

Alpha Edition



杨文博 著

2008年2月21日 16:46 Revision 1

《使用开源软件-自己动手写操作系统》官方网站:

http://share.solrex.cn/WriteOS/

您可以在本书官方网站下载到本书的最新版本和附带的全部程序源代码。

由于此版本非最终发布版,如果您对本书感兴趣,请关注作者在博客http://blog.solrex.cn上发布的更新公告。

写在前面的话

本书起源于中国电子工业出版社出版的一本书:《自己动手写操作系统》(于渊著)。我对《自己 动手写操作系统》这本书中使用商业软件做为演示平台比较惊讶,因为不是每个人都买得起正版软件 的,尤其是穷学生。我想《自》所面向的主要受众也应该是学生,那么一本介绍只有商业软件才能实现 的编程技巧的书将会逼着穷学生去使用盗版,这是非常罪恶的行为 ⑤。

由于本人是一个 Linux 用户,一个开源软件的拥护者,所以就试着使用开源软件实现这本书中的所有 demo,并在自己的博客上进行推广。后来我觉得,为什么我不能自己写本书呢?这样我就能插入漂亮的插图,写更详尽的介绍而不怕篇幅过长,更容易让读者接受也更容易传播,所以我就开始写这本《使用开源软件自己动手写操作系统》。

定下写一本书的目标毕竟不像写一篇博客,我将尽量详尽的介绍我使用的方法和过程,以图能让不同技术背景的读者都能通畅地完成阅读。但是自己写并且排版一本书不是很轻松的事情,需要耗费大量时间,所以我只能抽空一点一点的将这本书堆砌起来,这也是您之所以在本页上方看到本书版本号的原因 ②。

本书的最终目标是成为一本大学"计算机操作系统"课程的参考工具书,为学生提供一个 step by step 的引导去实现一个操作系统。这不是一个容易实现的目标,因为我本人现在并不自信有那个实力了 解操作系统的方方面面。但是我想,立志百里行九十总好过于踯躅不前。

《自己动手写操作系统》一书开了个好头,所以在前面部分,我将主要讨论使用开源软件实现 《自》的 demo。如果您有《自》这本书,参考阅读效果会更好,不过我将尽我所能在本书中给出清楚 的讲解,尽量使您免于去参考《自》一书。

出于开放性和易编辑性考虑,本书采用 IATEX 排版,在成书前期由于专注于版面,代码比较杂乱,可读性不强,暂不开放本书 TEX 源代码下载。

如果您在阅读过程中有什么问题,发现书中的错误,或者好的建议,欢迎您使用我留下的联系方式与我联系,本人将非常感谢。

杨文博

电子邮件: solrex@gmail.com 个人主页: http://solrex.cn 个人博客: http://blog.solrex.cn 2008年1月9日

版权声明

本文遵从署名-非商业性使用-相同方式共享2.5 中国大陆创作共享协议。

您可以自由:

- 复制、发行、展览、表演、放映、广播或通过信息网络传播本作品
- 创作演绎作品

惟须遵守下列条件:

- •署名.您必须按照作者或者许可人指定的方式对作品进行署名。
- 非商业性使用. 您不得将本作品用于商业目的。
- 相同方式共享. 如果您改变、转换本作品或者以本作品为基础进行创作,您只能采用与本协议相同 的许可协议发布基于本作品的演绎作品。
- 对任何再使用或者发行,您都必须向他人清楚地展示本作品使用的许可协议条款。
- 如果得到著作权人的许可,您可以不受任何这些条件的限制。
- Nothing in this license impairs or restricts the author's moral rights.

您的合理使用以及其他权利不受上述规定的影响。 这是一份普通人可以理解的法律文本(许可协议全文)的概要。

更新历史

Rev. 1 确定书本排版样式,添加第一章,第二章。

Rev. 2

目录

写在前面的话

序言

vii

i

第一章	计算机	启动	1
1.1	计算机周	自动过程	1
1.2	磁盘抽筆	象物理结构	2
	1.2.1	硬盘	3
	1.2.2	软盘	4
	1.2.3	启动扇区	5
1.3	使用虚打	以机	5
	1.3.1	VirtualBox	6
	1.3.2	Bochs	14
1.4	使用软垫	盐镜像	14
	1.4.1	制作软盘镜像	14
	1 / 9	田软盘镜像启动虚拟机	1/
	1.4.2)		14
	1.4.2)		14
第二章	1.4.2 最小的	"操作系统"	19
第二章 2.1	L.4.2 , 最小的 Hello O	"操作系统" S world!	1 9
第二章 2.1	最小的 Hello O 2.1.1	"操作系统" S world!	14 19 19 20
第二章 2.1	最小的 Hello O 2.1.1 〕 2.1.2 ,	"操作系统" S world!	19 19 20 20
第二章 2.1	最小的 Hello O 2.1.1 〕 2.1.2 , 2.1.3 ,	"操作系统" S world!	19 19 20 20 22
第二章 2.1	最小的 Hello O 2.1.1 〕 2.1.2 , 2.1.3 , 2.1.4 ,	"操作系统" S world!	14 19 20 20 22 24
第二章 2.1 2.2	最小的 Hello O 2.1.1 〕 2.1.2 , 2.1.3 , 2.1.4 , FAT 文	"操作系统" S world! Intel 汇编转化为 AT&T(GAS) 汇编 用连接脚本控制地址空间 用直接脚本控制地址空间 用虚拟机加载执行 boot.img 件系统	14 19 20 20 22 24 25
第二章 2.1 2.2	最小的 Hello O 2.1.1 〕 2.1.2 , 2.1.3 , 2.1.4 , FAT 文 ⁴ 2.2.1 〕	"操作系统" S world! Intel 汇编转化为 AT&T(GAS) 汇编 用连接脚本控制地址空间 用直接脚本控制地址空间 用 Makefile 编译连接 明虚拟机加载执行 boot.img 件系统 FAT12 文件系统	14 19 20 20 22 24 25 25
第二章 2.1 2.2	最小的 Hello O 2.1.1 〕 2.1.2 , 2.1.3 , 2.1.4 , FAT 文 2.2.1 〕 2.2.2 ,	"操作系统" S world! Intel 汇编转化为 AT&T(GAS) 汇编 用连接脚本控制地址空间 用基拟机加载执行 boot.img 件系统 FAT12 文件系统 启动扇区与 BPB	14 19 20 20 22 24 25 25 25 26
第二章 2.1 2.2	1.4.2 , 最小的 Hello O 2.1.1 1 2.1.2 , 2.1.3 , 2.1.4 , FAT 文/ 2.2.1 2.2.2 , 2.2.3 1	"操作系统" S world! Intel 汇编转化为 AT&T(GAS) 汇编 用连接脚本控制地址空间 用基拟机加载执行 boot.img 件系统 FAT12 文件系统 启动扇区与 BPB FAT12 数据结构	19 19 20 20 22 24 25 25 26 28

٠			
а	а	٤	2
		١	٢

2.3	让启动	助扇区加载引导文件	30
	2.3.1	一个最简单的 loader	30
	2.3.2	读取软盘扇区的 BIOS 13h 号中断	30
	2.3.3	搜索 loader.bin	32
	2.3.4	加载 loader 入内存	35
	2.3.5	向 loader 转交控制权	37
	2.3.6	生成镜像并测试	37
2.4	本章全	≥部源代码	40

插图

1.1	硬盘	3
1.2	硬盘的抽象物理结构	3
1.3	软盘	4
1.4	启动扇区加载示意图	5
1.5	VirtualBox 个人使用协议	6
1.6	同意 VirtualBox 个人使用协议	7
1.7	VirtualBox 用户注册对话框	7
1.8	VirtualBox 主界面	8
1.9	新建一个虚拟机	8
1.10	设置虚拟机名字和操作系统类型	9
1.11	设置虚拟机内存容量	9
1.12	设置虚拟机硬盘镜像	10
1.13	新建一块虚拟硬盘	10
1.14	设置虚拟硬盘类型	11
1.15	设置虚拟硬盘镜像名字和容量	11
1.16	虚拟硬盘信息	12
1.17	使用新建的虚拟硬盘	12
1.18	虚拟机信息	13
1.19	回到 VirtualBox 主界面	13
1.20	虚拟机设置界面	14
1.21	虚拟机软盘设置	15
1.22	选择软盘镜像	15
1.23	选择启动软盘镜像	16
1.24	确认启动镜像软盘文件信息	16
1.25	查看虚拟机设置信息	17
1.26	自动键盘捕获警告信息	17
1.27	虚拟机运行时	18

《自》第一个实例代码 boot.asm	20
boot.S(chapter2/1/boot.S)	20
boot.S 的连接脚本 (chapter2/1/solrex_x86.ld)	21
《自》代码 1-2 (chapter2/1/boot.asm)	21
boot.S 的 Makefile(chapter2/1/Makefile)	22
使用 hexedit 打开 boot.img	23
使用 kde 图形界面工具 khexedit 打开 boot.img	24
选择启动软盘镜像 boot.img	24
虚拟机启动后打印出红色的"Hello OS world!"	25
生成启动扇区头的汇编代码(节自chapter2/2/boot.S)	27
FAT 文件系统存储结构图	28
一个最简单的 loader(chapter2/2/loader.S)	30
一个最简单的 loader(chapter2/2/solrex_x86_dos.ld)	30
读取软盘扇区的函数(节自chapter2/2/boot.S)	32
搜索 loader.bin 的代码片段(节自chapter2/2/boot.S)	34
搜索 loader.bin 使用的变量定义(节自chapter2/2/boot.S)	34
打印字符串函数 DispStr (节自chapter2/2/boot.S)	35
寻找 FAT 项的函数 GetFATEntry (节自chapter2/2/boot.S)	36
加载 loader.bin 的代码(节自chapter2/2/boot.S)	37
跳转到 loader 执行(节自chapter2/2/boot.S)	37
用 Makefile 编译(节自chapter2/2/boot.S)	38
拷贝 LOADER.BIN 入 boot.img(节自chapter2/2/boot.S)	39
没有装入 LOADER.BIN 的软盘启动	39
装入了 LOADER.BIN 以后再启动	40
chapter2/1/boot.S	41
chapter2/1/solrex_x86.ld	41
chapter2/1/Makefile	42
chapter2/2/boot.S	47
chapter2/2/loader.S	48
chapter2/2/solrex_x86_boot.ld	48
chapter2/2/solrex_x86_dos.ld	49
chapter2/2/Makefile	50
	《自》第一个实例代码 boot.asm boot.S(chapter2/1/boot.S). boot.S 的连接脚本(chapter2/1/solrex_x86.ld). 《自》代码 1-2 (chapter2/1/boot.asm). boot.S 的 Makefile(chapter2/1/Makefile). 使用 hexedit 打开 boot.img 使用 kake 图形界面工具 khexedit 打开 boot.img 选择启动软盘镜像 boot.img 虚拟机启动后打印出红色的"Hello OS world!" 生成启动和国区头的汇编代码(节自chapter2/2/boot.S) FAT 文件系统存储结构图 一个最简单的 loader(chapter2/2/loader.S) 一个最简单的 loader(chapter2/2/loader.S) 一个最简单的 loader(chapter2/2/loader.S) 一个最简单的 loader(chapter2/2/loader.S) 建家 loader.bin 的代码方段(节自chapter2/2/boot.S) 搜索 loader.bin 的K码后载定义(节自chapter2/2/boot.S) 搜索 loader.bin 使用的变量定义(节自chapter2/2/boot.S) 搜索 loader.bin 使用的变量定义(节自chapter2/2/boot.S) 引和 和kefile 编译(节自chapter2/2/boot.S) 动 n载 loader.bin 的代码(节自chapter2/2/boot.S) 费材 EAT 项的函数 GetFATEntry (节自chapter2/2/boot.S) 费制 loader.bin 的代码(节自chapter2/2/boot.S) 费制 loader.bin 的代码(节自chapter2/2/boot.S) 建特到 loader.bin 的代码(节自chapter2/2/boot.S) 发行 LOADER.BIN \boton.img(节自chapter2/2/boot.S) 发力 TOADER.BIN \boton.img(节自chapter2/2/boot.S) 没有 未入 T LOADER.BIN \boton.img(节自chapter2/2/boot.S) chapter2/1/solrex.x86.ld

序言

这里应该是各个章节的摘要和版式简介,不过因为写摘要向来是件让人心烦的事情,所以我准备把 它放在最后写 ©。

₹\$ € viii

CHAPTER 1

计算机启动

任何一个计算机软件都是由一系列的可执行文件组成的,可执行文件的内容是可以被机器识别的二 进制指令和数据。一般可执行文件的运行是在操作系统的照看下加载进内存并运行的,由操作系统给它 分配资源和处理器时间,并确定它的执行方式。操作系统也是由可执行文件组成的,但是操作系统的启 动方式和一般应用软件是不同的,这也就是它叫做"操作系统"的原因 ⑤。

没有操作系统的机器,一般情况下被我们称为"裸机",意思就是只有硬件,什么都干不了。但是 一个机器怎么知道自己是不是裸机呢?它总要有方式去判断机器上安装没有安装操作系统吧。下面我们 就简单介绍一下计算机启动的过程:

1.1 计算机启动过程

计算机启动过程一般是指计算机从点亮到加载操作系统的一个过程。对于 IBM 兼容机(个人电脑)来讲,这个过程大致是这样的:

- 加电 电源开关被按下时,机器就开始供电,主板的控制芯片组会向 CPU (Central Processing Unit,中央处理器)发出并保持一个 RESET (重置)信号,让 CPU 恢复到初始状态。当芯片组 检测到电源已经开始稳定供电时就会撤去 RESET 信号(松开台式机的重启键是一样的效果), 这时 CPU 就从 0xffff0 处开始执行指令。这个地址在系统 BIOS (Basic Input/Output System, 基本输入输出系统)的地址范围内,大部分系统 BIOS 厂商放在这里的都只是一条跳转指令,跳 到系统 BIOS 真正的启动代码处。
- 2. 自检系统 BIOS 的启动代码首先要做的事情就是进行 POST (Power-On Self Test,加电后自检),POST 的主要任务是检测系统中一些关键设备是否存在和能否正常工作,例如内存和显卡等。由于 POST 是最早进行的检测过程,此时显卡还没有初始化,如果系统 BIOS 在 POST 的过程中发现了一些致命错误,例如没有找到内存或者内存有问题(此时只会检查 640K 常规内存),那么系统 BIOS 就会直接控制喇叭发声来报告错误,声音的长短和次数代表了错误的类型。
- 3. 初始化设备 接下来系统 BIOS 将查找显卡的BIOS,存放显卡 BIOS 的 ROM 芯片的起始地址通 常设在 0xC0000 处,系统 BIOS 在这个地方找到显卡 BIOS 之后就调用它的初始化代码,由显卡

BIOS 来初始化显卡,此时多数显卡都会在屏幕上显示出一些初始化信息,介绍生产厂商、图形芯片类型等内容。系统 BIOS 接着会查找其它设备的 BIOS 程序,找到之后同样要调用这些 BIOS 内部的初始化代码来初始化相关的设备。

- 4. 测试设备 查找完所有其它设备的 BIOS 之后,系统 BIOS 将显示出它自己的启动画面,其中包括 有系统 BIOS 的类型、序列号和版本号等内容。接着系统 BIOS 将检测和显示 CPU 的类型和工 作频率,然后开始测试所有的 RAM (Random Access Memory,随机访问存储器),并同时在 屏幕上显示内存测试的进度。内存测试通过之后,系统 BIOS 将开始检测系统中安装的一些标准 硬件设备,包括硬盘、光驱、串口、并口、软驱等,另外绝大多数较新版本的系统 BIOS 在这一 过程中还要自动检测和设置内存的定时参数、硬盘参数和访问模式等。标准设备检测完毕后, 系统 BIOS 内部的支持即插即用的代码将开始检测和配置系统中安装的即插即用设备,每找到一 个设备之后,系统 BIOS 都会在屏幕上显示出设备的名称和型号等信息,同时为该设备分配中断 (INT)、DMA (Direct Memory Access,直接存储器存取)通道和 I/O (Input/Output,输入 输出)端口等资源。
- 5. 更新 SECD 所有硬件都检测配置完毕后,多数系统 BIOS 会重新清屏并在屏幕上方显示出一个 表格,其中概略地列出了系统中安装的各种标准硬件设备,以及它们使用的资源和一些相关工作 参数。接下来系统 BIOS 将更新 ESCD (Extended System Configuration Data, 扩展系统配置数 据)。 ESCD 是系统 BIOS 用来与操作系统交换硬件配置信息的一种手段,这些数据被存放在 CMOS (Complementary Metal Oxide Semiconductor,互补金属氧化物半导体)之中。
- 6. 启动操作系统 ESCD 更新完毕后,系统 BIOS 的启动代码将进行它的最后一项工作,即根据用户指定的启动顺序从软盘、硬盘或光驱启动操作系统。以 Windows XP 为例,系统 BIOS 将启动盘(一般是主硬盘)的第一个扇区(Boot Sector,引导扇区)读入到内存的 0x7c00 处,并检查 0x7dfe 地址的内存,如果其内容是 0xaa55,跳转到 0x7c00 处执行 MBR (Master Boot Record,主引导记录),MBR 接着从分区表(Partition Table)中找到第一个活动分区 (Active Partition,一般是C盘分区),然后按照类似方式读取并执行这个活动分区的引导扇区 (Partition Boot Sector),而引导扇区将负责读取并执行 NTLDR (NT LoaDeR, Windows NT 的加载程序),然后主动权就移交给了 Windows。

从以上介绍中我们可以看到,在第6步之前,电脑的启动过程完全依仗于系统 BIOS,这个程序一般是厂商写就固化在硬盘上的。我们所需要做的,就是第6步之后的内容,即:

如何写一个操作系统并把它加载到内存?

1.2 磁盘抽象物理结构

由于操作系统的启动涉及到硬件地址写入和磁盘文件寻找,为了更好理解内存地址和文件存储的相 关知识,我们先来了解一下磁盘的结构。

 $\mathbf{2}$

1.2.1 硬盘



Figure 1.1: 硬盘

图 1.1 所示就是硬盘(如非特指,我们这里的"硬盘"一般指代磁介质非固态硬盘)的外观图。其中左边是硬盘盒拆开后盘片、磁头和内部机械结构的透视图,右边是普通台式机硬盘的外观图。现在的硬盘容量较以前已经有大幅度增加,一般笔记本电脑硬盘容量已经在 120G 以上,台式机硬盘容量一般也达到了 160G 大小。一般情况下,硬盘都是由坚硬金属材料(或者玻璃等)制成的涂以磁性介质的盘片构成的,一般有层叠的多片,每个盘片都有两个面,两面都可以记录信息。



Figure 1.2: 硬盘的抽象物理结构

图 1.2 为硬盘的抽象物理结构,需要注意的是这并不是硬盘真正的物理构造,所以这里我们称其为"抽象"物理结构。因此我们下面讨论的也不是真正的硬盘技术实现,仅仅就硬盘(以及软盘等类似磁介质存储器)存储结构以程序员易于理解的角度进行简单的介绍。

如图 1.2 所示,硬盘是由很多盘片组成的,那些上下有分割的圆盘就表示一个个盘片。每个盘片被分成许多扇形的区域,每个区域叫一个扇区,通常每个扇区存储 512 字节(FAT 文件格式),盘片表面上以盘片中心为圆心,不同半径的同心圆称为磁道。硬盘中,不同盘片相同半径的磁道所组成的圆柱称为柱面。磁道与柱面都是表示不同半径的圆,在许多场合,磁道和柱面可以互换使用。每个磁盘有两

个面,每个面都有一个磁头,习惯用磁头号来区分。扇区,磁道(或柱面)和磁头数构成了硬盘结构的 基本参数,使用这些参数可以得到硬盘的容量,其计算公式为:

存储容量 = 磁头数 × 磁道(柱面)数 × 每磁道扇区数 × 每扇区字节数



1.2.2 软盘

由于我们在本书中主要使用软盘作为系统启动盘,所以下面对应于硬盘介绍一下软盘的相关知识。



Figure 1.3: 软盘

现在通常能看到的软盘主要是 3.5 英寸软盘, 3.5 英寸指的是其内部磁介质盘片的直径。从存储结构上来讲,软盘与硬盘的主要不同就是软盘只有一个盘片且其存储密度较低。

由于软盘只有一个盘片,两个面,所以3.5英寸软盘的容量可以根据上一小节的公式算出:

2(磁头) × 80(磁道) × 18(扇区) × 512 bytes(扇区的大小) = 2880 x 512 bytes = 1440 KB = 1.44MB 在这里需要引起我们特别注意的就是第0号磁头(面),第0号磁道的第0号扇区,这里是一切的 开始。

1.2.3 启动扇区

软盘是没有所谓的 MBR 的,因为软盘容量较小,没有所谓的分区,一张软盘就显示为一个逻辑 磁盘。当我们使用软盘启动电脑的时候,系统从软盘中首先读取的就是第一个扇区,即前面所说的第 0 面,第 0 号磁道的第 0 号扇区,如果这个扇区的最后两个字节是 0xaa55,这里就简单叫做启动扇区 (Boot Sector)。所以我们首先要做的就是:在启动扇区的开始填入需要被执行的机器指令;在启动 扇区的最后两个字节中填入 0xaa55,这样这张软盘就成为了一张可启动盘。

 启动扇区最后两个字节的内容为 0xaa55,这种说法是正确的−−当且仅当表 2.1 中 的 BPB_BytesPerSec (每扇区字节数)的值为 512。如果 BPB_BytesPerSec 的值大于 512, 0xaa55 的位置不会变化,但已经不是启动扇区最后两个字节了。

整个过程如图 1.4 所示:



Figure 1.4: 启动扇区加载示意图

需要注意的是,软盘的启动扇区并不像一个文件一样,可以直接读取,写入启动扇区的过程是需要 一些技巧的,下面我们将讨论如何去实现。

1.3 使用虚拟机

在实现一个简单的操作系统时,我们是不可能拿一台真正的机器做实验的,一是很少有人有这个条件,还有就是那样做比较麻烦。所以我们使用虚拟机来模拟一台真实的电脑,这样我们就能直接用虚拟 机加载软盘镜像来启动了,而制作软盘镜像显然要比写一张真实的软盘简单许多。

在 Linux 下有很多虚拟机软件,我们选择 VirtualBox 和 Bochs 作为主要的实现平台,我们用 VirtualBox 做 demo,而 Bochs 主要用作调试。下面给出一些虚拟机设置的指导,其实用哪种虚拟机都 没有关系,我们需要的只是虚拟机支持加载软盘镜像并能从软盘启动。

1.3.1 VirtualBox

VirtualBox 是遵从 GPL 协议的开源软件,它的官方网站是 http://www.virtualbox.org。 VirtualBox 的官方网站上提供针对很多 Linux 系统平台的二进制安装包,比如针对 Red Hat 系列 (Fedora, RHEL) 各种版本的 RPM 安装包,针对 Debian 系(Debian, Ubuntu) 各种版本的 DEB 安 装包,其中 Ubuntu Linux 可以更方便地从 Ubuntu 软件仓库中直接下载安装: sudo apt-get install virtualbox。

安装好 VirtualBox 后,需要使用 sudo adduser 'whoami' vboxusers (某些系统中的添加用户命 令可能是 useradd) 将自己添加到 VirtualBox 的用户组 vboxusers 中去;当然,也可以使用 GNOME 或者 KDE 的图形界面用户和组的管理工具来添加组用户,也可以直接编辑 /etc/group 文件,将自己的 用户名添加到 vboxusers 对应行的最后,例如 vboxusers:x:501:solrex,部分 Linux 可能需要注销后 重新登录当前用户。

我们下面使用 CentOS 上安装的 VirtualBox 演示如何用它建立一个虚拟机。

第一次启动 VirtualBox, 会首先弹出一个 VirtualBox 个人使用协议 PUEL 的对话框(某些版本的 Linux 可能不会弹出):

22	VirtualBox License	×
Virtua	Box Personal Use and Evaluation License (PUEL)	-
Versio	n 1.4, October 2, 2007	
innote use the do not becaus	k GmbH, Werkstrasse 24, 71384 Weinstadt, Germany grants you the right to e software product as defined in § 1 according to the following provisions. If you agree to all conditions set forth by this license, you may not use the product, se only innotek as the product's owner can give you permission to use it.	
§1 Su softwa each w with a execut system and do	bject of license. ``Product'', as referred to in this License, shall be the binary re package ``VirtualBox'', which allows for creating multiple virtual computers, with different operating systems (``Guest Computers''), on a physical computer specific operating system (``Host Computer''), to allow for installing and ting these Guest Computers simultaneously. The Product consists of table files in machine code for the Windows 2000/XP and Linux operating ms as well as other data files as required by the executable files at run-time iccumentation in electronic form.	
§ 2 Gra the Pro Evalua	ant of license. (1)innotek grants you a personal right to install and execute aduct on a Host Computer for Personal Use or Educational Use or for ation. ``Personal Use'' requires that you use the product on the same Host	*
	I <u>Ag</u> ree I <u>D</u> isagre	е

Figure 1.5: VirtualBox 个人使用协议

Chapter 1. 计算机启动

阅 误 元 协 仅 后 ,	将下拉条拉到冣骶可以激活冣下方的问息按钮,只击之: VirtualBox License	×
viii Aff Pri S S Wa Pri bu pa wi re Pri ca	cense only for material causes. In particular, such a material cause can be a colation of the usage terms or a breach of other essential duties from this contract. ter termination, you are required to delete and destroy all remaining copies of the oduct. This includes, but is not limited to, installed copies and backups. 5 No warranties. Since you have not paid for the use of the Product, there is no arranty for it, to the extent permitted by applicable law. innotek provides the oduct ``as is'' without warranty of any kind, either expressed or implied, including, it not limited to, the implied warranties of merchantability and fitness for a pricular purpose. The entire risk as to the quality and performance of the Product is th you. Should it prove defective, you assume the cost of all necessary servicing, pair, or correction. In addition, innotek shall be allowed to provide updates to the oduct in urgent cases. You are then obliged to install such updates. Such an urgent is includes, but is not limited to, a claim of rights to the Product by a third party.	•
5 d ag sh re pe Ge	5 Miscellaneous. There are no license terms beyond the written ones in this preement. Amendments of, additions to and the joint revocation of this agreement hall require the written form. The same shall apply to the preceding written form quirement. Standard business conditions of the parties shall not apply. Place of prormance and legal venue shall be Weinstadt, the domicile of innotek. Solely erman law shall apply to this agreement.	•

Figure 1.6: 同意 VirtualBox 个人使用协议

弹出的 VirtualBox 用户注册对话框,可忽视关闭之:

à	VirtualBox Registration Dialog	×
Welcome to	e VirtualBox Registration Form!	
	Please fill out this registration form to let us know that you use VirtualBox and, optionally, to keep you informed about VirtualBox news and updates. Enter your full name using Latin characters and your e-mail address to the fields below. Please note that innotek will use this information only to gather product usage statistics and to send you VirtualBox newsletters. In particular, innotek will never pass your data to third parties. Detailed information about how we use your personal data can be found in the Privacy Policy page of the VirtualBox web-site.	
	Name E-mail Please do not use this information to contact me	

Figure 1.7: VirtualBox 用户注册对话框



接下来我们就见到了 VirtualBox 主界面:

Figure 1.8: VirtualBox 主界面

点击 New 按钮新建一个虚

2	Create New Virtual Machine	×
Welcome to the N	lew Virtual Machine Wizard!	
	This wizard will guide you through the steps that are necessary to create a new virtual machine for VirtualBox. Use the Next button to go the next page of the wizard and the Back button to return to the previous page.	
	< <u>B</u> ack <u>N</u> ext > <u>C</u> ancel	

Figure 1.9: 新建一个虚拟机

我们使用 solrex 作为虚	拟机的名字,系统类型未知:	
22	Create New Virtual Machine	×
VM Name and	d OS Type	
	Enter a name for the new virtual machine and select the of the guest operating system you plan to install onto a virtual machine. The name of the virtual machine usually indicates its software and hardware configuration. It will be used by VirtualBox components to identify your virtual machine	e type the all e.
	OS Iype Other/Unknown	OTNER
	< <u>B</u> ack <u>N</u> ext > <u>C</u>	ancel

Figure 1.10: 设置虚拟机名字和操作系统类型

设置虚拟机的内存容量,这里随便设了 32M:

2	Create New Virtual Machine
Memory	
	Select the amount of base memory (RAM) in megabytes to be allocated to the virtual machine. The recommended base memory size is 64 MB. Base Memory Size
	< <u>B</u> ack <u>N</u> ext > <u>C</u> ancel

Figure 1.11: 设置虚拟机内存容量

设置虚拟机硬盘镜像:		
2	Create New Virtual Machine	×
Virtual Hard Disk		
	Select a hard disk image to be used as the boot hard disk of the virtual machine. You can either create a new hard disk using the New button or select an existing hard disk image from the drop-down list or by pressing the Existing button (to invoke the Virtual Disk Manager dialog). If you need a more complicated hard disk setup, you can also skip this step and attach hard disks later using the VM Settings dialog. The recommended size of the boot hard disk is 2048 MB. Boot Hard Disk (Primary Master)	0
	N <u>e</u> w E <u>x</u> isting	
	< <u>B</u> ack <u>N</u> ext > <u>C</u> ancel	

Figure 1.12: 设置虚拟机硬盘镜像

如果没有硬盘镜像, 需点 "New" 新建一块硬盘镜像:

12 C	Create New Virtual Disk	×
Welcome to the C	reate New Virtual Disk Wizard!	
	This wizard will help you to create a new virtual hard disk image for your virtual machine. Use the Next button to go to the next page of the wizard and the Back button to return to the previous page.	I.
	< <u>B</u> ack <u>Next ></u> <u>C</u> ancel	

Figure 1.13: 新建一块虚拟硬盘

点

Select the type of virtual hard disk image you want to create. A dynamically expanding image initially occupies a very small amount of space on your physical hard disk. It will grow dynamically (up to the size specified) as the Guest OS claims disk space. A fixed-size image does not grow. It is stored in a file of approximately the same size as the size of the virtual hard disk.
Dynamically expanding image Eixed-size image

Figure 1.14: 设置虚拟硬盘类型

这里将虚拟硬盘镜像的名字设置为	"solrex"	,	并将容量设置为	"32M"	:	

16 C	Create New Virtual Disk	×
Virtual Disk Loc	cation and Size	
	Press the Select button to select the location and name of the file to store the virtual hard disk image or type a file name in the entry field. Image File Name /home/solrex/solrex.vdi Select the size of the virtual hard disk image in megabytes. This size will be reported to the Guest OS as the size of the virtual hard disk. Image Size 4.00 MB 2.00 TB]
	< <u>B</u> ack <u>N</u> ext > <u>C</u> ance	I

Figure 1.15: 设置虚拟硬盘镜像名字和容量

	Create New Virtual Disk	
Summary		
	You are going to create a new virtual hard the following parameters:	disk image with
	Type: Dynamically expanding ima	ige
	Location: /home/solrex/solrex.vdi	
	Size: 32.00 MB (33554432 Bytes)	
	If the above settings are correct, press the Once you press it, a new hard disk image	e Finish button. will be created.
	< <u>B</u> ack <u>E</u> inis	sh <u>C</u> ancel

最后查看新建的虚拟硬盘信息,点击 Finish 确认新建硬盘镜像:

Figure 1.16: 虚拟硬盘信息

令虚拟机使用已建立的虚拟硬盘 solrex.vdi:

	Create New Virtual Machine	×
Virtual Hard Disk		
	Select a hard disk image to be used as the boot hard disk of the virtual machine. You can either create a new hard disk using the New button or select an existing hard disk image from the drop-down list or by pressing the Existing button (to invoke the Virtual Disk Manager dialog).	D
	If you need a more complicated hard disk setup, you can also skip this step and attach hard disks later using the VM Settings dialog.	,
	The recommended size of the boot hard disk is 2048 MB.	
	Boot Hard Disk (Primary Master)	
	solrex.vdi (/home/solrex)	
	New Existing	
	< <u>B</u> ack <u>N</u> ext > <u>C</u> ancel	

Figure 1.17: 使用新建的虚拟硬盘

You are going to create a following parameters: Name: solre OS Type: Othe Base Memory: 32 M Boot Hard Disk: solre	new virtual machine with the x r/Unknown
Name: solre OS Type: Othe Base Memory: 32 M Boot Hard Disk: solre	r/Unknown R
OS Type: Othe Base Memory: 32 M Boot Hard Disk: solre	r/Unknown R
Base Memory: 32 M Boot Hard Disk: solre	R
Boot Hard Disk: solre	6
	x.vdi (/home/solrex)
If the above is correct propress it, a new virtual ma	ess the Finish button. Once you ichine will be created.
Note that you can alter the created virtual machine a dialog accessible through	hese and all other setting of the at any time using the Settings In the menu of the main window.

最后查看新建的虚拟机信息,点击 Finish 确认新建虚拟机:

Figure 1.18: 虚拟机信息



回到 VirtualBox 主界面, 左侧列表中有新建立的虚拟机 solrex:

Figure 1.19: 回到 VirtualBox 主界面

1.3.2 Bochs

1.4 使用软盘镜像

1.4.1 制作软盘镜像

前面我们说过,软盘的结构比较简单,所以我们选择使用软盘镜像来启动虚拟计算机。在 Linux 下制作一个软盘镜像很简单,只需要使用:

\$ dd if=/dev/zero of=emptydisk.img bs=512 count=2880

命令就可以在当前目录下生成一个名为 emptydisk.img 的空白软盘镜像,下面我们使用这个空白软盘 镜像来启动虚拟机。

dd:转换和拷贝文件的工具。dd 可以设置很多拷贝时候的参数,在本例中 if=FILE 选项 代表从 FILE 中读取内容; of=FILE 选项代表将导出输出到 FILE; bs=BYTES 代表每次读 取和输出 BYTES 个字节; count=BLOCKS 代表从输入文件中共读取 BLOCKS 个输入块。 而这里的 /dev/zero 则是一个 Linux 的特殊文件,读取这个文件可以得到持续的 0。那么 上面命令的意思就是以每块 512 字节共 2880 块全空的字符填入文件 emptydisk.img 中。我们 注意到前面提及的软盘容量计算公式:

2(磁头) × 80(磁道) × 18(扇区) × 512 bytes(扇区的大小) = 2880 x 512 bytes = 1440 KB = 1.44MB

可以发现我们用上述命令得到的就是一张全空的未格式化的软盘镜像。

1.4.2 用软盘镜像启动虚拟机

在虚拟机主界面选中虚拟机后点 Settings 按钮,进入虚拟机的设置界面:

	solrex - Settings
General	General
Hard Disks	
CD/DVD-ROM	Basic Advanced Description Other
Floppy	Identification
Audio	Name solrex OS Type Other/Unknown
Network	
Serial Ports	Base Memory Size
USB USB	32 MF
Shared Folders	4 MB 2000 MB
Remote Display	
	Video Memory Size
	8 ME
	1 MB 120 MB
	Select a settings category from the list on the left side and move the mouse over a settings item to get more information.

Figure 1.20: 虚拟机设置界面

	solrex - Settings	
 General Hard Disks CD/DVD-ROM 	Floppy ✓ Mount Floppy Drive	
 Corperbrokedia Floppy ▲ Audio Prevents Ø Serial Ports Ø USB Shared Folders Isplay 	O Host Floppy Drive] 🖬
	Floppy image file is not selected on the Floppy page.	

在左侧列表中选择 Floppy 进入虚拟机软盘设置界面:

Figure 1.21: 虚拟机软盘设置

点击 Image	File 最不	5侧的文件夹标志,	进入选择软盘镜像界面
----------	---------	-----------	------------

	Virtual Disk Manager	×
Action	ns	
New	Add Remove Release Refresh	
O H	ard Disks 📀 <u>C</u> D/DVD Images 💾 <u>F</u> loppy Images	
Na	me Size	
Lo	ocation:	
At	ttached to:	
н	lelp Select Cancel	
		11

Figure 1.22: 选择软盘镜像

点击 Add 按钮添加新的软盘镜像 emptydisk.img,	并点击 select 按钮选中其作为启动软盘:
----------------------------------	-------------------------

<u>a</u>	Virtual Disk Manager	×
Actions		
New Add	Remove Release Refresh	
<u>H</u> ard Disks	☑ CD/DVD Images Ploppy Images	
Name		Size
- emptydis	k.img	1.41 MB
Location: Attached to	/home/solrex/Desktop/Documents/WriteOS/chapter1a	ind2/emptydisk
Help	5	elect Cancel

Figure 1.23: 选择启动软盘镜像

8	solrex - Settings	
 General Hard Disks CD/DVD-ROM Floppy Audio Network Serial Ports USB Shared Folders Remote Display 	Floppy Mount Floppy Drive Host Floppy Drive Image File Image File Image File	
	Select a settings category from the list on the left side and move the mouse over a settings item to get more information.	
Help	<u>Q</u> K Cancel	

Figure 1.24: 确认启动镜像软盘文件信息

	innotek VirtualBox	
<u>F</u> ile <u>M</u> achine <u>H</u> elp		
	🙆 Details 🗑 Snapshots 🕢 Description	
New Settings Delete Start Discard	General Name solrex OS Type Other/Unknown Base Memory 32 MB Video Memory 8 MB Boot Order Floppy, CD/DVD-ROM, Hard Disk ACPI Enabled DATE Disklard	
	Hard Disks Primary Master solrex.vdi [Normal, 32.00 MB] CD/DVD-ROM Not mounted	
	Floppy Image emptydisk.img Audio Disabled	
	Network Adapter 0 NAT	

返回虚拟机主界面,查看右侧的虚拟机设置信息:

Figure 1.25: 查看虚拟机设置信息

选中虚拟机后,	双击或点击 Start 按钮运行它,第一次运行可能给出如下信息:
- The second sec	VirtualBox - Information
•	You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture the keyboard every time the VM window is activated and make it unavailable to other applications running on your host machine: when the keyboard is captured, all keystrokes (including system ones like Alt-Tab) will be directed to the VM.
	You can press the host key at any time to uncapture the keyboard and mouse (if it is captured) and return them to normal operation. The currently assigned host key is shown on the status bar at the bottom of the Virtual Machine window, next to the 🕑 icon. This icon, together with the mouse icon placed nearby, indicate the current keyboard and mouse capture state.
	OK

Figure 1.26: 自动键盘捕获警告信息

这个对话框的意思就是,当鼠标在虚拟机内部点击时,鼠标和键盘的消息将被虚拟机自动捕获,成为虚拟机的键盘和鼠标,可以敲击键盘右侧的 Ctrl 键解除捕获。

显示虚拟机的运行时内容:

- C			solrex [Running] - innotek VirtualBox	_ - ×
Machine	Devices	Help		
FATAL: N	lo boota	ble me	dium found! System halted.	
-				
			2 🖉 💾 🖉 🗂 🖉	🖲 Right Ctrl

Figure 1.27: 虚拟机运行时

我们可以看到在图 1.27 中,虚拟机加载空白软盘启动后提示消息为: "FATAL: No bootable medium found! System halted.",换成中文是找不到可启动媒体,系统停机。它的实际意思就是在前面第 1.1 节"计算机启动过程"中提到的第6步中虚拟机遍历了软驱、光驱、硬盘后没有找到可启动的媒体,所以就只好停机。因为我们在启动前已经在软驱中加载了软盘镜像,所以提示信息就表明那个软盘镜像不具有启动系统的功能,那么如何才能创建一个可启动的软盘呢,我们将在第2章介绍。

CHAPTER 2

最小的"操作系统"

任何一个完善的操作系统都是从启动扇区开始的,这一章,我们就关注如何写一个启动扇区,以及 如何将其写入到软盘镜像中。

先介绍一下需要使用的工具:

- 系统: Cent OS 5.1(RHEL 5.1)
- 使用工具: gcc, binutils(as, ld, objcopy), dd, make, hexdump, vim, virtualbox

2.1 Hello OS world!

◆ 本章节内容需要和 gcc, make 相关的 Linux C 语言编程以及 PC 汇编语言的基础知识。

很多编程书籍给出的第一个例子往往是在终端里输出一个字符串"Hello world!",那么要写操作 系统的第一步给出的例子自然就是如何在屏幕上打印出一个字符串喽。所以,我们首先看《自己动手写 操作系统》一书中给出的第一个示例代码,在屏幕上打印"Hello OS world!":

```
;告诉编译器程序加载到7c00处
     org
            07c00h
     mov
            ax, cs
2
     mov
            ds, ax
     mov
            es, ax
     call
           DispStr
                       ; 调用显示字符串例程
            $
                       ; 无限循环
     jmp
7 DispStr:
           ax, BootMessage
     mov
     mov
           bp, ax
                      ; ES:BP = 串地址
9
           cx, 16
                      ; CX = 串长度
10
     mov
           ax, 01301h ; AH = 13, AL = 01h
11
     mov
           bx, 000ch
                      ; 页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)
12
     mov
           dl, 0
     mov
13
```

 14
 int
 10h
 ; 10h 号中断

 15
 ret

 16 BootMessage:
 db
 "Hello, OS world!"

 17 times 510-(\$-\$\$)
 0; 填充剩下的空间,使生成的二进制代码恰好为512字节

 18 dw
 0xaa55
 ; 结束标志

Figure 2.1: 《自》第一个实例代码 boot.asm

2.1.1 Intel 汇编转化为 AT&T(GAS) 汇编

上面 boot.asm 中代码使用 Intel 风格的汇编语言写成,本也可以在 Linux 下使用同样开源的 NASM 编译,但是鉴于很少有人在 Linux 下使用此汇编语法,它在 Linux 平台上的扩展性和可调试性都不好(GCC 不兼容),而且不是采用 Linux 平台上编译习惯,所以我把它改成了使用 GNU 工具链去编译连接。这样的话,对以后使用 GNU 工具链编写其它体系结构的 bootloader 也有帮助,毕竟 NASM 没有GAS 用户多(也许 ③)。

上面的汇编源程序可以改写成 AT&T 风格的汇编源代码:

1	.code16	#	#使用16位模式汇编
2	.text	#	#代码段开始
3	mov	%cs,%ax	
4	mov	%ax,%ds	
5	mov	%ax,%es	
6	call	DispStr #	#调用显示字符串例程
7	jmp	. #	#无限循环
8	DispStr:		
9	mov	\$BootMessage	e, %ax
10	mov	%ax,%bp	#ES:BP = 串地址
11	mov	\$16,%cx	#CX = 串长度
12	mov	\$0x1301,%ax	#AH = 13, AL = 01h
13	mov	\$0x00c,%bx	#页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)
14	mov	\$0,%dl	
15	int	\$0x10	#10h 号中断
16	ret		
17	BootMessage	e:.ascii "Hel	llo, OS world!"
18	.org 510	#1	填充到~510~字节处
19	.word Oxaa	55 #4	结束标志

Figure 2.2: boot.S(chapter2/1/boot.S)

2.1.2 用连接脚本控制地址空间

但有一个问题,我们可以使用 nasm boot.asm -o boot.bin 命令将 boot.asm 直接编译成二进制文件, GAS 不能。不过 GAS 的不能恰好给开发者一个机会去分步地实现从汇编源代码到二进制文件这个过程, 使编译更为灵活。下面请看 GAS 是如何通过连接脚本控制程序地址空间的:

1 SECTIONS {

 $_2$. = 0x7c00;

Chapter 2. 最小的"操作系统"

```
3 .text:
4 {
5 _ftext = .; //程序将被加载到内存的7c00处
6 } = 0
7 }
```

Figure 2.3: boot.S 的连接脚本 (chapter2/1/solrex_x86.ld)

连接脚本: GNU 连接器 ld 的每一个连接过程都由连接脚本控制。连接脚本主要用于,怎 样把输入文件内的 section 放入输出文件内,并且控制输出文件内各部分在程序地址空间内的 布局。连接器有个默认的内置连接脚本,可以用命令 ld -verbose 查看。选项-T 选项可以指 定自己的连接脚本,它将代替默认的连接脚本。

这个连接脚本的功能就是,在连接的时候,将程序入口设置为内存 0x7c00 的位置(BOIS 将 PC 控制权转交给这个位置运行的程序),相当于 boot.asm 中的 org 07c00h 一句。有人可能觉得麻烦,还 需要用一个脚本控制加载地址,但是《自己动手写操作系统》就给了一个很好的反例:《自》第 1.5 节代码 1-2,作者切换调试和运行模式时候需要对代码进行注释。

```
1;%define _BOOT_DEBUG_ ;做 Boot Sector 时一定将此行注释掉!将此行打开后用
                    ; nasm Boot.asm -o Boot.com 做成一个.COM文件易于调试
2
3
4 %ifdef _BOOT_DEBUG_
    org 0100h ; 调试状态, 做成 .COM 文件, 可调试
5
6 %else
     org 07c00h ; Boot 状态, Bios 将把 Boot Sector 加载到 0:7C00 处并开始执行
7
8 %endif
9
10
    mov
          ax, cs
11
    mov
         ds, ax
12
    mov
          es, ax
    call DispStr ; 调用显示字符串例程
13
    jmp
         $
                   ; 无限循环
14
15 DispStr:
   mov ax, BootMessage
16
    mov bp, ax ; ES:BP = 串地址
17
        cx, 16
                   ; CX = 串长度
18
   mov
        ax, 01301h ; AH = 13, AL = 01h
19
   mov
        bx,000ch ; 页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)
   mov
20
        dl, 0
21
    mov
        10h
                   ; 10h 号中断
22
    int
    ret
23
24 BootMessage: db "Hello, OS world!"
                   0; 填充剩下的空间, 使生成的二进制代码恰好为512字节
25 times 510-($-$$) db
26 dw 0xaa55
                     ; 结束标志
```

Figure 2.4: 《自》代码 1-2 (chapter2/1/boot.asm)

而如果换成使用脚本控制程序地址空间,只需要编译时候调用不同脚本进行连接,就能解决这个问题。这在嵌入式编程中是很常见的处理方式,即使用不同的连接脚本一次 make 从一个源程序文件生成分别运行在开发板上和软件模拟器上的两个二进制文件。

2.1.3 用 Makefile 编译连接

下面的这个 Makefile 文件,就是我们用来自动编译 boot.S 汇编源代码的脚本文件:

```
1 CC=gcc
2 LD=1d
3 LDFILE=solrex_x86.ld #使用上面提供的连接脚本 solrex_x86.ld
4 OBJCOPY=objcopy
6 all: boot.img
s # Step 1: gcc 调用 as 将 boot.S 编译成目标文件 boot.o
9 boot.o: boot.S
10
        $(CC) -c boot.S
11
12 # Step 2: ld 调用连接脚本 solrex_x86.ld 将 boot.o 连接成可执行文件 boot.elf
13 boot.elf: boot.o
14
        $(LD) boot.o -o boot.elf -e c -T$(LDFILE)
15
16 # Step 3: objcopy 移除 boot.elf 中没有用的 section(.pdr,.comment,.note),
17 #
          strip 掉所有符号信息,输出为二进制文件 boot.bin 。
18 boot.bin : boot.elf
        @$(OBJCOPY) -R .pdr -R .comment -R.note -S -O binary boot.elf boot.bin
19
20
21 # Step 4: 生成可启动软盘镜像。
22 boot.img: boot.bin
                                                           #用 boot.bin 生成镜像文件第一个扇区
        @dd if=boot.bin of=boot.img bs=512 count=1
23
         # 在 bin 生成的镜像文件后补上空白,最后成为合适大小的软盘镜像
24
         @dd if=/dev/zero of=boot.img skip=1 seek=1 bs=512 count=2879
25
26
27 clean:
^{28}
        @rm -rf boot.o boot.elf boot.bin boot.img
```

Figure 2.5: boot.S 的 Makefile(chapter2/1/Makefile)

我们将上面内容保存成 Makefile, 与图 2.2 所示 boog.S 和图 2.3 所示 solrex_x86.ld 放在同一个目录下, 然后在此目录下使用下面命令编译:

```
$ make
gcc -c boot.S
ld boot.o -o boot.elf -e c -Tsolrex_x86.ld
ld: warning: cannot find entry symbol c; defaulting to 00007c00
1+0 records in
1+0 records out
512 bytes (512 B) copied, 3.1289e-05 seconds, 16.4 MB/s
2879+0 records in
2879+0 records out
1474048 bytes (1.5 MB) copied, 0.0141508 seconds, 104 MB/s
$ ls
boot.asm boot.elf boot.o Makefile solrex_x86.ld
boot.bin boot.img boot.S solrex.img
```

 $\mathbf{22}$

Chapter 2. 最小的"操作系统"

可以看到,我们只需执行一条命令 make 就可以编译、连接和直接生成可启动的软盘镜像文件,其间对源文件的每一步处理也都一清二楚。不用任何商业软件,也不用自己写任何转换工具,比如《自己动手写操作系统》文中提到的 HD-COPY 和 Floopy Writer 都没有使用到。

在这里需要特别注意的是图 2.5 中的 Step 4,其实对这一步的解释应该结合图 1.4 来查看。我们用 boot.S 编译生成的 boot.bin 其实只是图 1.4 中所指的软盘的启动扇区,例如 boot.S 最后一行:

.word 0xaa55 #结束标志

生成就是启动扇区最后的 0xaa55 那两个字节,而 boot.bin 的大小是 512 字节,正好是启动扇区的大小。那么 Step 4 的功能就是把 boot.bin 放入到一个空白软盘的启动扇区,这样呢当虚拟机启动时能识别出这是一张可启动软盘,并且执行我们在启动扇区中写入的打印代码。

为了验证软盘镜像文件的正确性也可以先用

\$ hexdump -x -n 512 boot.img

将 boot.img 前 512 个字节打印出来,可以看到 boot.img dump 的内容和《自》一书附送光盘中的 TINIX.IMG dump 的内容完全相同。这里我们也显然用不到 EditPlus 或者 UltraEdit,即使需要修改 二进制码,也可以使用 hexedit, ghex2, khexedit 等工具对二进制文件进行修改。

下图为使用命令行工具 hexedit 打开 boot.img 的窗口截图,从图中我们可以看到,左列是该行开头与文件头对应的偏移地址,中间一列是文件的二进制内容,最右列是文件内容的 ASCII 显示内容,可以看到,此界面与 UltraEdit 的十六进制编辑界面没有本质不同。

	solrex	@loc	alho	st:~	/De	skt	op/	Doc	um	ent	s/W	rite) 5/	cha	pte	rland2			×
<u>File</u> <u>E</u> dit	View	Term	inal	Tab	s l	Help	,												
00000000	BC C	8 8E	D8	8E	CO	E8	02	00	EB	FE	B8	1E	7C	89	C5		- 1		-
00000010	B9 1	0 00	B 8	01	13	BB	0C	00	B2	00	CD	10	С3	48	65			He	
00000020	6C 6	C 6F	2C	20	4F	53	20	77	6F	72	6C	64	21	00	00	llo, OS worl	d!		
00000030	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000040	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000050	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00			• •	
00000060	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000070	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000080	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000090	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
000000A0	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
000000B0	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				_
00000000	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				-
000000D0	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
000000E0	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
000000F0	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000100	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00			• •	
00000110	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000120	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000130	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000140	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000150	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
00000160	00 0	0 00	00	00	00	00	00	00	00	00	00	00	00	00	00				
boot	t.img		0	x0/0	9×16	5800	90												-

Figure 2.6: 使用 hexedit 打开 boot.img

2						fil	e://	/ho	me,	sol	rex	/bo	ot.i	img	- K	(He	xEd	lit		×
<u>F</u> ile	<u>E</u> dit	<u>V</u> ie	w	Doc	um	ents	Ē	200	kma	rks	Τœ	ols	<u>S</u> e	ettin	gs	H	elp			
2	3	7 6				4		9	3	Đ	C		d	•	-		Θ	8		B
0000	:0000	8c	с8	8e	d8	8e	c0	e8	02	00	eb	fe	b8	1e	7c	89	c5	.0.0.000	00.1.0	•
0000	:0010	b9	10	00	b 8	01	13	bb	θc	00	b2	00	cd	10	c3	48	65	0000	.[].[]He	112
0000	:0020	6C	6C	61	2c	20	41	53	20	77	61	72	6C	64	21	00	00	llo, OS wo	rld!	
0000	:0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	- 8858	00	88	88	88	00	00	88	88	88	00	00	88	88	88	00	88			
0000	:0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	:0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	:0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	:0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	:00a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	:00b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			
0000	:00c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00			•
Hex	-															Fi	nd	B <u>a</u> ckwar	ds 🔲 Ignore case [X
	Signed	8 bi	t: [-11	6		Sig	ned	32 1	bit:	Г	-66	173	118	88	н	lexadecimal:	80	
			-			=	100					1				=				1
Un	signed	8 bi	t		14	0	Ur	nsig	ned	32 1	bit:	L	363	323	610	8		Octal:	214	1
Si	gned 1	6 bi	t: [-1	419	6			32 bi	t flo	at:	-1	255	936	E+1	15		Binary:	10001100	
Unsi	gned 1	6 bi	t: [5	134	0		6	54 bi	t flo	at:	1.	2111	1218	-29	4		Text:		
XS	how <u>l</u> itt	tle e	ndia	an d	eco	ding		S	now	unsi	igne	ed a	s he	xad	leci	mal	St	ream length:	Fixed 8 Bit	
	Encoding: Default																			

Figure 2.7: 使用 kde 图形界面工具 khexedit 打开 boot.img

2.1.4 用虚拟机加载执行 boot.img

ctions	
New Add Remove Release	(C) Refresh
<u>Hard Disks</u> <u>CD/DVD Images</u>	1 Eloppy Images
Name	Size
- boot.img	1.41 ME
Location: /home/solrex/Desk	top/Documents/WriteOS/chapter1and2/boot.img
Location: /home/solrex/Desk Attached to:	ktop/Documents/WriteOS/chapter1and2/boot.img

当我们生成 boot.img 之后, 仿照第 1.4.2 节中加载软盘镜像的方法, 用虚拟机加载 boot.img:

Figure 2.8: 选择启动软盘镜像 boot.img



Figure 2.9: 虚拟机启动后打印出红色的"Hello OS world!"

我们看到虚拟机如我们所料的打印出了红色的"Hello OS world!"字样,这说明我们以上的程序和 编译过程是正确的。

2.2 FAT 文件系统

我们在上一节中介绍的内容,仅仅是写一个启动扇区并将其放入软盘镜像的合适位置。由于启动扇 区 512 字节的大小限制,我们仅仅能写入像打印一个字符串这样的非常简单的程序,那么如何突破 512 字节的限制呢? 很显然的答案是我们要利用其它的扇区,将程序保存在其它扇区,运行前将其加载到内 存后再跳转过去执行。那么又一个问题产生了:程序在软盘上应该怎样存储呢?

可能最直接最容易理解的存储方式就是顺序存储,即将一个大程序从启动扇区开始按顺序存储在相 邻的扇区,可能这样需要的工作量最小,在启动时操作系统仅仅需要序列地将可执行代码拷贝到内存中 来继续运行。可是经过简单的思考我们就可以发现这样做有几个缺陷:1.软盘中仅能存储操作系统程 序,无法存储其它内容;2.我们必须使用二进制拷贝方式来制作软盘镜像,修改系统麻烦。

那么怎么避免这两个缺点呢?引入文件系统可以让我们在一张软盘上存储不同的文件,并提供文件 管理功能,可以让我们避免上述的两个缺点。在使用某种文件系统对软盘格式化之后,我们可以像普通 软盘一样使用它来存储多个文件和目录,为了使用软盘上的文件,我们给启动扇区的代码加上寻找文件 和加载执行文件功能,让启动扇区将系统控制权转移给软盘上的某个文件,这样突破启动扇区 512 字节 大小的限制。

2.2.1 FAT12 文件系统

FAT(File Allocation Table) 文件系统规格在 20 世纪 70 年代末和 80 年代初形成,是微软的 MS-DOS 操作系统使用的文件系统格式。它的初衷是为小于 500K 容量的软盘制定的简单文件系统,但在

将近三十年的发展过程中,它已经被一次次修改加强以支持更大的存储媒体。在目前主要有三种 FAT 文件系统类型: FAT12, FAT16 和 FAT32。这几种类型最基本的区别就像它们的名字字面区别一样, 主要在于大小,即盘上 FAT 表的记录项所占的比特数。FAT12 的记录项占 12 比特, FAT16 占 16 比特, FAT32 占 32 比特。

由于 FAT12 最为简单和易实施,这里我们仅简单介绍 FAT12 文件系统,想要了解更多 FAT 文件 系统知识的话,可以到 http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx 下载微软发布的 FAT 文件系统官方文档。

FAT12 文件系统和其它文件系统一样,都将磁盘划分为层次进行管理。从逻辑上划分,一般将磁盘划分为分区,目录和文件;从抽象物理结构来讲,将磁盘划分为分区,簇和扇区。那么,如何将逻辑上的目录和文件映射到物理上实际的簇和扇区,就是文件系统要解决的问题。

如果让虚拟机直接读取我们上一节生成的可启动软盘镜像,或者将 boot.img 软盘用 mount -o loop boot.img mountdir/挂载到某个目录上,系统肯定会报出"软盘未格式化"或者"文件格式不可识别"的错误。这是因为任何系统可读取的软盘都是被格式化过的,而我们的 boot.img 是一个非常 原始的软盘镜像。那么如何才能使软盘被识别为 FAT12 格式的软盘并且可以像普通软盘一样存取呢?

系统在读取一张软盘的时候,会读取软盘上存储的一些关于文件系统的信息,软盘格式化的过程也 就是系统把文件系统信息写入到软盘上的过程。但是我们不能让系统来格式化我们的 boot.img,如果 那样的话,我们写入的启动程序也会被擦除。所以呢,我们需要自己对软盘进行格式化。●可能有人看 到这里就会很沮丧,天那,那该有多麻烦啊!不过我相信在读完以下内容以后你会欢呼雀跃,啊哈,原 来文件系统挺简单的嘛!

2.2.2 启动扇区与 BPB

FAT 文件系统的主要信息,都被提供在前几个扇区内,其中第0号扇区尤其重要。在这个扇区内 隐藏着一个叫做 BPB(BIOS Parameter Block)的数据结构,一旦我们把这个数据结构写对了,格式化 过程也基本完成了③。下面这个表中所示内容,主要就是启动扇区的 BPB 数据结构。

名称	偏移	大小	描述	Solrex.img
	bytes	bytes		文件中的值
$BS_jmpBoot$	0	3	跳转指令,用于过以下的扇区信息	jmp LABEL_START
				nop
BS_OEMName	3	8	厂商名	"WB. YANG"
BPB_BytesPerSec	11	2	扇区大小(字节),应为512	512
BPB_secPerClus	13	1	簇的扇区数, 应为2 的幂, FAT12 为1	1
BPB_RsvdSecCnt	14	2	保留扇区, FAT12/16 应为1	1
BPB_NumFATs	16	1	FAT 结构数目,一般为2	2
BPB_RootEntCnt	17	2	根目录项目数, FAT12 为224	224
BPB_TotSec16	19	2	扇区总数, 1.44M 软盘为2880	2880
BPB_Media	21	1	设备类型, 1.44M 软盘为FOh	0xf0
BPB_FATSz16	22	2	FAT 占用扇区数,9	9

表 2.1: 启动扇区的 BPB 数据结构和其它内容

BPB_SecPerTrk	24	2	磁道扇区数,18	18
BPB_NumHeads	26	2	磁头数,2	2
BPB_HiddSec	28	4	隐藏扇区,默认为0	0
BPB_TotSec32	32	4	如果 BPB_TotSec16 为 0,它记录总扇区数	0
	-	下面的扇	区头信息 FAT12/FAT16 与 FAT32 不同	
BS_DrvNum	36	1	中断 0x13 的驱动器参数,0 为软盘	0
BS_Reserved1	37	1	Windows NT 使用, O	0
BS_BootSig	38	1	扩展引导标记 (29h),指明此后 3 个域可用	0x29
BS_VolID	39	4	卷标序列号,0000000h	0
BS_VolLab	43	11	卷标,11 字节,必须用空格20h 补齐	"Solrex 0.01"
BS_FilSysType	54	8	文件系统标志, "FAT12 "	"FAT12 "
			以下为非扇区头信息部分	
启动代码及其它	62	448	启动代码、数据及填充字符	mov %cs,%ax
启动扇区标识符	510	2	可启动扇区标志, OxAA55	0xaa55

哇, 天那, 这个 BPB 看起来很多东西的嘛, 怎么写啊?其实写入这些信息很简单, 因为它们都是 固定不变的内容, 用下面的代码就可以实现。

21 /* Floppy header of	f FAT12 *,	/	
22 jmp LABEL_S	START /* S	Start to boot.	*/
23 nop	/*]	nop required */	/
24 BS_OEMName:	.ascii	"WB. YANG"	<pre>/* OEM String, 8 bytes required */</pre>
<pre>25 BPB_BytsPerSec:</pre>	.2byte	512	/* Bytes per sector */
26 BPB_SecPerCluse:	.byte	1	/* Sector per cluse */
27 BPB_ResvdSecCnt:	.2byte	1	/* Reserved sector count */
28 BPB_NumFATs:	.byte	2	/* Number of FATs */
29 BPB_RootEntCnt:	.2byte	224	/* Root entries count */
30 BPB_TotSec16:	.2byte	2880	/* Total sector number */
31 BPB_Media:	.byte	0xf0	/* Media descriptor */
32 BPB_FATSz16:	.2byte	9	/* FAT size(sectors) */
33 BPB_SecPerTrk:	.2byte	18	/* Sector per track */
34 BPB_NumHeads:	.2byte	2	/* Number of magnetic heads */
35 BPB_HiddSec:	.4byte	0	/* Number of hidden sectors */
36 BPB_TotSec32:	.4byte	0	<pre>/* If TotSec16 equal 0, this works */</pre>
37 BS_DrvNum:	.byte	0	/* Driver number of interrupt 13 */
38 BS_Reserved1:	.byte	0	/* Reserved */
39 BS_BootSig:	.byte	0x29	/* Boot signal */
40 BS_VolID:	.4byte	0	/* Volume ID */
41 BS_VolLab:	.ascii	"Solrex 0.01"	/* Volume label, 11 bytes required */
<pre>42 BS_FileSysType:</pre>	.ascii	"FAT12 "	<pre>/* File system type, 8 bytes required */</pre>
43			
44 /* Initial register	rs. */		
45 LABEL_START:			

Figure 2.10: 生成启动扇区头的汇编代码(节自chapter2/2/boot.S)

在上面的汇编代码中,我们只是顺序地用字符填充了启动扇区头的数据结构,填充的内容与表 2.1 中最后一列的内容相对应。把图 2.10 中所示代码添加到图 2.2 的第二行和第三行之间,然后再 make,就能得到一张已经被格式化,可启动也可存储文件的软盘,就是既可以使用 mount -o loop boot.img mountdir/命令在普通 Linux 系统里挂载,也可用作虚拟机启动的软盘镜像文件。

2.2.3 FAT12 数据结构

在上一个小节里,我们制作出了可以当作普通软盘使用的启动软盘,这样我们就可以在这张软盘上存储多个文件了。可还有一步要求我们没有达到,怎样寻找存储的某个引导文件并将其加载到内存中运行呢?这就涉及到 FAT12 文件系统中文件的存储方式了,需要我们了解一些 FAT 数据结构和目录结构的知识。

FAT 文件系统对存储空间分配的最小单位是"簇",因此文件在占用存储空间时,基本单位是簇而不是字节。即使文件仅仅有1字节大小,系统也必须分给它一个最小存储单元--簇。由表2.1中的 BPB_secPerClus 和 BPB_BytsPerSec 相乘可以得到每簇所包含的字节数,可见我们设置的是每簇包含 1*512=512 个字节,恰好是每簇包含一个扇区。

存储空间分配的最小单位确定了,那么 FAT 是如何分配和管理这些存储空间的呢? FAT 的存储空间管理是通过管理 FAT 表来实现的, FAT 表一般位于启动扇区之后,根目录之前的若干个扇区,而且一般是两个表。从根目录区的下一个簇开始,每个簇按照它在磁盘上的位置映射到 FAT 表里。FAT 文件系统的存储结构粗略上来讲如图 2.11 所示。



Figure 2.11: FAT 文件系统存储结构图

FAT 表的表项有点儿像数据结构中的单向链表节点的 next 指针,先回忆一下,单向链表节点的数据结构(C语言)是:

```
struct node {
    char * data;
    struct node *next;
};
```

在链表中, next 指针指向的是下一个相邻节点。那么 FAT 表与链表有什么区别呢?首先, FAT 表将 next 指针集中管理,放在一起被称为 FAT 表;其次, FAT 表项指向的是固定大小的文件"簇"(data

段),而且每个文件簇都有自己对应的 FAT 表项。由于每个文件簇都有自己的 FAT 表项,这个表项可能指向另一个文件簇,所以 FAT 表项所占字节的多少就决定了 FAT 表最大能管理多少内存, FAT12 的 FAT 表项有 12 个比特,大约能管理 2¹² – 个文件簇。

一个文件往往要占据多个簇,只要我们知道这个文件的第一个簇,就可以到 FAT 表里查询该簇对 应的 FAT 表项,该表项的内容一般就是此文件下一个簇号。如果该表项的值大于 0xff8,则表示该簇 是文件最后一个簇,相当于单向链表节点的 next 指针为 NULL;如果该表项的值是 0xff7则表示它是 一个坏簇。这就是 文件的链式存储。

2.2.4 FAT12 根目录结构

怎样读取一个文件我们知道了,但是如何找到某个文件,即如何得到该文件对应的第一个簇呢?这 就到目录结构派上用场的时候了,为了简单起见,我们这里只介绍根目录结构。

如图 2.11 所示,对于 FAT12/16,根目录存储在磁盘中固定的地方,紧跟在最后一个 FAT 表之后。根目录的扇区数也是固定的,可以根据 BPB_RootEntCnt 计算得出:

```
RootDirSectors = ((BPB_RootEntCnt * 32) + (BPB_BytesPerSec - 1)) / BPB_BytesPerSec
```

根目录的扇区号是相对于该 FAT 卷启动扇区的偏移量:

```
FirstRootDirSecNum = BPB_RsvdSecCnt + (BPB_NumFATs * BPB_FATSz16)
```

FAT 根目录其实就是一个由 32-bytes 的线性表构成的"文件",其每一个条目代表着一个文件,这个 32-bytes 目录项的格式如图 2.2 所示。

名称	偏移(bytes)	长度(bytes)	描述	举例(loader.bin)
DIR_Name	0	0xb	文件名8字节,扩展名3字节	"LOADER 🗆 🗆 BIN"
DIR_Attr	0xb	1	文件属性	0x20
保留位	0xc	10	保留位	0
DIR_WrtTime	0x16	2	最后一次写入时间	0x7a5a
DIR_WrtDate	0x18	2	最后一次写入日期	0x3188
DIR_FstClus	0x1a	2	此目录项的开始簇编号	0x0002
DIR_FileSize	0x1c	4	文件大小	0x000000f

表 2.2: 根目录的条目格式

知道了这些,我们就得到了足够的信息去在磁盘上寻找某个文件,在磁盘根目录搜索并读取某个文件的步骤大致如下:

1. 确定根目录区的开始扇区和结束扇区;

2. 遍历根目录区,寻找与被搜索名相对应根目录项;

3. 找到该目录项对应的开始簇编号;

4. 以文件的开始簇为根据寻找整个文件的链接簇,并依次读取每个簇的内容。

2.3 让启动扇区加载引导文件

有了 FAT12 文件系统的相关知识之后,我们就可以跨越 512 字节的限制,从文件系统中加载文件并执行了。

2.3.1 一个最简单的 loader

为做测试用,我们写一个最小的程序,让它显示一个字符,然后进入死循环,这样如果 loader 加载成功并成功执行的话,就能看到这个字符。

新建一个文件 loader.S, 内容如图 2.12 所示。

11	.code16	
12	.text	
13	mov	\$0xb800,%ax
14	movw	%ax,%gs
15	mov	\$0xf,%ah
16	mov	\$'L',%al
17	mov	%ax,%gs:((80*0+39)*2)
18	jmp	

Figure 2.12: 一个最简单的 loader(chapter2/2/loader.S)

这个程序在连接时需要使用连接文件 solrex_x86_dos.ld,如图 2.13 所示,这样能更改代码段的偏移 量为 0x0100。这样做的目的仅仅是为了与 DOS 系统兼容,可以用此代码生成在 DOS 下可调试的二进 制文件。

11	SECTIONS
12	{
13	$. = 0 \times 0100;$
14	.text :
15	{
16	_ftext = .;
17	} = 0
18	}

Figure 2.13: 一个最简单的 loader(chapter2/2/solrex_x86_dos.ld)

2.3.2 读取软盘扇区的 BIOS 13h 号中断

我们知道了如何在磁盘上寻找一个文件,但是该如何将磁盘上内容读取到内存中去呢?我们在第 2.1节中写的启动扇区不需要自己写代码来读取,是因为它每次都被加载到内存的固定位置,计算机在 发现可启动标识 0xaa55 的时候自动就会做加载工作。但如果我们想自己从软盘上读取文件的时候,就 需要使用到底层 BIOS 系统提供的磁盘读取功能了。这里,我们主要用到 BIOS 13h 号中断。 表 2.3 所示,就是 BIOS 13 号中断的参数表。从表中我们可以看到,读取磁盘驱动器所需要的参数 是磁道(柱面)号、磁头号以及当前磁道上的扇区号三个分量。由第 1.2 节所介绍的磁盘知识,我们可 以得到计算这三个分量的公式 2.1。

$$\frac{\overline{\mathbf{k}}\overline{\mathbf{k}}\overline{\mathbf{k}}}{\mathbf{k}(\overline{\mathbf{k}}\overline{\mathbf{k}}\overline{\mathbf{k}}\overline{\mathbf{k}}\overline{\mathbf{k}})} = \begin{cases} \overline{\mathbf{k}}\overline{\mathbf{k}}\overline{\mathbf{k}}\overline{\mathbf{k}} = Q >> 1\\ \overline{\mathbf{k}}\overline$$

中断号	AH	功能	调用参数 返回参数			
	0	磁盘复位	DL = 驱动器号	失败:		
			00,01 为软盘,80h,81h,为硬盘	AH = 错误码		
	1 读磁盘驱动器状态			AH=状态字节		
	2	读磁盘扇区	AL = 扇区数	读成功:		
			$(CL)_{6,7}(CH)_{0\sim7} = 磁盘号$	AH = O		
			$(CL)_{0\sim 5} = 扇区 $	AL = 读取的扇区数		
			DH/DL = 磁头号/驱动器号	读失败:		
			ES:BX = 数据缓冲区地址	AH = 错误码		
	3	写磁盘扇区	同上	写成功:		
				AH = O		
				AL = 写入的扇区数		
				写失败:		
13				AH = 错误码		
	4	检验磁盘扇区	AL = 扇区数	成功:		
			$(CL)_{6,7}(CH)_{0\sim7} = 磁盘号$	AH = O		
			$(CL)_{0\sim 5} = 扇区号$	AL = 检验的扇区数		
			DH/DL = 磁头号/驱动器号	失败: AH = 错误码		
	5	格式化盘磁道	AL = 扇区数	成功:		
			$(CL)_{6,7}(CH)_{0\sim7} = 磁盘号$	AH = O		
			$(CL)_{0\sim 5} = 扇区 $	失败:		
			DH/DL = 磁头号/驱动器号	AH = 错误码		
			ES:BX = 格式化参数表指针			

表 2.3: BIOS 13h 号中断的参数表

知道了这些,我们就可以写一个读取软盘扇区的子函数了:

Action: Read %cl Sectors from %ax sector(floppy) to %es:%bx(memory)

```
Assume sector number is 'x', then:
212
         x/(BPB_SecPerTrk) = y,
213
         x%(BPB_SecPerTrk) = z.
214
        The remainder 'z' PLUS 1 is the start sector number;
215
        The quotient 'y' devide by BPB_NumHeads(RIGHT SHIFT 1 bit) is cylinder
216
         number;
217
        AND 'y' by 1 can got magnetic header.
218
219 */
220 ReadSector:
               %ebp
221
       push
               %esp,%ebp
222
       mov
               $2,%esp
                              /* Reserve space for saving %cl */
223
       sub
               %cl,-2(%ebp)
224
      mov
225
      push
             %bx
                              /* Save bx */
              (BPB_SecPerTrk), %bl /* %bl: the devider */
226
       mov
                             /* 'y' in %al, 'z' in %ah */
227
      div
              %bl
      inc
            %ah
                              /* z++, got start sector */
228
            %ah,%cl
                             /* %cl <- start sector number */
229
      mov
             %al,%dh
                              /* %dh <- 'y' */
230
      mov
               $1,%al
                              /* 'y'/BPB_NumHeads */
231
      \operatorname{shr}
               %al,%ch
                              /* %ch <- Cylinder number(y>>1) */
232
      mov
               $1,%dh
                              /* %dh <- Magnetic header(y&1) */
233
      and
                              /* Restore %bx */
234
      pop
               %bx
      /* Now, we got cylinder number in %ch, start sector number in %cl, magnetic
235
         header in %dh. */
236
       mov
               (BS_DrvNum), %dl
237
238 GoOnReading:
       mov
               $2,%ah
239
               -2(%ebp),%al
                               /* Read %al sectors */
240
       mov
      int
               $0x13
241
               GoOnReading
                               /* If CF set 1, mean read error, reread. */
242
       jc
               $2,%esp
243
       add
               %ebp
244
       pop
^{245}
       ret
246
```

Figure 2.14: 读取软盘扇区的函数(节自chapter2/2/boot.S)

2.3.3 搜索 loader.bin

62

读取扇区的子函数写好了,下面我们编写在软盘中搜索 loader.bin 的代码:

```
/* Reset floppy */
63
              %ah,%ah
64
      xor
              %dl,%dl
                             /* %dl=0: floppy driver 0 */
65
      xor
              $0x13
                             /* BIOS int 13h, ah=0: Reset driver 0 */
66
      int
67
      /* Find LOADER.BIN in root directory of driver 0 */
68
      movw
            $SecNoOfRootDir, (wSectorNo)
69
70
71 /* Read root dir sector to memory */
```

 $\mathbf{32}$

```
72 LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
              $0,(wRootDirSizeForLoop)
                                          /* If searching in root dir */
73
      cmpw
              LABEL_NO_LOADERBIN
                                          /* can find LOADER.BIN ? */
      jz
74
      decw
              (wRootDirSizeForLoop)
75
              $BaseOfLoader,%ax
      mov
76
              %ax,%es
                                          /* %es <- BaseOfLoader*/
      mov
77
              $0ffsetOfLoader,%bx
                                          /* %bx <- OffsetOfLoader */
      mov
78
              (wSectorNo),%ax
                                         /* %ax <- sector number in root */</pre>
      mov
79
              $1,%cl
      mov
80
      call ReadSector
81
              $LoaderFileName,%si
                                         /* %ds:%si -> LOADER BIN */
82
      mov
              $0ffsetOfLoader,%di
                                         /* BaseOfLoader<<4+100*/
83
      mov
      cld
84
85
      mov
             $0x10,%dx
86
87 /* Search for "LOADER BIN", FAT12 save file name in 12 bytes, 8 bytes for
     file name, 3 bytes for suffix, last 1 bytes for '\20'. If file name is
88
      less than 8 bytes, filled with '\20'. So "LOADER.BIN" is saved as:
89
      "LOADER BIN"(4f4c 4441 5245 2020 4942 204e).
90
91 */
92 LABEL_SEARCH_FOR_LOADERBIN:
                                          /* Read control */
93
    cmp
             $0,%dx
              LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR
^{94}
      jz
95
      dec
              %dx
    mov
              $11,%cx
96
97
98 LABEL_CMP_FILENAME:
    cmp
              $0.%cx
99
      jz
              LABEL_FILENAME_FOUND
                                    /* If 11 chars are all identical? */
100
      dec
              %cx
101
      lodsb
                                      /* %ds:(%si) -> %al*/
102
              %es:(%di),%al
103
      \mathtt{cmp}
            LABEL_GO_ON
104
      jz
              LABEL_DIFFERENT
                                      /* Different */
105
      jmp
106
107 LABEL_GO_ON:
108
     inc
              %di
      jmp
              LABEL_CMP_FILENAME
                                     /* Go on loop */
109
110
111 LABEL_DIFFERENT:
      and
              $0xffe0,%di
                                      /* Go to head of this entry */
112
      add
              $0x20,%di
113
              $LoaderFileName,%si
                                    /* Next entry */
      mov
114
              LABEL_SEARCH_FOR_LOADERBIN
      jmp
115
116
117 LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
    addw
              $1,(wSectorNo)
118
              LABEL_SEARCH_IN_ROOT_DIR_BEGIN
119
      jmp
120
121 /* Not found LOADER.BIN in root dir. */
122 LABEL_NO_LOADERBIN:
              $2,%dh
123
    mov
            DispStr
                             /* Display string(index 2) */
124
    call
                             /* Infinite loop */
125
    jmp
              .
126
```

127 /*	Found. *	k/	
128 LA	BEL_FILEN	NAME_FOUND:	
129	mov	<pre>\$RootDirSectors,%ax</pre>	
130	and	\$0xffe0,%di	<pre>/* Start of current entry, 32 bytes per entry */</pre>
131	add	\$0x1a,%di	<pre>/* First sector of this file */</pre>
132	mov	%es:(%di),%cx	
133	push	%cx	<pre>/* Save index of this sector in FAT */</pre>
134	add	%ax,%cx	
135	add	<pre>\$DeltaSecNo,%cx</pre>	<pre>/* LOADER.BIN's start sector saved in %cl */</pre>
136	mov	\$BaseOfLoader,%ax	
137	mov	%ax,%es	/* %es <- BaseOfLoader */
138	mov	<pre>\$0ffsetOfLoader,%bx</pre>	/* %bx <- OffsetOfLoader */
139	mov	%cx,%ax	/* %ax <- Sector number */
140			

Figure 2.15: 搜索 loader.bin 的代码片段(节自chapter2/2/boot.S)

这段代码的功能就是我们前面提到过的,遍历根目录的所有扇区,将每个扇区加载入内存,然后从中寻找文件名为 loader.bin 的条目,直到找到为止。找到之后,计算出 loader.bin 的起始扇区号。其中用到的变量和字符串的定义见图 2.16 中代码片段的定义。

```
12 .set BaseOfStack, 0x7c00 /* Stack base address, inner */
 13 .set BaseOfLoader, 0x9000 /* Section loading address of LOADER.BIN */

      13. Set
      Dasedificater, 0x0000
      /* Dection roading address of Londer.Dir.

      14. set
      OffsetOfLoader, 0x0100
      /* Loading offset of LOADER.BIN */

      15. set
      RootDirSectors, 14
      /* Root directory sector count */

      16. set
      SecNoOfRootDir, 19
      /* 1st sector of root directory */

      17. set
      SecNoOfFAT1, 1
      /* 1st sector of FAT1 */

      18. set
      DeltaSecNo, 17
      /* BPB_(RsvdSecCnt+NumFATs*FATSz) -2 */

174
176 Variable table
177 */
178 wRootDirSizeForLoop: .2byte RootDirSectors
179 wSectorNo:.2byte0/* Sector number to read */
                                     .byte O
                                                            /* odd or even? */
180 bOdd:
181
182 /* -----
183 String table
184 */
185 LoaderFileName: .asciz "LOADER BIN" /* File name */
186 .set MessageLength,9

        187 BootMessage:
        .ascii
        "Booting**"
        /* index 0 */

        188 Message1:
        .ascii
        "Loaded in"
        /* index 1 */

188 Message1:
189 Message2:
                               .ascii "No LOADER"
                                                                         /* index 2 */
190
```

Figure 2.16: 搜索 loader.bin 使用的变量定义(节自chapter2/2/boot.S)

由于在代码中有一些打印工作,我们写了一个函数专门做这项工作。为了节省代码长度,被打印字 符串的长度都设置为9字节,不够则用空格补齐,这样就相当于一个备用的二维数组,通过数字定位要 打印的字符串,很方便。打印字符串的函数 DispStr 见图 2.17,调用它的时候需要从寄存器 dh 传入参数字符串序号。

```
190
191 /* =========
                           _____
      Routine: DispStr
192
      Action: Display a string, string index stored in %dh
193
194 */
195 DispStr:
196
      mov
              $MessageLength, %ax
197
      mul
              %dh
              $BootMessage,%ax
      add
198
      mov
              %ax,%bp
                                    /* String address */
199
              %ds,%ax
      mov
200
              %ax,%es
      mov
201
              $MessageLength,%cx
                                    /* String length */
202
      mov
              $0x1301,%ax
                                    /* ah = 0x13, al = 0x01(W) */
203
      mov
              $0x07,%bx
                                    /* PageNum 0(bh = 0), bw(bl= 0x07)*/
204
      mov
              $0,%dl
                                    /* Start row and column */
205
      mov
                                    /* BIOS INT 10h, display string */
              $0x10
206
      int
207
      ret
208
```

Figure 2.17: 打印字符串函数 DispStr (节自chapter2/2/boot.S)

2.3.4 加载 loader 入内存

在寻找到 loader.bin 之后,就需要把它装入内存。现在我们已经有了 loader.bin 的起始扇区号,利 用这个扇区号可以做两件事:一,把起始扇区装入内存;二,通过它找到 FAT 中的条目,从而找到 loader.bin 文件所占用的其它扇区。

这里,我们把 loader.bin 装入内存中的 BaseOfLoader:OffsetOfLoader 处,但是在图 2.15 中我们将 根目录区也是装载到这个位置。因为在找到 loader.bin 之后,该内存区域对我们已经没有用处了,所以 它尽可以被覆盖。

我们已经知道了如何装入一个扇区,但是从 FAT 表中寻找其它的扇区还是意见麻烦的事情,所以 我们写了一个函数 GetFATEntry 来专门做这件事情,函数的输入是扇区号,输出是其对应的 FAT 项的 值,见图 2.18。

```
246
247 /* =========
                            _____
     Routine: GetFATEntry
248
     Action: Find %ax sector's index in FAT, save result in %ax
249
250 */
251 GetFATEntry:
252
      push
             %es
253
      push
             %bx
      push
             %ax
254
      mov
             $BaseOfLoader,%ax
255
             $0x0100,%ax
      sub
256
```

257	mov	%ax,%es	/*	Left 4K bytes for FAT */
258	рор	%ax		
259	movb	\$0,(bOdd)		
260	mov	\$3,%bx		
261	mul	%bx	/*	%dx:%ax = %ax*3 */
262	mov	\$2,%bx		
263	div	%bx	/*	%dx:%ax/2 */
264	cmp	\$0,%bx	/*	remainder %dx = 0 ? */
265	jz	LABEL_EVEN		
266	movb	\$1,(bOdd)		
267				
268 LAB	EL_EVEN:			
269	xor	%dx,%dx	/*	Now %ax is the offset of FATEntry in FAT */
270	mov	(BPB_BytsPerSec),%	bx	
271	div	%bx	/*	%dx:%ax/BPB_BytsPerSec */
272	push	%dx		
273	mov	\$0,%bx		
274	add	\$SecNoOfFAT1,%ax	/*	%ax <- FATEntry's sector */
275	mov	\$2,%cl	/*	Read 2 sectors in 1 time, because FATEntry */
276	call	ReadSector	/*	may be in 2 sectors. */
277	рор	%dx		
278	add	%dx,%bx		
279	mov	%es:(%bx),%ax		
280	cmpb	\$1,(bOdd)		
281	jnz	LABEL_EVEN_2		
282	shr	\$4,%ax		
283				
284 LAB	EL_EVEN_2	2:		
285	and	\$0x0fff,%ax		
286				
287 LAB	EL_GET_FA	AT_ENTRY_OK:		
288	pop	%bx		
289	pop	%es		
290	ret			
201				

Figure 2.18: 寻找 FAT 项的函数 GetFATEntry (节自chapter2/2/boot.S)

这样,我们就可以将 loader.bin 整个文件加载到内存中去了,见图 2.19。

140			
141 /	* Load LO	ADER.BIN's se	ector's to memory. */
142 I	ABEL_GOON	LOADING_FILE	I:
143	push	%ax	
144	push	%bx	
145	mov	\$0x0e,%ah	
146	mov	\$'.',%al	/* Char to print */
147	mov	\$0x0f,%bl	<pre>/* Front color: white */</pre>
148	int	\$0x10	/* BIOS int 10h, ah=0xe: Print char */
149	pop	%bx	
150	pop	%ax	
151			
152	mov	\$1,%cl	

Chapter 2. 最小的"操作系统"

153	call	ReadSector									
154	рор	%ax	/* (Got :	index	of	this	sector	in	FAT	*/
155	call	GetFATEntry									
156	cmp	\$0x0fff,%ax									
157	jz	LABEL_FILE_LOAD	DED								
158	push	%ax	/* 5	Save	index	of	this	sector	in	FAT	' */
159	mov	\$RootDirSectors	s,%dx								
160	add	%dx,%ax									
161	add	<pre>\$DeltaSecNo,%as</pre>	ĸ								
162	add	(BPB_BytsPerSec	c),%bz	ĸ							
163	jmp	LABEL_GOON_LOAD	DING_H	FILE							
164											
165	LABEL_FILE_	LOADED:									
166	mov	\$1,%dh									
167	call	DispStr	/* Di	ispla	ay str	ing	(inde	x 1) */	'		
168											

Figure 2.19: 加载 loader.bin 的代码(节自chapter2/2/boot.S)

在图 2.19 中我们看到一个宏 DeltaSectorNo,这个宏就是为了将 FAT 中的簇号转换为扇区号。由于根目录区的开始扇区号是 19,而 FAT 表的前两个项 0,1 分别是磁盘识别字和被保留,其表项其实是从第 2 项开始的,第 2 项对应着根目录区后的第一个扇区,所以扇区号和簇号的对应关系就是:

扇区号 = 簇号 + 根目录区占用扇区数 + 根目录区开始扇区号 - 2
 = 簇号 + 根目录区占用扇区数 + 17

这就是 DeltaSectorNo 的值 17 的由来。

2.3.5 向 loader 转交控制权

我们已经将 loader 成功地加载入了内存,然后就需要进行一个跳转,来执行 loader。

Figure 2.20: 跳转到 loader 执行(节自chapter2/2/boot.S)

2.3.6 生成镜像并测试

我们写好了汇编源代码,那么就需要将源代码编译成可执行文件,并生成光盘镜像了。

11 CC=gcc 12 LD=ld 13 OBJCOPY=objcopy 14 15 CFLAGS=-c $_{16}\ TRIM_FLAGS=-R$.pdr -R .comment -R.note -S -O binary 17 18 LDFILE_BOOT=solrex_x86_boot.ld 19 LDFILE_DOS=solrex_x86_dos.ld 20 LDFLAGS_BOOT=-e c -T\$(LDFILE_BOOT) 21 LDFLAGS_DOS=-e c -T\$(LDFILE_DOS) 2223 all: boot.img LOADER.BIN 24 25 boot.bin: boot.S 26 \$(CC) \$(CFLAGS) boot.S \$(LD) boot.o -o boot.elf \$(LDFLAGS_BOOT) 27\$(OBJCOPY) \$(TRIM_FLAGS) boot.elf \$@ 28 29 30 LOADER.BIN: loader.S \$(CC) \$(CFLAGS) loader.S 31 \$(LD) loader.o -o loader.elf \$(LDFLAGS_DOS) 32 \$(OBJCOPY) \$(TRIM_FLAGS) loader.elf \$@ 33 34 35 boot.img: boot.bin 36 dd if=boot.bin of=boot.img bs=512 count=1 37 dd if=/dev/zero of=boot.img skip=1 seek=1 bs=512 count=2879 38

Figure 2.21: 用 Makefile 编译(节自chapter2/2/boot.S)

上面的代码比较简单,我们可以通过一个 make 命令编译生成 boot.img 和 LOADER.BIN:

```
$ make
gcc -c boot.S
boot.S:293:67: warning: no newline at end of file
ld boot.o -o boot.elf -e c -Tsolrex_x86_boot.ld
ld: warning: cannot find entry symbol c; defaulting to 00007c00
objcopy -R .pdr -R .comment -R.note -S -O binary boot.elf boot.bin
dd if=boot.bin of=boot.img bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 6.5651e-05 seconds, 7.8 MB/s
dd if=/dev/zero of=boot.img skip=1 seek=1 bs=512 count=2879
2879+0 records in
2879+0 records out
1474048 bytes (1.5 MB) copied, 0.0305594 seconds, 48.2 MB/s
gcc -c loader.S
ld loader.o -o loader.elf -e c -Tsolrex_x86_dos.ld
ld: warning: cannot find entry symbol c; defaulting to 00000100
objcopy -R .pdr -R .comment -R.note -S -O binary loader.elf LOADER.BIN
```

由于我们的目标就是让启动扇区加载引导文件,所以需要把引导文件放入软盘镜像中。那么如何

38

Chapter 2. 最小的"操作系统"

38

将 LOADER.BIN 放入 boot.img 中呢?我们只需要挂载 boot.img 并将 LOADER.BIN 拷贝进入被挂载 的目录,为此我们在 Makefile 中添加新的编译目标 copy:

```
39 # You must have the authority to do mount, or you must use "su root" or
40 # "sudo" command to do "make copy"
41 copy: boot.img LOADER.BIN
42 mkdir -p /tmp/floppy;\
43 mount -o loop boot.img /tmp/floppy/ -o fat=12;\
44 cp LOADER.BIN /tmp/floppy/;\
45 umount /tmp/floppy/;\
46 rm -rf /tmp/floppy/;
47
```

Figure 2.22: 拷贝 LOADER.BIN 入 boot.img(节自chapter2/2/boot.S)

由于挂载软盘镜像在很多 Linux 系统上需要 root 权限,所以我们没有将 copy 目标添加到 all 的依赖关系中。在执行 make copy 命令之前我们必须先获得 root 权限。

```
$ su
Password:
# make copy
mkdir -p /tmp/floppy;\
            mount -o loop boot.img /tmp/floppy/ -o fat=12;\
            cp LOADER.BIN /tmp/floppy/;\
            umount /tmp/floppy/;\
            rm -rf /tmp/floppy/;
```

如果仅仅生成 boot.img 而不将 loader.bin 装入它,用这样的软盘启动会显示找不到 LOADER:



Figure 2.23: 没有装入 LOADER.BIN 的软盘启动

装入 loader.bin 之后再用 boot.img 启动,我们看到虚拟机启动并在屏幕中间打印出了一个字符 "L",这说明我们前面的工作都是正确的。

2			solrex [Running] - innotek VirtualBox	
Machine	<u>D</u> evices	<u>H</u> elp		
Booting* Loaded i	*, n		L	
Louucu	"-			
			🗹 ⊘ 💾 🗇 🖉 🖉 🗷 Rig	ht Ctrl

Figure 2.24: 装入了 LOADER.BIN 以后再启动

2.4 本章全部源代码

为了避免本书官方网站发生问题,所以我们特别将每章的源代码附在每章最后一节,以方便读者使用。

```
1 /* chapter2/1/boot.S
 2
     Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
3
 4
     This file is part of the source code of book "Write Operating System with
 5
     Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
 6
 7
     This file is licensed under the GNU General Public License; either
 8
 9
     version 3 of the License, or (at your option) any later version. \ast/
10
11 .code16
12 .text
13
    mov
                 %cs,%ax
                 %ax,%ds
14
    mov
15
    mov
                 %ax,%es
16
    call
              DispStr
17
      jmp
                 .
18 DispStr:
                 $BootMessage, %ax
    mov
19
```

```
Chapter 2. 最小的"操作系统"
```

```
%ax,%bp
      mov
20
                  $16,%cx
21
      mov
                  $0x1301,%ax
      mov
22
                  $0x00c,%bx
      mov
23
                  $0,%dl
      mov
24
                  $0x10
      int
25
      ret
26
27 BootMessage:.ascii "Hello, OS world!"
28 .org 510
29 .word 0xaa55
```

Figure 2.25: chapter2/1/boot.S

```
1 /* chapter2/2/boot.S
2
     Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
3
4
     This file is part of the source code of book "Write Operating System with
5
     Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
6
     This file is licensed under the GNU General Public License; either
8
     version 3 of the License, or (at your option) any later version. */
9
10
11 SECTIONS
12 {
   . = 0x7c00;
13
    .text :
14
15
   ſ
16
      _ftext = .;
  } = 0
17
18 }
```



```
1 # chapter2/1/Makefile
2 #
3 # Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
4 #
5 # This file is part of the source code of book "Write Operating System with
6 # Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
7 #
8 # This file is licensed under the GNU General Public License; either
9 # version 3 of the License, or (at your option) any later version.
10
11 CC=gcc
12 LD=ld
13 LDFILE=solrex_x86.ld
14 OBJCOPY=objcopy
15
16 all: boot.img
```

```
17
18 boot.img: boot.bin
      @dd if=boot.bin of=boot.img bs=512 count=1
19
      @dd if=/dev/zero of=boot.img skip=1 seek=1 bs=512 count=2879
20
21
22 boot.bin: boot.elf
      @$(OBJCOPY) -R .pdr -R .comment -R.note -S -O binary boot.elf boot.bin
23
24
25 boot.elf: boot.o
    $(LD) boot.o -o boot.elf -e c -Tsolrex_x86.ld
26
27
28 boot.o: boot.S
    $(CC) -c boot.S
29
30
31 clean:
    @rm -f boot.o boot.elf boot.bin
32
33
34 distclean:
35 Orm -f *.img
```

Figure 2.27: chapter2/1/Makefile

```
1 /* chapter2/2/boot.S
 2
     Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
з
 4
    This file is part of the source code of book "Write Operating System with
\mathbf{5}
    Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
 6
 7
     This file is licensed under the GNU General Public License; either
 8
    version 3 of the License, or (at your option) any later version. */
9
10
11 .code16
        BaseOfStack,
                        0x7c00
                                  /* Stack base address, inner */
12 .set
                       0x9000
                                 /* Section loading address of LOADER.BIN */
        BaseOfLoader,
13 .set
                                 /* Loading offset of LOADER.BIN */
14 .set OffsetOfLoader, 0x0100
15 .set RootDirSectors, 14 /* Root directory sector count */
16 .set SecNoOfRootDir, 19
                                 /* 1st sector of root directory */
17 .set SecNoOfFAT1, 1
                                 /* 1st sector of FAT1 */
                                  /* BPB_(RsvdSecCnt+NumFATs*FATSz) -2 */
18 .set DeltaSecNo,
                        17
19 /* Start sector of file space =*/
20 .text
21 /* Floppy header of FAT12 */
22 jmp LABEL_START /* Start to boot. */
                       /* nop required */
23 nop
24 BS_OEMName:
                    .ascii "WB. YANG" /* OEM String, 8 bytes required */
25 BPB_BytsPerSec:
                    .2byte 512
                                        /* Bytes per sector */
26 BPB_SecPerCluse: .byte 1
                                        /* Sector per cluse */
27 BPB_ResvdSecCnt: .2byte 1
                                        /* Reserved sector count */
28 BPB_NumFATs:
                    .byte 2
                                        /* Number of FATs */
29 BPB_RootEntCnt: .2byte 224
                                        /* Root entries count */
30 BPB_TotSec16:
                   .2byte 2880
                                        /* Total sector number */
31 BPB_Media:
                    .byte 0xf0
                                        /* Media descriptor */
```

42

```
32 BPB_FATSz16:
                     .2byte 9
                                          /* FAT size(sectors) */
33 BPB_SecPerTrk:
                     .2byte 18
                                          /* Sector per track */
                                          /* Number of magnetic heads */
34 BPB_NumHeads:
                     .2byte 2
                                          /* Number of hidden sectors */
35 BPB_HiddSec:
                     .4byte 0
36 BPB_TotSec32:
                     .4byte 0
                                          /* If TotSec16 equal 0, this works */
37 BS_DrvNum:
                     .byte O
                                          /* Driver number of interrupt 13 */
38 BS_Reserved1:
                     .byte 0
                                          /* Reserved */
39 BS_BootSig:
                                          /* Boot signal */
                     .byte 0x29
40 BS_VolID:
                                          /* Volume ID */
                     .4byte 0
                     .ascii "Solrex 0.01" /* Volume label, 11 bytes required */
41 BS_VolLab:
42 BS_FileSysType:
                     .ascii "FAT12 " /* File system type, 8 bytes required */
43
44 /* Initial registers. */
45 LABEL_START:
46
    mov
             %cs,%ax
             %ax,%ds
47
     mov
             %ax,%es
48
    mov
             %ax,%ss
49
   mov
             $BaseOfStack, %sp
50
    mov
51
   /* Clear screen */
52
   mov $0x0600,%ax /* %ah=6, %al=0 */
53
             $0x0700,%bx /* Black white */
54
     mov
     mov
          $0,%cx /* Top left: (0,0) */
55
          $0x184f,%dx /* Bottom right: (80,50) */
     mov
56
     int $0x10
                          /* BIOS int 10h, ah=6: Initialize screen */
57
58
      /* Display "Booting**" */
59
     mov
             $0,%dh
60
     call
             DispStr
                           /* Display string(index 0)*/
61
62
      /* Reset floppy */
63
             %ah,%ah
64
     xor
65
             %dl,%dl
                           /* %dl=0: floppy driver 0 */
      xor
                           /* BIOS int 13h, ah=0: Reset driver 0 */
      int
             $0x13
66
67
      /* Find LOADER.BIN in root directory of driver 0 */
68
      movw
             $SecNoOfRootDir, (wSectorNo)
69
70
71 /* Read root dir sector to memory */
72 LABEL_SEARCH_IN_ROOT_DIR_BEGIN:
                                         /* If searching in root dir */
      cmpw
             $0,(wRootDirSizeForLoop)
73
             LABEL_NO_LOADERBIN
                                         /* can find LOADER.BIN ? */
      jz
74
             (wRootDirSizeForLoop)
      decw
75
     mov
             $BaseOfLoader,%ax
76
                                         /* %es <- BaseOfLoader*/
     mov
             %ax,%es
77
                                         /* %bx <- OffsetOfLoader */
             $0ffsetOfLoader,%bx
      mov
78
             (wSectorNo),%ax
                                         /* %ax <- sector number in root */</pre>
79
      mov
             $1.%cl
80
     mov
             ReadSector
81
      call
             $LoaderFileName,%si
                                        /* %ds:%si -> LOADER BIN */
82
     mov
                                        /* BaseOfLoader<<4+100*/
             $0ffsetOfLoader,%di
83
     mov
84
     cld
             $0x10,%dx
85
     mov
86
```

```
87 /* Search for "LOADER BIN", FAT12 save file name in 12 bytes, 8 bytes for
      file name, 3 bytes for suffix, last 1 bytes for '\20'. If file name is
 88
      less than 8 bytes, filled with '\20'. So "LOADER.BIN" is saved as:
 89
      "LOADER BIN"(4f4c 4441 5245 2020 4942 204e).
90
91 */
92 LABEL_SEARCH_FOR_LOADERBIN:
               $0,%dx
                                           /* Read control */
      cmp
93
              LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR
       jz
94
      dec
              %dx
95
              $11,%cx
96
      mov
97
98 LABEL_CMP_FILENAME:
              $0,%cx
99
      \mathtt{cmp}
              LABEL_FILENAME_FOUND
                                       /* If 11 chars are all identical? */
100
      jz
101
       dec
               %cx
                                       /* %ds:(%si) -> %al*/
102
      lodsb
      cmp
            %es:(%di),%al
103
             LABEL_GO_ON
104
     jz
            LABEL_DIFFERENT
                                       /* Different */
105
      jmp
106
107 LABEL_GO_ON:
108
      inc
               %di
              LABEL_CMP_FILENAME
                                       /* Go on loop */
109
       jmp
110
111 LABEL_DIFFERENT:
112
      and
            $0xffe0,%di
                                       /* Go to head of this entry */
      add
            $0x20,%di
113
               $LoaderFileName,%si
      mov
                                      /* Next entry */
114
      jmp
            LABEL_SEARCH_FOR_LOADERBIN
115
116
117 LABEL_GOTO_NEXT_SECTOR_IN_ROOT_DIR:
      addw $1,(wSectorNo)
118
              LABEL_SEARCH_IN_ROOT_DIR_BEGIN
119
       jmp
120
121 /* Not found LOADER.BIN in root dir. */
122 LABEL_NO_LOADERBIN:
              $2,%dh
123
       mov
                               /* Display string(index 2) */
       call
               DispStr
124
                               /* Infinite loop */
125
       jmp
                .
126
127 /* Found. */
128 LABEL_FILENAME_FOUND:
              $RootDirSectors,%ax
129
      mov
               $0xffe0,%di
                                       /* Start of current entry, 32 bytes per entry */
      and
130
              $0x1a,%di
                                       /* First sector of this file */
      add
131
              %es:(%di),%cx
      mov
132
     push
              %cx
                                       /* Save index of this sector in FAT */
133
              %ax,%cx
      add
134
               $DeltaSecNo,%cx
                                       /* LOADER.BIN's start sector saved in %cl */
      add
135
               $BaseOfLoader,%ax
136
      mov
               %ax,%es
                                       /* %es <- BaseOfLoader */
137
      mov
                                       /* %bx <- OffsetOfLoader */
               $0ffsetOfLoader,%bx
138
      mov
                                       /* %ax <- Sector number */
139
      mov
              %cx,%ax
140
```

 $_{\rm 141}$ /* Load LOADER.BIN's sector's to memory. */

44

Chapter 2. 最小的"操作系统"

```
142 LABEL_GOON_LOADING_FILE:
143
   push
          %ax
   push
144
          %bx
          $0x0e,%ah
    mov
145
          $'.',%al
                  /* Char to print */
    mov
146
         $0x0f,%bl /* Front color: white */
    mov
147
    int
         $0x10 /* BIOS int 10h, ah=0xe: Print char */
148
   pop
          %bx
149
          %ax
150
   pop
151
         $1,%cl
152
   mov
   call ReadSector
153
   pop %ax
                     /* Got index of this sector in FAT */
154
   call GetFATEntry
155
156 cmp $0x0fff,%ax
         LABEL_FILE_LOADED
157 jz
   push %ax /* Save index of this sector in FAT */
158
   mov
         $RootDirSectors,%dx
159
   add %dx,%ax
160
161 add $DeltaSecNo,%ax
   add (BPB_BytsPerSec),%bx
162
   jmp LABEL_GOON_LOADING_FILE
163
164
165 LABEL_FILE_LOADED:
166 mov $1,%dh
   call DispStr
                   /* Display string(index 1) */
167
168
Jump to LOADER.BIN's start address in memory.
171 */
        $BaseOfLoader,$OffsetOfLoader
172 jmp
174
175 /* -----
176 Variable table
177 */
178 wRootDirSizeForLoop: .2byte RootDirSectors
179 wSectorNo:
                  .2byte 0 /* Sector number to read */
                  .byte 0
180 bOdd:
                             /* odd or even? */
181
183 String table
184 */
185 LoaderFileName: .asciz "LOADER BIN" /* File name */
186 .set MessageLength,9
                      "Booting**"
                                   /* index 0 */
187 BootMessage: .ascii
                                   /* index 1 */
               .ascii
                      "Loaded in"
188 Message1:
              .ascii "No LOADER"
                                   /* index 2 */
189 Message2:
190
192 Routine: DispStr
193 Action: Display a string, string index stored in %dh
194 */
195 DispStr:
196 mov $MessageLength, %ax
```

%dh mul 197 \$BootMessage,%ax 198 add mov %ax,%bp /* String address */ 199 %ds,%ax 200 mov %ax,%es 201 mov \$MessageLength,%cx /* String length */ 202 mov \$0x1301,%ax /* ah = 0x13, al = 0x01(W) */ mov 203 \$0x07,%bx /* PageNum 0(bh = 0), bw(bl= 0x07)*/ mov 204 /* Start row and column */ \$0,%dl mov 205 \$0x10 /* BIOS INT 10h, display string */ 206 int 207 ret 208 Routine: ReadSector 210 Action: Read %cl Sectors from %ax sector(floppy) to %es:%bx(memory) 211Assume sector number is 'x', then: 212 $x/(BPB_SecPerTrk) = y,$ 213x%(BPB_SecPerTrk) = z. 214The remainder 'z' PLUS 1 is the start sector number; 215The quotient 'y' devide by BPB_NumHeads(RIGHT SHIFT 1 bit)is cylinder 216 217 number; AND 'y' by 1 can got magnetic header. 218219 */ 220 ReadSector: 221 push %ebp 222 %esp,%ebp mov \$2,%esp /* Reserve space for saving %cl */ 223 sub %cl,-2(%ebp) 224 mov push %bx /* Save bx */ 225mov (BPB_SecPerTrk), %bl /* %bl: the devider */ 226 div %bl /* 'y' in %al, 'z' in %ah */ 227 %ah /* z++, got start sector */ 228 inc %ah,%cl /* %cl <- start sector number */ 229 mov %al,%dh /* %dh <- 'y' */ 230 mov \$1,%al /* 'y'/BPB_NumHeads */ 231 shr %al,%ch /* %ch <- Cylinder number(y>>1) */ 232mov /* %dh <- Magnetic header(y&1) */</pre> 233 and \$1,%dh %bx /* Restore %bx */ 234pop /* Now, we got cylinder number in %ch, start sector number in %cl, magnetic 235header in %dh. */ 236 (BS_DrvNum), %dl mov 237 238 GoOnReading: \$2,%ah mov 239 -2(%ebp),%al /* Read %al sectors */ mov 240int \$0x13 241jc GoOnReading /* If CF set 1, mean read error, reread. */ 242 add \$2,%esp 243 %ebp 244 pop 245ret 246Routine: GetFATEntry 248Action: Find %ax sector's index in FAT, save result in %ax 249250 */ 251 GetFATEntry:

Chapter 2. 最小的"操作系统"

252	push	%es	
253	push	%bx	
254	push	%ax	
255	mov	<pre>\$BaseOfLoader,%ax</pre>	
256	sub	\$0x0100,%ax	
257	mov	%ax,%es /	* Left 4K bytes for FAT */
258	pop	%ax	
259	movb	\$0,(bOdd)	
260	mov	\$3,%bx	
261	mul	%bx /	/* %dx:%ax = %ax*3 */
262	mov	\$2,%bx	
263	div	%bx /	/* %dx:%ax/2 */
264	cmp	\$0,%bx /	* remainder %dx = 0 ? */
265	jz	LABEL_EVEN	
266	movb	\$1,(bOdd)	
267			
268	LABEL_EVEN:		
269	xor	%dx,%dx /	* Now %ax is the offset of FATEntry in FAT */
270	mov	(BPB_BytsPerSec),%b	X
271	div	%bx /	* %dx:%ax/BPB_BytsPerSec */
272	push	%dx	
273	mov	\$0,%bx	
274	add	<pre>\$SecNoOfFAT1,%ax /</pre>	* %ax <- FATEntry's sector */
275	mov	\$2,%cl /	* Read 2 sectors in 1 time, because FATEntry *
276	call	ReadSector /	* may be in 2 sectors. */
277	pop	%dx	
278	add	%dx,%bx	
279	mov	%es:(%bx),%ax	
280	cmpb	\$1,(bOdd)	
281	jnz	LABEL_EVEN_2	
282	shr	\$4,%ax	
283			
284	LABEL_EVEN_	2:	
285	and	\$0x0fff,%ax	
286			
287	LABEL_GET_F	AT_ENTRY_OK:	
288	pop	%bx	
289	pop	%es	
290	ret		
291			
292	.org 510	<pre>/* Skip to addr</pre>	ress 0x510. */
293	.2byte Oxaa	55 /* Write boot f	lag to 1st sector(512 bytes) end $*/$

Figure 2.28: chapter 2/2/boot.S

1 /* chapter2/2/loader.S
2
3 Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
4
5 This file is part of the source code of book "Write Operating System with
6 Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
7
8 This file is licensed under the GNU General Public License; either

9	version	3 of the	License,	or	(at	your	option)	any	later	version.	*/
10											
11	.code16										
12	.text										
13	mov	\$0xb800,%ax									
14	movw	%ax,%gs									
15	mov	\$0xf	,%ah								
16	mov	\$'L',%al									
17	mov	%ax,%gs:((80*0+39)*2)									
18	jmp										

Figure 2.29: chapter2/2/loader.S

```
1 /* chapter2/2/solrex_x86_boot.ld
2
     Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
з
4
     This file is part of the source code of book "Write Operating System with
5
     Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
6
     This file is licensed under the GNU General Public License; either
8
     version 3 of the License, or (at your option) any later version. \ast/
9
10
11 SECTIONS
12 {
    . = 0x7c00;
13
14
    .text :
15
   {
16
      _ftext = .;
17
   } = 0
18 }
```



```
1 /* chapter2/2/solrex_x86_dos.ld
 2
3
     Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
 4
     This file is part of the source code of book "Write Operating System with
 \mathbf{5}
     Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
 6
 \overline{7}
     This file is licensed under the GNU General Public License; either
 8
     version 3 of the License, or (at your option) any later version. */
9
10
_{11} SECTIONS
12 {
13
   = 0x0100;
14
   .text :
15 {
      _ftext = .;
16
```

$\mathbf{48}$

```
17 } = 0
18 }
```

Figure 2.31: chapter2/2/solrex_x86_dos.ld

```
1 # chapter2/2/Makefile
2 #
3 # Author: Wenbo Yang <solrex@gmail.com> <http://solrex.cn>
4 #
5 # This file is part of the source code of book "Write Operating System with
6 # Free Software". The latest version, see <http://share.solrex.cn/WriteOS/>.
7 #
8 # This file is licensed under the GNU General Public License; either
9 # version 3 of the License, or (at your option) any later version.
10
11 CC=gcc
12 LD=ld
13 OBJCOPY=objcopy
14
15 CFLAGS=-c
16 TRIM_FLAGS=-R .pdr -R .comment -R.note -S -O binary
17
18 LDFILE_BOOT=solrex_x86_boot.ld
19 LDFILE_DOS=solrex_x86_dos.ld
20 LDFLAGS_BOOT=-e c -T$(LDFILE_BOOT)
21 LDFLAGS_DOS=-e c -T$(LDFILE_DOS)
22
23 all: boot.img LOADER.BIN
^{24}
25 boot.bin: boot.S
      $(CC) $(CFLAGS) boot.S
26
      $(LD) boot.o -o boot.elf $(LDFLAGS_BOOT)
27
      $(OBJCOPY) $(TRIM_FLAGS) boot.elf $@
^{28}
29
30 LOADER.BIN: loader.S
      $(CC) $(CFLAGS) loader.S
31
      $(LD) loader.o -o loader.elf $(LDFLAGS_DOS)
32
      $(OBJCOPY) $(TRIM_FLAGS) loader.elf $@
33
^{34}
35 boot.img: boot.bin
36
      dd if=boot.bin of=boot.img bs=512 count=1
37
      dd if=/dev/zero of=boot.img skip=1 seek=1 bs=512 count=2879
38
39 # You must have the authority to do mount, or you must use "su root" or
40 # "sudo" command to do "make copy"
41 copy: boot.img LOADER.BIN
    mkdir -p /tmp/floppy;\
42
    mount -o loop boot.img /tmp/floppy/ -o fat=12;\
^{43}
44
    cp LOADER.BIN /tmp/floppy/;\
^{45}
    umount /tmp/floppy/;\
    rm -rf /tmp/floppy/;
46
47
48 clean:
```

```
49  @rm -f *.o *.elf *.bin *.BIN
50
51 distclean:
52  @rm -f *.img
```

Figure 2.32: chapter 2/2/Makefile

50