk 或 cdrom。还必须指明设备的“路径名”path。“路径名”可以是一个硬盘映象文件、CDROM 的 iso 文件或者直接指向系统的 CDROM 驱动器。在 Linux 系统中，可以使用系统设备作为 Bochs 的硬盘，但由于安全原因，在 windows 下不赞成直接使用系统上的物理硬盘。

对于类型是 disk 的设备，选项 path、cylinders、heads 和 spt 是必须的。对于类型是 cdrom 的设备，选项 path 是必须的。

磁盘变换方案（在传统 int13 bios 功能中实现，并且用于象 DOS 这样的老式操作系统）可以定义为：

- none：无需变换，适用于容量小于 528MB（1032192 个扇区）的硬盘；
- large：标准比特移位算法，用于容量小于 4.2GB（8257536 个扇区）的硬盘；
- rechs：修正移位算法，使用 15 磁头的伪物理硬盘参数，适用于容量小于 7.9GB（15482880 个扇区）的硬盘；
- lba：标准 lba-辅助算法。适用于容量小于 8.4GB（16450560 个扇区）的硬盘；
- auto：自动选择最佳变换方案。（如果模拟系统启动不了就应该改变）。

mode 选项用于说明如何使用硬盘映象文件。它可以是以下模式之一：

- flat：一个平坦顺序文件；
- concat：多个文件；
- external：由开发者专用，通过 C++类来指定；
- dll：开发者专用，通过 DLL 来使用；
- sparse：可堆砌的、可确认的、可退回的；
- vmware3：支持 vmware3 的硬盘格式；
- undoable：具有确认重做的平坦文件；
- growing：容量可扩展的映象文件；
- volatile：具有易变重做的平坦文件。

以上选项的默认值是：

mode=flag, biosdetect=auto, translation=auto, model=”Generic 1234”

5. boot

boot 用来定义模拟机器中用于引导启动的驱动器。可以指定是软盘、硬盘或者 CDROM。也可以使用驱动器号’c’和’a’。例子如下：

```
boot: a
boot: c
boot: floppy
```

```
boot: disk
boot: cdrom
```

6. ips

指定每秒钟仿真的指令条数。这是 Bochs 在主机系统中运行的 IPS 数值。这个值会影响模拟系统中与时间有关的很多事件。例如改变 IPS 值会影响到 VGA 更新的速率以及其它一些模拟系统评估值。因此需要根据所使用的主机性能来设定该值。可参考下表进行设置。

速度	机器配置	IPS 典型值
650Mhz	Athlon K-7 with Linux 2.4.x	2 to 2.5 million
400Mhz	Pentium II with Linux 2.0.x	1 to 1.8 million
166Mhz	64bit Sparc with Solaris 2.x	0.75 million
200Mhz	Pentium with Linux 2.x	0.5 million

例如：

```
ips: 1000000
```

7. log

指定 log 的路径名可以让 Bochs 记录执行的一些日志信息。如果在 Bochs 中运行的系统发生不能正常运行的情况就可以参考其中的信息来找出基本原由。log 通常设置为：

```
log: bochsout.txt
```

14.2.3 调试功能

参见 bochs Debugger

14.3 创建磁盘映象文件

磁盘映象文件（Disk Image File）是软盘或硬盘上信息的一个完整映象，并以文件的形式保存。磁盘映象文件中存储信息的格式与对应磁盘上保存信息的格式完全一样。空磁盘映象文件是容量与我们创建的磁盘相同但内容全为 0 的一个文件。这些空映象文件就象刚买来的新软盘或硬盘，还需要经过分区或/以及格式化才能使用。

在制作磁盘映象文件之前，我们首先需要确定所创建映象文件的容量。对于软盘映象文件，各种规格（1.2MB 或 1.44MB）的容量都是固定的。因此这里主要说明如何确定自己需要的硬盘映象文件的容量。普通硬盘的结构由堆积的金属圆盘组成。每个圆盘的上下两面用于保存数据，并且以同心圆的方式把整个表面划分成一个个磁道，或称为柱面（Cylinder）。因此一个圆盘需要一个磁头（Head）来读写上面的数据。在圆盘旋转时磁头只需要作径向移动就可以在任何磁道上方移动，从而能够访问圆盘表面所有有效的位置。每个磁道被划分成若干个扇区，扇区长度一般由 256 -- 1024 字节组成。对于大多数系统来说，通常扇区长度均为 512 字节。一个典型的硬盘结构见下图所示。

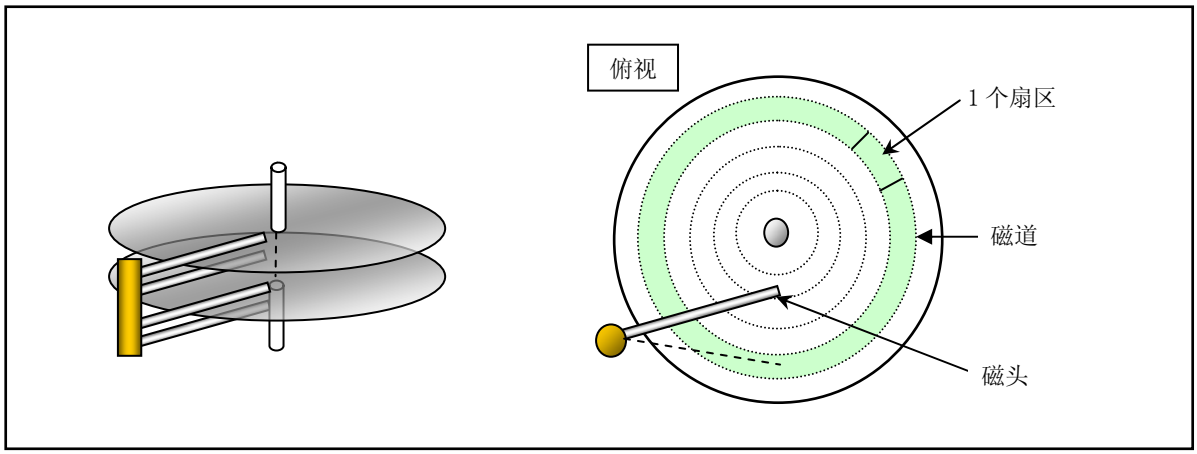

图14.1 典型硬盘内部结构

图中示出了具有两个金属圆盘的硬盘结构。因此该硬盘有 4 个物理磁头。所含的最大柱面数在生产时已确定。当对硬盘进行分区和格式化时，圆盘表面的磁介质就被初始化成指定格式的数据，从而每个磁道（或柱面）被划分成指定数量的扇区。因此这个硬盘的总扇区数为：

硬盘总扇区数 = 物理磁道数 X 物理磁头数 X 每磁道扇区数

硬盘中以上这些实际的物理参数与一个操作系统中所使用的参数会有区别，称为逻辑参数。但这些参数所计算出的总扇区数与硬盘物理参数计算出的肯定是相同的。由于在设计 PC 机系统时没有考虑到硬件设备性能和容量发展得如此之快，ROM BIOS 某些表示硬盘参数所使用的比特位太少而不能符合实际硬盘物理参数的要求。因此目前操作系统或机器 BIOS 中普遍采用的措施就是在保证硬盘总扇区数相等的情况下适当调整磁道数、词头数和每磁道扇区数，以符合兼容性和参数表示限制的要求。在 Bochs 配置文件有关硬盘设备参数中的变换（Translation）选项也是为此目的而设置的。

在我们为 Linux 0.11 系统制作硬盘 Image 文件时，考虑到其本身代码量很少，而且所使用的 MINIX 1.5 文件系统最大容量为 64MB 的限制，因此每个硬盘分区大小最大也只能是 64MB。另外，Linux 0.11 系统尚未支持扩展分区，因此对于一个硬盘 Image 文件来说，最多有 4 个分区。因此，Linux 0.11 系统可使用的硬盘 Image 文件最大容量是 64 x 4 = 256MB。在下面的说明中，我们将以创建一个具有 4 个分区、每个分区为 60MB 的硬盘 Image 文件为例子进行说明。

对于软盘来说，我们可以把它看作是一种具有固定磁道数（柱面数）、磁头数和每磁道扇区数（spt - Sectors Per Track）的超小型硬盘。例如容量是 1.44MB 的软盘参数是 80 个磁道、2 个磁头和每磁道有 18 个扇区、每个扇区有 512 字节。其扇区总数是 2880，总容量是 80 x 2 x 18 x 512 = 1474560 字节。因此下面介绍的所有针对硬盘映象文件的制作方式都可以用来制作软盘映象文件。为了叙述上的方便，在没有特别指出时，我们把所有磁盘映象文件统称为 Image 文件。

14.3.1 利用 Bochs 软件自带的 Image 生成工具

Bochs 系统带有一个 Image 生成工具“Disk Image Creation Tool”（bximage.exe）。用它可以制作软盘和硬盘的空 Image 文件。在运行并出现了 Image 创建界面时，程序首先会提示选择需要创建的 Image 类型（硬盘 hd 还是软盘 fd）。若是创建硬盘，还会提示输入硬盘 Image 的 mode 类型。通常只需要选择其默认值 flat 即可。然后输入你需要创建的 Image 容量。程序会显示对应的硬盘参数值：柱面数（磁道数、磁头数和每磁道扇区数，并要求输入 Image 文件的名称。程序在生成了 Image 文件之后，会显示一条用于 Bochs 配置文件中设置硬盘参数的配置信息。记下这条信息并编辑到配置文件中。下面是创建一个 256MB 硬盘 Image 文件的过程。

```
========================================================================
                                bximage
                  Disk Image Creation Tool for Bochs
        $Id: bximage.c,v 1.19 2003/08/01 01:20:00 cbothamy Exp $
========================================================================
Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd]

What kind of image should I create?
Please type flat, sparse or growing. [flat]

Enter the hard disk size in megabytes, between 1 and 32255
[10] 256

I will create a 'flat' hard disk image with
  cyl=520
  heads=16
  sectors per track=63
  total sectors=524160
  total size=255.94 megabytes

What should I name the image?
[c.img] hdc.img

Writing: [] Done.

I wrote 268369920 bytes to (null).

The following line should appear in your bochsrc:
  ata0-master: type=disk, path="hdc.img", mode=flat, cylinders=520, heads=16, spt=63

Press any key to continue
```

如果已经有了一个容量满足要求的硬盘 Image 文件,那么可以直接复制该文件就能产生另一个 Image 文件。然后可以按照自己的要求对该文件进行处理。对于创建软盘 Image 文件其过程与上述类似，只是还会提示你选择软盘种类的提示。同样，如果已经有其它软盘 Image 文件，那么采用直接复制方法即可。

14.3.2 在 Linux 系统下使用 dd 命令创建 Image 文件。

前面已经说明，刚创建的 Image 文件是一个内容全为 0 的空文件，只是其容量与要求的一致。因此我们可以首先计算出要求容量的 Image 文件的扇区数，然后使用 dd 命令来产生相应的 Image 文件。

例如我们想要建立柱面数是 520、磁头数是 16、每磁道扇区数是 63 的硬盘 Image 文件，其扇区总数为：520 * 16 * 63 = 524160，则命令为：

```
dd if=/dev/zero of=hdc.img bs=512 count=524160
```

对于 1.44MB 的软盘 Image 文件，其扇区数是 2880，因此命令为：

```
dd if=/dev/zero of=diska.img bs=512 count=2880
```

14.3.3 利用 WinImage 创建 DOS 格式的软盘 Image 文件

WinImage 是一个 DOS 格式 Image 文件访问和创建工具。双击 DOS 软盘 Image 文件的图标就可以浏览、删除或往里添加文件。除此之外，它还能用于浏览 CDROM 的 iso 文件。使用 WinImage 创建软盘 Image 时可以生成一个带有 DOS 格式的 Image 文件。方法如下：

a) 运行 WinImage。选择“Options->Settings”菜单，选择其中的 Image 设置页。设置 Compression 为“None”（也即把指示标拉到最左边）。
b) 创建 Image 文件。选择菜单 File->New，此时会弹出一个软盘格式选择框。请选择容量是 1.44MB 的格式。
c) 再选择引导扇区属性菜单项 Image->Boot Sector properties，单击对话框中的 MS-DOS 按钮。
d) 保存文件。

注意，在保存文件对话框中“保存类型”一定要选择“All files (*.*)”，否则创建的 Image 文件中会包含一些 WinImage 自己的信息，从而会造成 Image 文件在 Bochs 下不能正常使用。可以通过查看文件长度来确定新创建 Image 是否符合要求。标准 1.44MB 软盘的容量应该是 1474560 字节。如果新的 Image 文件长度大于该值，那么请严格按照所述方法重新制作或者使用 UltraEdit 等二进制编辑器删除多余的字节。删除操作的方法如下：

- 使用 UltraEdit 以二进制模式打开 Image 文件。根据磁盘映象文件第 511，512 字节是 55,AA 两个十六进制数，我们倒推 512 字节，删除这之前的所有字节。此时对于使用 MSDOS5.0 作为引导的磁盘来讲，文件头几个字节应该类似于“EB 3C 90 4D ...”。
- 然后下拉右边滚动条，移动到 img 文件末尾处。删除“...F6 F6 F6”后面的所有数据。通常来讲就是删除从 0x168000 开始的所有数据。操作完成时最后一行应该是完整的一行“F6 F6 F6...”。存盘退出即可使用该 Image 文件了。

14.4 访问磁盘映象文件中的信息

Bochs 使用磁盘映象文件来仿真被模拟系统中外部存储设备，被模拟操作系统中的所有文件均以软盘或硬盘设备中的格式保存在映象文件中。由此就带来了主机操作系统与 Bochs 中被模拟系统之间交换信息的问题。虽然 Bochs 系统能被配置成直接使用主机的软盘驱动器、CDROM 驱动器等物理设备来运行，但是利用这种信息交换方法比较烦琐。因此最好能够直接读写 Image 文件中的信息。如果需要往被模拟的操作系统中添加文件，就把文件存入 Image 文件中。如果要取出其中的文件就从 Image 文件中读出。但由于保存在 Image 文件中的信息不仅是按照相应的软盘或硬盘格式存放，而且还以一定的文件系统格式存放。因此访问 Image 文件中信息的程序必须能够识别其中的文件系统才能操作。对于本章应用来说，我们需要一些工具来识别 Image 文件中的 MINIX 和（或）DOS 文件系统格式。

总体来说，如果与模拟系统交换的文件长度比较小，我们可以采用软盘 Image 文件作为交换媒介。如果有大批量文件需要从模拟系统中取出或放入模拟系统，那么我们可以利用现有 Linux 系统来操作。下面就从这两个方面讨论可采用的几种方法。

利用磁盘映象读写工具访问软盘映象文件中的信息（小文件或分割的文件）
在 Linux 主环境中利用 loop 设备访问硬盘映象文件中的信息。（大批量信息交换）
利用 iso 格式文件进行信息交换（大批量信息交换）

14.4.1 使用 WinImage 工具软件

使用软盘 Image 文件，我们可以与模拟系统进行少量文件的交换。前提条件是被模拟系统支持对 DOS 格式软盘进行读写（例如通过 mtools 命令）。下面以实例来说明具体的操作方法。

在读写文件之前，首先需要根据前面描述的方法准备一个 1.44MB Image 文件（文件名假设是 diskb.img）。并修改 Linux 0.11 的 bochs.bxrc 配置文件，在 floppya 参数下增加以下一行信息：

```
floppyb: 1_44="diskb.img", status=inserted
```

也即给模拟系统增加第 2 个 1.44MB 软盘设备，并且该设备对应的 Image 文件名是 diskb.img。

如果想把 Linux 0.11 系统中的某个文件取出来，那么现在可以双击配置文件图标开始运行 Linux 0.11 系统。在进入 Linux 0.11 系统后，使用 DOS 软盘读写工具 mtools 把 hello.c 文件写到第 2 个软盘 Image 中：

```
[/usr/root]# mcopy hello.c b:
Copying HELLO.C
[/usr/root]# mdir b:
 Volume in drive B has no label
 Directory for B:/

HELLO    C            74    4-30-104   4:47p
       1 File(s)     1457152 bytes free
[/usr/root]# _
```

现在退出 Bochs 系统，并使用 WinImage 打开 diskb.img 文件，在 WinImage 的主窗口中会有一个 hello.c 文件存在。用鼠标选中该文件并拖到桌面上即完成了取文件的整个操作过程。 如果需要把某个文件输入到模拟系统中，那么操作步骤正好与上述相反。

14.4.2 利用现有 Linux 系统

现有 Linux 系统（例如 RedHat 9）能够访问多种文件系统，包括利用 loop 设备访问存储在文件中的文件系统。对于软盘 Image 文件，我们可以直接使用 mount 命令来加载 Image 中的文件系统进行读写访问。例如我们需要访问 rootimage-0.11 中的文件，那么只要执行以下命令。

```
[root@plinux images]# mount -t minix rootimage-0.11 /mnt -o loop
[root@plinux images]# cd /mnt
[root@plinux mnt]# ls
bin  dev  etc  root  tmp  usr
[root@plinux mnt]# _
```

其中 mount 命令的-t minix 选项指明所读文件系统类型是 MINIX，-o loop 选项说明通过 loop 设备来加载文件系统。若需要访问 DOS 格式软盘 Image 文件，只需把 mount 命令中的文件类型选项 minix 换成 msdos 即可。

如果想访问硬盘 Image 文件，那么操作过程与上述不同。由于软盘 Image 文件一般包含一个完整文件系统的映象，因此可以直接使用 mount 命令加载软盘 Image 中的文件系统，但是硬盘 Image 文件中通常含有分区信息，并且文件系统是在各个分区中建立的。也即我们可以把硬盘中的每个分区看成是一个完整的“大”软盘。

因此，为了访问一个硬盘 Image 文件某个分区中的信息，我们需要首先了解这个硬盘 Image 文件中分区信息，以确定需要访问的分区在 Image 文件中的起始偏移位置。关于硬盘 Image 文件中分区信息，我们可以在模拟系统运行时使用 fdisk 命令查看，也可以在这里查看。这里以下面软件包中包括的硬盘 Image 文件 hdc-0.11.img 为例来说明访问其中第 1 个分区中文件系统的方法。

http://oldlinux.org/Linux.old/bochs/linux-0.11-devel-040329.zip

这里需要用到 loop 设备设置与控制命令 losetup。该命令主要用于把一个普通文件或一个块设备与 loop 设备相关联，或用于释放一个 loop 设备、查询一个 loop 设备的状态。该命令的详细说明请参照在线手册页。

首先执行下面命令把 hdc-0.11.img 文件与 loop1 相关联，并利用 fdisk 命令查看其中的分区信息。

```
[root@plinux devel]# losetup /dev/loop1 hdc-0.11.img
[root@plinux devel]# fdisk /dev/loop1
Command (m for help): x                                    # 进入扩展功能菜单
Expert command (m for help): p                             # 显示分区表
Disk /dev/loop1: 16 heads, 63 sectors, 121 cylinders

Nr AF  Hd Sec  Cyl  Hd Sec  Cyl    Start     Size ID
 1 80   1   1    0  15  63  119        1   120959 81
 2 00   0   0    0   0   0    0        0        0 00
 3 00   0   0    0   0   0    0        0        0 00
 4 00   0   0    0   0   0    0        0        0 00
Expert command (m for help): q
[root@plinux devel]# _
```

从上面 fdisk 给出的分区信息可以看出，该 Image 文件仅含有 1 个分区。记下该分区的起始扇区号（也即分区表中 Start 一栏的内容）。如果你需要访问具有多个分区的硬盘 Image，那么你就需要记住相关分区的起始扇区号。

接下来，我们先使用 losetup 的-d 选项把 hdc-0.11.img 文件与 loop1 的关联解除，重新把它关联到 hdc-0.11.img 文件第 1 个分区的起始位置处。这需要使用 losetup 的-o 选项，该选项指明关联的起始字节偏移位置。由上面分区信息可知，这里第 1 个分区的起始偏移位置是 1 * 512 字节。在把第 1 个分区与 loop1 重新关联后，我们就可以使用 mount 命令来访问其中的文件了。

```
[root@plinux devel]# losetup -d /dev/loop1
[root@plinux devel]# losetup -o 512 /dev/loop1 hdc-0.11.img
[root@plinux devel]# mount -t minix /dev/loop1 /mnt
[root@plinux devel]# cd /mnt
[root@plinux mnt]# ls
bin  dev  etc  image  mnt  tmp  usr  var
[root@plinux mnt]# _
```

在对分区中文件系统访问结束后，最后请卸载和解除关联。

```
[root@plinux mnt]# cd
[root@plinux root]# umount /dev/loop1
[root@plinux root]# losetup -d /dev/loop1
[root@plinux root]# _
```

14.5 制作根文件系统

本节的目标是在硬盘上建立一个根文件系统。虽然在 oldlinux.org 上可以下载到已经制作好的软盘和硬盘根文件系统 Image 文件，但这里还是把制作过程详细描述一遍，以供大家学习参考。在制作过程中还可以参考 Linus 的文章：INSTALL-0.11。在制作根文件系统盘之前，我们首先下载 rootimage-0.11 和 bootimage-0.11 映象文件：

http://oldlinux.org/Linux.old/images/bootimage-0.11-20040305
http://oldlinux.org/Linux.old/images/rootimage-0.11-20040305

将这两个文件修改成便于记忆的 bootimage-0.11 和 rootimage-0.11，并专门建立一个名为 Linux-0.11 的子目录。在制作过程中，我们需要复制 rootimage-0.11 软盘中的一些执行程序并使用 bootimage-0.11 引导盘来启动模拟系统。因此在开始着手制作根文件系统之前，首先需要确认已经能够运行这两个软盘 Image 文件组成的最小 Linux 系统。

14.5.1 根文件系统和根文件设备

Linux 引导启动时，默认使用的文件系统是根文件系统。其中一般都包括以下一些子目录和文件：

- etc/ 目录主要含有一些系统配置文件；
- dev/ 含有设备特殊文件，用于使用文件操作语句操作设备；
- bin/ 存放系统执行程序。例如 sh、mkfs、fdisk 等；
- usr/ 存放库函数、手册和其它一些文件；
- usr/bin 存放用户常用的普通命令；
- var/ 用于存放系统运行时可变的数据或者是日志等信息。

存放文件系统的设备就是文件系统设备。比如，对于一般使用的 Windows2000 操作系统，硬盘 C 盘就是文件系统设备，而硬盘上按一定规则存放的文件就组成文件系统，Windows2000 有 NTFS 或 FAT32 等文件系统。而 Linux 0.11 内核所支持的文件系统是 MINIX 1.0 文件系统。

14.5.2 创建文件系统

对于上面创建的硬盘 Image 文件，在能使用之前还必须对其进行分区和创建文件系统。通常的做法是把需要处理的硬盘 Image 文件挂接到 Bochs 下已有的模拟系统中（例如上面提到的 SLS Linux），然后使用模拟系统中的命令对新的 Image 文件进行处理。下面假设你已经安装了 SLS Linux 模拟系统，并且该系统存放在名称为 SLS-Linux 的子目录中。我们利用它对上面创建的 256MB 硬盘 Image 文件 hdc.img 进行分区并创建 MINIX 文件系统。我们将在这个 Image 文件中创建 1 个分区，并且建立成 MINIX 文件系统。我们执行的步骤如下：

1. 在 SLS-Linux 同级目录下建立一个名称为 Linux-0.11 的子目录，把 hdc.img 文件移动到该目录下。
2. 进入 SLS-Linux 目录，编辑 SLS Linux 系统的 Bochs 配置文件 bochsrc.bxrc。在 ata0-master 一行下加入我们的硬盘 Image 文件的配置参数行：

   ata0-slave:type=disk, path=..\Linux-0.11\hdc.img, cylinders=520, heads=16, spt=63

3. 退出编辑器。双击 bochsrc.bxrc 的图标，运行 SLS Linux 模拟系统。在出现 Login 提示符时键入’root’

并按回车键。如果此时 Bochs 不能正常运行，一般是由于配置文件信息有误，请重新编辑该配置文件。

4. 利用 fdisk 命令在 hdc.img 文件中建立 1 个分区。下面是建立第 1 个分区的命令序列。建立另外 3 个分区的过程与此相仿。由于 SLS Linux 默认建立的分区类型是支持 MINIX2.0 文件系统的 81 类型（Linux/MINIX），因此需要使用 fdisk 的 t 命令把类型修改成 80（Old MINIX）类型。这里请注意，我们已经把 hdc.img 挂接成 SLS Linux 系统下的第 2 个硬盘。按照 Linux 0.11 对硬盘的命名规则，该硬盘整体的设备名应为/dev/hd5（参见下面表）。但是从 Linux 0.95 版开始硬盘的命名规则已经修改成目前使用的规则，因此在 SLS Linux 下第 2 个硬盘整体的设备名称是/dev/hdb。

```
[/]# fdisk /dev/hdb
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-520): 1
Last cylinder or +size or +sizeM or +sizeK (1-520): +63M

Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): L
 0  Empty           8  AIX             75  PC/IX           b8  BSDI swap
 1  DOS 12-bit FAT  9  AIX bootable    80  Old MINIX       c7  Syrinx
 2  XENIX root      a  OPUS            81  Linux/MINIX     db  CP/M
 3  XENIX user     40  Venix           82  Linux swap      e1  DOS access
 4  DOS 16-bit <32M 51  Novell?        83  Linux extfs     e3  DOS R/O
 5  Extended       52  Microport       93  Amoeba          f2  DOS secondary
 6  DOS 16-bit >=32 63  GNU HURD       94  Amoeba BBT      ff  BBT
 7  OS/2 HPFS      64  Novell          b7  BSDI fs
Hex code (type L to list codes): 80

Command (m for help): p
Disk /dev/hdb: 16 heads, 63 sectors, 520 cylinders
Units = cylinders of 1008 * 512 bytes
   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hdb1           1       1     129   65015+  80  Old MINIX

Command (m for help):w
The partition table has been altered.
Please reboot before doing anything else.
[/]#
```

表14.1 硬盘逻辑设备号

逻辑设备号	对应设备文件	说明
0x300	/dev/hd0	代表整个第 1 个硬盘
0x301	/dev/hd1	表示第 1 个硬盘的第 1 个分区
0x302	/dev/hd2	表示第 1 个硬盘的第 2 个分区
0x303	/dev/hd3	表示第 1 个硬盘的第 3 个分区

0x304	/dev/hd4	表示第 1 个硬盘的第 4 个分区
0x305	/dev/hd5	代表整个第 2 个硬盘
0x306	/dev/hd6	表示第 2 个硬盘的第 1 个分区
0x307	/dev/hd7	表示第 2 个硬盘的第 2 个分区
0x308	/dev/hd8	表示第 2 个硬盘的第 3 个分区
0x309	/dev/hd9	表示第 2 个硬盘的第 4 个分区

5. 请记住该分区中数据块数大小（这里是 65015），在创建文件系统时会使用到这个值。当分区建立好后，按照通常的做法需要重新启动一次系统，以让 SLS Linux 系统内核能正确识别这个新加的分区。
6. 再次进入 SLS Linux 模拟系统后，我们使用 mkfs 命令在刚建立的第 1 个分区上创建 MINIX 文件系统。命令与信息如下所示。这里创建了具有 64000 个数据块的分区（一个数据块为 1KB 字节）。

```
[/]# mkfs /dev/hdb1 64000
21333 inodes
64000 blocks
Firstdatazone=680 (680)
Zonesize=1024
Maxsize=268966912
[/]#
```

至此，我们完成了在 hdc.img 文件的第 1 个分区中创建文件系统的工作。当然，建立创建文件系统也可以在运行 Linux 0.11 软盘上的根文件系统时建立。的现在我们可以把这个分区中建立成一个根文件系统。

14.5.3 Linux-0.11 的 Bochs 配置文件

在 Bochs 中运行 Linux 0.11 模拟系统时，其配置文件 bochsrc.bxrc 中通常需要以下这些内容。

```
romimage: file=$BXSHARE\BIOS-bochs-latest, address=0xf0000
megs: 16
vgaromimage: $BXSHARE\VGABIOS-elpin-2.40
floppya: 1_44="bootimage-0.11", status=inserted
ata0-master: type=disk, path="hdc.img", mode=flat, cylinders=520, heads=16, spt=63
boot: a
log: bochsout.txt
panic: action=ask
#error: action=report
#info: action=report
#debug: action=ignore
ips: 1000000
mouse: enabled=0
```

我们可以把 SLS Linux 的 Bochs 配置文件 bochsrc.bxrc 复制到 Linux-0.11 目录中，然后修改成与上面相同的内容。需要特别注意 floppya、ata0-master 和 boot，这 3 个参数一定要与上面一致。

现在我们用鼠标双击这个配置文件。首先 Bochs 显示窗口应该出现以下画面。

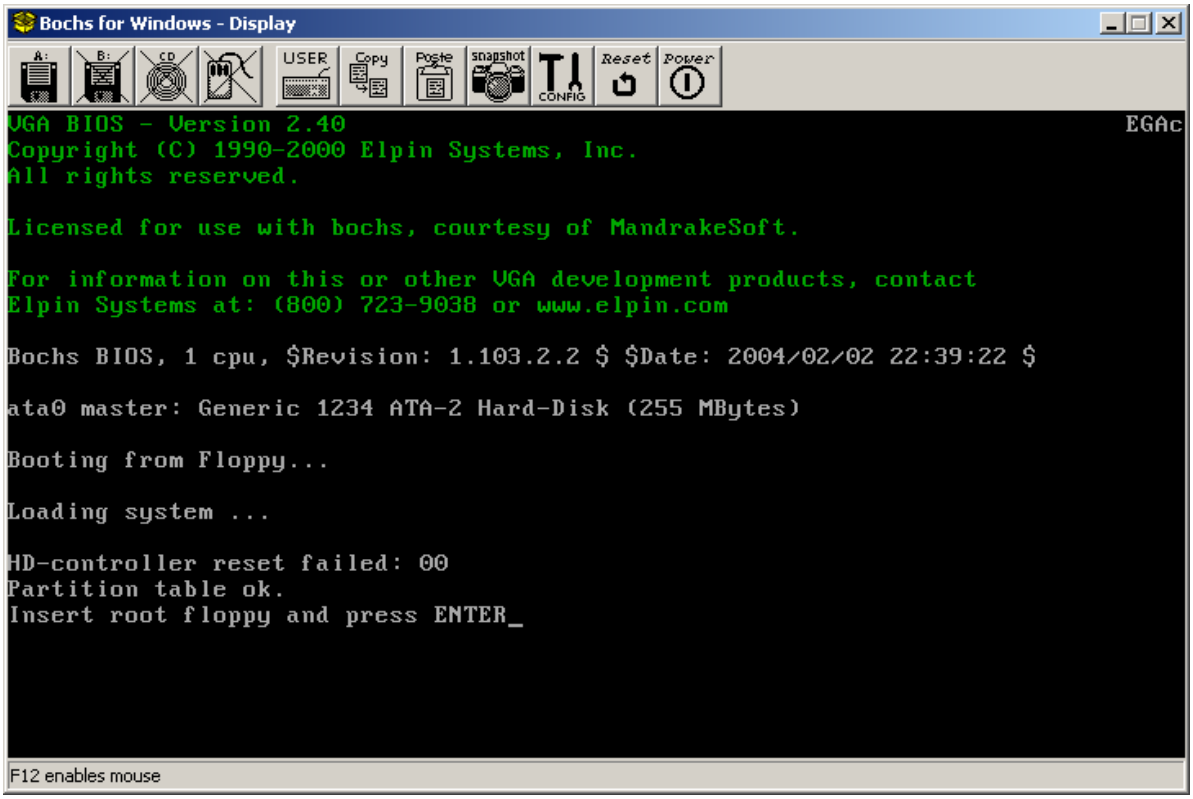

图14.2 Bochs 系统运行窗口

此时应该单击菜单条上的'CONFIG'图标进入 Bochs 设置窗口（需要用鼠标点击才能把该窗口提到最前面）。设置窗口显示的内容见下图所示。

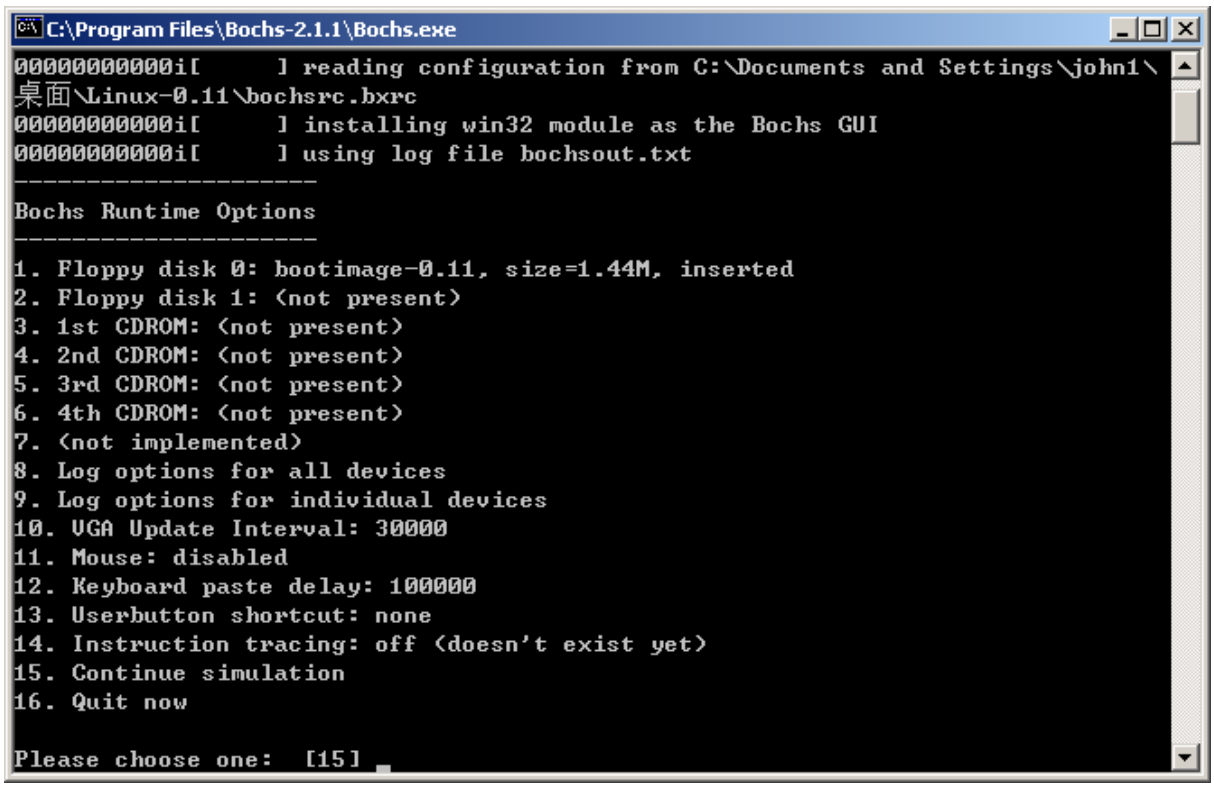

图14.3 Bochs 系统配置窗口

修改第 1 项的软盘设置，让其指向 rootimage-0.11 盘。然后连续按回车键，直到设置窗口最后一行信息显示'Continuing simulation'为止。此时再切换到 Bochs 运行窗口。单击回车键后就正式进入了 Linux 0.11 系统。见下图所示。

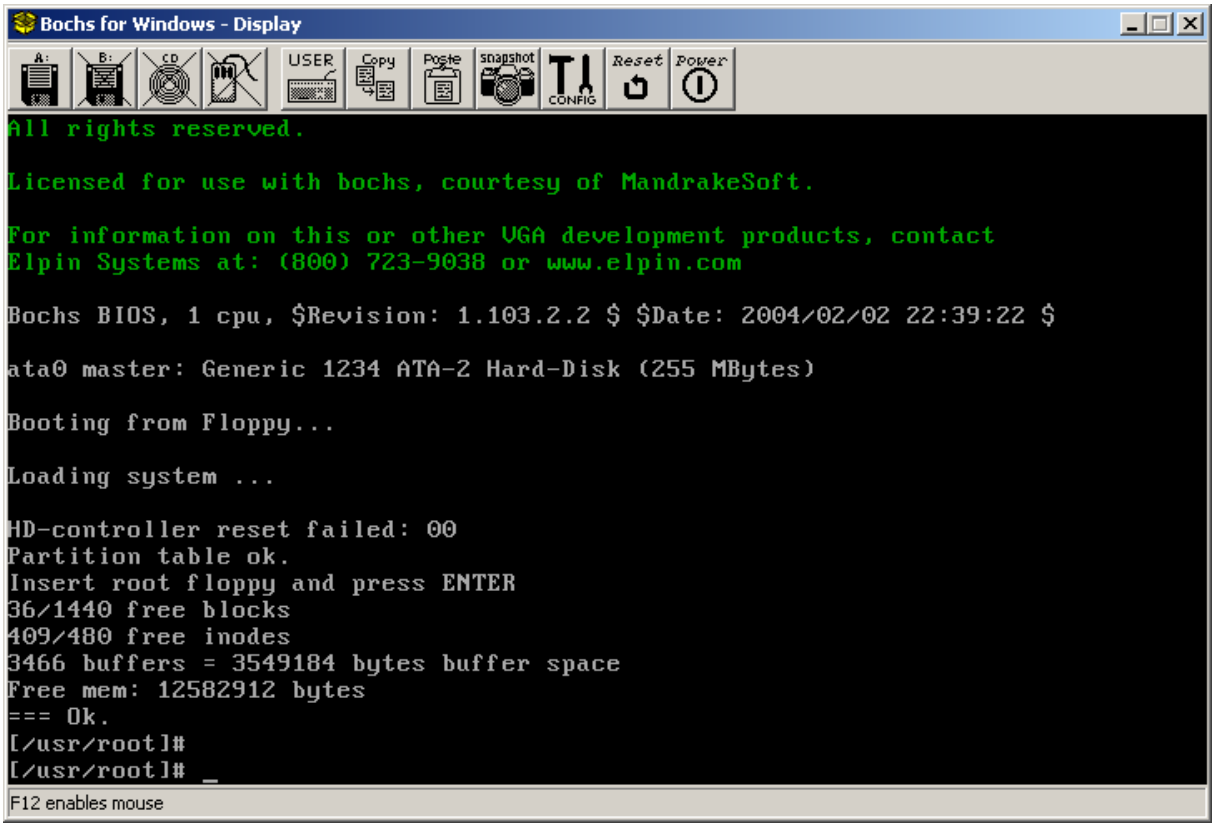

图14.4 Bochs 中运行的 Linux 0.11 系统

14.5.4 在 hdc.img 上建立根文件系统

由于软盘容量太小，若要让 Linux 0.11 系统真正能做点什么的话，就需要在硬盘（这里是指硬盘 Image 文件）上建立根文件系统。在前面我们已经建立一个 256MB 的硬盘 Image 文件 hdc.img，并且此时已经连接到了运行着的 Bochs 环境中，因此上图中出现一条有关硬盘的信息：

“ata0 master: Generic 1234 ATA-2 Hard-Disk (255 Mbytes)”

如果没有看到这条信息，说明你的 Linux 0.11 配置文件没有设置正确。请重新编辑 bochsrc.bxrc 文件，并重新运行 Bochs 系统，直到出现上述相同画面。

我们在前面已经在 hdc.img 第 1 个分区上建立了 MINIX 文件系统。若还没建好或者想再试一边的话，那么就请键入一下命令并按回车键：

```
[/usr/root]# mkfs /dev/hd1 64000
```

现在可以开始加载硬盘上的文件系统了。执行下列命令，把新的文件系统加载到/mnt 目录上。

```
[/usr/root]# cd /
[/]# mount /dev/hd1 /mnt
[/]#
```

在加载了硬盘分区上的文件系统之后，我们就可以把软盘上的根文件系统复制到硬盘上去了。请执行以下命令：

```
[/]# cd /mnt
[/mnt]# for i in bin dev etc usr tmp
> do
> cp +recursive +verbose /$i $i
done
```

此时软盘根文件系统上的所有文件就会被复制到硬盘上的文件系统中。在复制过程中会出现很多类似下面的信息。

```
/usr/bin/mv -> usr/bin/mv
/usr/bin/rm -> usr/bin/rm
/usr/bin/rmdir -> usr/bin/rmdir
/usr/bin/tail -> usr/bin/tail
/usr/bin/more -> usr/bin/more
/usr/local -> usr/local
/usr/root -> usr/root
/usr/root/.bash_history -> usr/root/.bash_history
/usr/root/a.out -> usr/root/a.out
/usr/root/hello.c -> usr/root/hello.c
/tmp -> tmp
[/mnt]# _
```

现在说明你已经在硬盘上建立好了一个基本的根文件系统。你可以在新文件系统中随处查看一下。然后卸载硬盘文件系统，并键入'logout'或'exit'退出 Linux 0.11 系统。此时会显示如下信息：

```
[/mnt]# cd /
[/]# umount /dev/hd1
[/]# logout

child 4 died with code 0000
[/usr/root]# _
```

14.5.5 使用硬盘 Image 上的根文件系统

一旦你在硬盘 Image 文件上建立好文件系统，就可以让 Linux 0.11 以它作为根文件系统启动。这通过修改引导盘 bootimage-0.11 文件的第 509、510 字节的内容就可以实现。请按照以下步骤来进行。

1. 首先复制 bootimage-0.11 和 bochsrc.bxrc 两个文件，产生 bootimage-0.11-hd 和 bochsrc-hd.bxrc 文件。

2. 编辑 bochsrc-hd.bxrc 配置文件。把其中的'floppya:'上的文件名修改成'bootimage-0.11-hd'，并存盘。
3. 用 UltraEdit 或任何其它可修改二进制文件的编辑器（winhex 等）编辑 bootimage-0.11-hd 二进制文件。修改第 509、510 字节（原值应该是 00、00）为 01、03，表示根文件系统设备在硬盘 Image 的第 1 个分区上。然后存盘退出。如果把文件系统安装在了别的分区上，那么需要修改前 1 个字节以对应到你的分区上。

```
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 01 03 55 AA ; ..............U?
```

现在可以双击 bochsrc-hd.bxrc 配置文件的图标，Bochs 系统应该会快速进入 Linux 0.11 系统并显示出以下图形来。

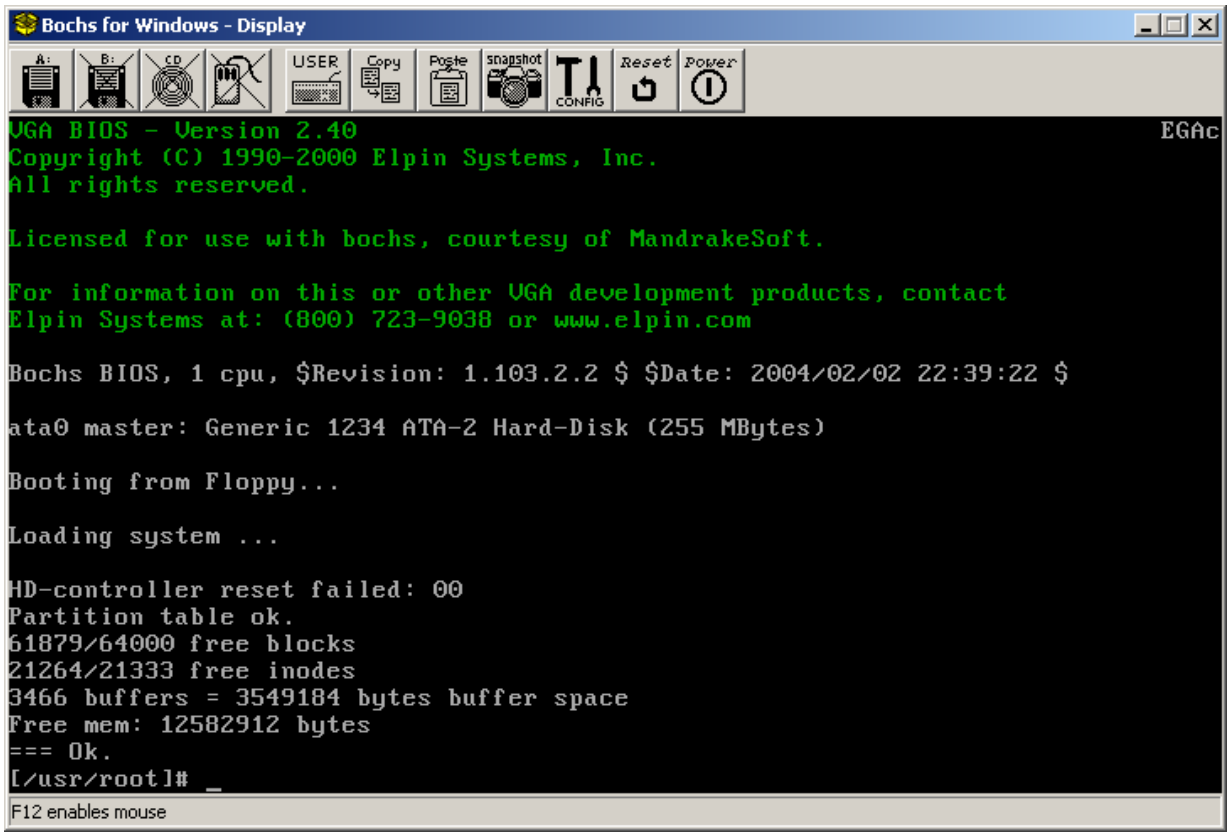

图14.5 使用硬盘 Image 文件上的文件系统

14.6 在 Linux 0.11 系统上编译 0.11 内核

目前作者已经重新组建了一个带有 gcc 1.40 编译环境的 Linux 0.11 系统软件包。该系统设置成在 Bochs 仿真系统下运行，并且已经配置好相应的 bochs 配置文件。该软件包可从下面地址得到。

http://oldlinux.org/Linux.old/bochs/linux-0.11-devel-040329.zip

该软件包中含有一个 README 文件，其中说明了软件包中所有文件的作用和使用方法。若你的系统中已经安装了 bochs 系统，那么只需双击配置文件 bochs-hd.bxrc 的图标即可运行硬盘 Image 文件作为根

文件系统的 Linux 0.11。在/usr/src/linux 目录下键入'make'命令即可编译 Linux 0.11 内核源代码，并生成引导启动映象文件 Image。若需要输出这个 Image 文件，可以首先备份 bootimage-0.11-hd 文件，然后使用下面命令就会把 bootimage-0.11-hd 替换成新的引导启动文件。直接重新启动 Bochs 即可使用该新编译生成的 bootimage-0.11-hd 来引导系统。

```
[/usr/src/linux]# make
[/usr/src/linux]# dd bs=8192 if=Image of=/dev/fd0
[/usr/src/linux]# _
```

也可以使用 mtools 命令把新生成的 Image 文件写到第 2 个软盘映象文件 diskb.img 中,然后使用工具软件 WinImage 将 diskb.img 中的文件取出。

```
[/usr/src/linux]# mcopy Image b:
Copying IMAGE
[/usr/src/linux]# mcopy System.map b:
Copying SYSTEM.MAP
[/usr/src/linux]# mdir b:
 Volume in drive B is B.
 Directory for B:/
GCCLIB-1 TAZ      934577    3-29-104   7:49p
IMAGE             121344    4-29-104  11:46p
SYSTEM   MAP       17162    4-29-104  11:47p
README               764    3-29-104   8:03p
       4 File(s)     382976 bytes free
[/usr/src/linux]# _
```

如果想把新的引导启动 Image 文件与软盘上的根文件系统 rootimage-0.11 一起使用,那么在编译之前首先编辑 Makefile 文件，使用#注释掉'ROOT_DEV='一行内容即可。

14.7 在 Redhat 9 系统下编译 Linux 0.11 内核

最初的 Linux 操作系统内核是在 Minix 1.5.10 操作系统的扩展版本 Minix-i386 上交叉编译开发的。Minix 1.5.10 该版本的操作系统是随 A.S. Tanenbaum 的《Minix 设计与实现》一书第 1 版一起由 Prentice Hall 发售的。该版本的 Minix 虽然可以运行在 80386 及其兼容微机上，但并没有利用 80386 的 32 位机制。为了能在该系统上进行 32 位操作系统的开发，Linus 使用了 Bruce Evans 的补丁程序将其升级为 MINIX-386，并把 GNU 的系列开发工具 gcc、gld、emacs、bash 等移植到 Minix-386 上。在这个平台上，Linus 进行交叉编译，开发出 Linux 0.01、0.03、0.11 等版本的内核。作者曾根据 Linux 邮件列表中的文章介绍，建立起了类似 Linus 当时的开发平台，并顺利地编译出 Linux 的早期版本内核（见 http://oldlinux.org 论坛中的介绍）。

但由于 Minix 1.5.10 早已过时，而且该开发平台的建立非常烦琐，因此这里只简单介绍一下如何修改 Linux 0.11 版内核源代码，使其能在目前常用的 RedHat 9 操作系统标准的编译环境下进行编译，生成可运行的启动映像文件 bootimage。读者可以在普通 PC 机上或 Bochs 等虚拟机软件中运行它。这里仅给出主要的修改方面，所有的修改之处可使用工具 diff 来比较修改后和未修改前的代码，找出其中的区别。假如，未修改过的代码在 linux 目录中，修改过的代码在 linux-mdf 中，则需要执行下面的命令：

```
diff -r linux linux-mdf > dif.out
```

其中文件 dif.out 中即包含代码中所有修改过的地方。已经修改好并能在 RedHat 9 下编译的 Linux 0.11 内核源代码可以从下面地址处下载：

http://oldlinux.org/Linux.old/kernel/linux-0.11-040327-rh9.tar.gz
http://oldlinux.org/Linux.old/kernel/linux-0.11-040327-rh9.diff.gz

用编译好的启动映像文件软盘启动时，屏幕上应该会显示以下信息：

```
Booting from Floppy...
Loading system ...

Insert root floppy and press ENTER
```

如果在显示出“Loding system...”后就没有反应了，这说明内核不能识别计算机中的硬盘控制器子系统。可以找一台老式的 PC 机再试试，或者使用 vmware、bochs 等虚拟机软件试验。在要求插入根文件系统盘时，如果直接按回车键，则会显示以下不能加载根文件系统的信息，并且死机。若要完整地运行 linux 0.11 操作系统，则还需要与之相配的根文件系统，可以到 oldlinux.org 网站上下载一个使用。

http://oldlinux.org/Linux.old/images/rootimage-0.11-for-orig

14.7.1 修改 makefile 文件

在 Linux 0.11 内核代码文件中，几乎每个子目录中都包括一个 makefile 文件，需要对它们都进行以下修改：

a. 将 gas =>as, gld=>ld。现在 gas 和 gld 已经直接改名称为 as 和 ld 了。
b. as(原 gas)已经不用-c 选项，所以要将 Makefile, 因此需要去掉其-c 编译选项。在内核主目录 Linux 下 makefile 文件中，是在 34 行上。
c. 去掉 gcc 的编译标志选项：-fcombine-regs、-mstring-insns 以及所有子目录中 Makefile 中的这两个选项。在 94 年的 gcc 手册中就已找不到-fcombine-regs 选项,而-mstring-insns 是 Linus 自己对 gcc 的修改增加的选项，所以你我的 gcc 中肯定不包括这个优化选项。
d. 在 gcc 的编译标志选项中，增加-m386 选项。这样在 RedHat 9 下编译出的内核映像文件中就不含有 80486 及以上 CPU 的指令，因此该内核就可以运行在 80386 机器上。

14.7.2 修改汇编程序中的注释

as86 编译程序不能识别 c 语言的注释语句，因此需要使用!注释掉 boot/bootsect.s 文件中的 C 注释语句。

14.7.3 内存位置对齐语句 align 值的修改

在 boot 目录下的三个汇编程序中，align 语句使用的方法目前已经改变。原来 align 后面带的数值是指对起内存位置的幂次值，而现在则需要直接给出对起的整数地址值。因此，原来的语句：

```
.align 3
```

需要修改成(2 的 3 次幂值 2^3=8)：

```
.align 8
```

14.7.4 修改嵌入宏汇编程序

由于对 as 的不断改进，目前其自动化程度越来越高，因此已经不需要人工指定一个变量需使用的 CPU 寄存器。因此内核代码中的__asm__("ax")需要全部去掉。例如 fs/bitmap.c 文件的第 20 行、26 行上，fs/namei.c 文件的第 65 行上等。

在嵌入汇编代码中，另外还需要去掉所有对寄存器内容无效的声明。例如 include/string.h 中第 84 行：

```
:"si","di","ax","cx");
```

需要修改成如下的样子：

```
: );
```

14.7.5 c 程序变量在汇编语句中的引用表示

在开发 Linux 0.11 时所用的汇编器，在引用 C 程序中的变量时需要在变量名前加一下划线字符'_'，而目前的 gcc 编译器可以直接识别使用这些汇编中引用的 c 变量，因此需要将汇编程序（包括嵌入汇编语句）中所有 c 变量之前的下划线去掉。例如 boot/head.s 程序中第 15 行语句：

```
.globl _idt,_gdt,_pg_dir,_tmp_floppy_area
```

需要改成：

```
.globl idt,gdt,pg_dir,tmp_floppy_area
```

第 31 行语句：

```
lss _stack_start,%esp
```

需要改成：

```
lss stack_start,%esp
```

14.7.6 保护模式下调试显示函数

在进入保护模式之前,可以用ROM BIOS中的int 0x10 调用在屏幕上显示信息，但进入了保护模式后，这些中断调用就不能使用了。为了能在保护模式运行环境中了解内核的内部数据结构和状态，我们可以使用下面这个数据显示函数check_data32()[1]。内核中虽然有printk()显示函数,但是它需要调用 tty_write(),在内核没有完全运转起来该函数是不能使用的。这个check_data32()函数可以在进入保护模式后,在屏幕上打印你感兴趣的东西。起用页功能与否，不影响效果，因为虚拟内存在 4M之内，正好使用了第一个页表目录项，而页表目录从物理地址 0 开始，再加上内核数据段基地址为 0，所以 4M范围内，虚拟内存与线性内存以及物理内存的地址相同。linus 当初可能也这样斟酌过的，觉得这样设置使用起来比较方便☺。

嵌入式汇编语句的使用方法参见 5.4.3.1 节中的说明。

```
/*
* 作用：在屏幕上用 16 进制显示一个 32 位整数。
* 参数：value -- 要显示的整数。
*       pos   -- 屏幕位置,以 16 个字符宽度为单位,例如为 2,即表示从左上角 32 字符宽度处开始显示。
* 返回：无。
* 如果要在汇编程序中用，要保证该函数被编译链接进了内核.gcc 汇编中的用法如下:
* pushl  pos      //pos 要用你实际的数据代替,例如 pushl $4
* pushl  value    //pos 和 value 可以是任何合法的寻址方式
* call   check_data32
```

[1] 该函数由oldlinux.org论坛上的朋友notrump提供。

```
*/
inline void check_data32(int value, int pos)
{
__asm__ __volatile__(                          // %0 - 含有欲显示的值 value；ebx - 屏幕位置。
"shl    $4, %%ebx\n\t"              // 将 pos 值乘 16，在加上 VGA 显示内存起始地址，
"addl   $0xb8000, %%ebx\n\t"        // ebx 中得到在屏幕左上角开始的显示字符位置。
"movl   $0xf0000000, %%eax\n\t"     // 设置 4 比特屏蔽码。
"movb   $28, %%cl\n"                // 设置初始右移比特数值。
"1:\n\t"
"movl   %0, %%edx\n\t"              // 取欲显示的值 value➔edx
"andl   %%eax, %%edx\n\t"           // 取 edx 中有 eax 指定的 4 个比特。
"shr    %%cl, %%edx\n\t"            // 右移 28 位，edx 中即为所取 4 比特的值。
"add    $0x30, %%dx\n\t"            // 将该值转换成 ASCII 码。
"cmp    $0x3a, %%dx\n\t"            // 若该 4 比特数值小于 10，则向前跳转到标号 2 处。
"jb2f\n\t"
"add    $0x07, %%dx\n"              // 否则再加上 7，将值转换成对应字符 A—F。
"2:\n\t"
"add    $0x0c00, %%dx\n\t"          // 设置显示属性。
"movw   %%dx, (%%ebx)\n\t"          // 将该值放到显示内存中。
"sub    $0x04, %%cl\n\t"            // 准备显示下一个 16 进制数，右移比特位数减 4。
"shr    $0x04, %%eax\n\t"           // 比特位屏蔽码右移 4 位。
"add    $0x02, %%ebx\n\t"           // 更新显示内存位置。
"cmpl   $0x0, %%eax\n\t"            // 屏蔽码值已经移出右端（已经显示完 8 个 16 进制数）？
"jnz1b\n"                           // 还有数值需要显示，则向后跳转到标号 1 处。
::"m"(value), "b"(pos));
}
```

14.8 利用 bochs 调试内核

（利用 System.map 文件中的信息）