bootsect.s分析—— Linux-0.11学习笔记（一）

为了节省篇幅，完整的代码就不贴了。感兴趣的朋友可以去下载，下载地址是：

http://oldlinux.org/Linux.old/

本文，我打算详解bootsect.s。如有纰缪，还请各位看官斧正。关于如何讲好代码，我暂时没有找到什么好的展示方法。姑且贴一段、注释一段、讲一段吧。为了不使代码片太长，我删去了一些原来的注释。

一些符号常量

SYSSIZE = 0x3000 ;system模块的长度

.globl begtext, begdata, begbss, endtext, enddata, endbss

.text

begtext:

.data

begdata:

.bss

begbss:

.text

SETUPLEN = 4 ! setup模块的长度，4个扇区

BOOTSEG = 0x07c0 ! original address of boot-sector

INITSEG = 0x9000 ! bootsect把自身搬运到0x90000

SETUPSEG = 0x9020 ! setup模块被加载到 0x90200

SYSSEG = 0x1000 ! system模块被加载到0x10000

ENDSEG = SYSSEG + SYSSIZE ! where to stop loading, 0x1000 + 0x3000 = 0x4000, 停止加载的段地址

ROOT\_DEV = 0x306 !第2个硬盘的第1个分区

ROOT\_DEV = 0x306 ，这里的0x306表示第2个硬盘的第1个分区，当年Linus是在第2个硬盘的第1个分区上安装了Linux-0.11操作系统。

老式Linux设备号的命名规则

设备号 = 主设备号 \* 256 + 次设备号

或者说：

dev\_no = (major << 8) + minor

这里的主设备号是事先定义好的（1-内存，2-磁盘，3-硬盘，4-ttyx，5-tty，6-并行口，7-非命名管道）。譬如对于硬盘，主设备号为3，因此3\*256+0=0x300即为系统中第一个硬盘的设备号。更多的例子如下表：

设备号 设备文件 对应的设备

0x300 /dev/hd0 系统中第一个硬盘

0x301 /dev/hd1 系统中第一个硬盘的第一分区

0x302 /dev/hd2 系统中第一个硬盘的第二分区

0x303 /dev/hd3 系统中第一个硬盘的第三分区

0x304 /dev/hd4 系统中第一个硬盘的第四分区

0x305 /dev/hd5 系统中第二个硬盘

0x306 /dev/hd6 系统中第二个硬盘的第一分区

0x307 /dev/hd7 系统中第二个硬盘的第二分区

0x308 /dev/hd8 系统中第二个硬盘的第三分区

0x309 /dev/hd9 系统中第二个硬盘的第四分区

bootsect 把自己搬运到 0x90000，并跳转

entry \_start

\_start:

mov ax,#BOOTSEG

mov ds,ax !ds = 0x07c0

mov ax,#INITSEG

mov es,ax !ex = 0x9000

mov cx,#256 !搬运256次

sub si,si !si = 0

sub di,di !di = 0

!ds:si=0x07c0:0x0, es:di=0x9000:0x0

rep

movw !每次搬运2个字节

jmpi go,INITSEG !跳转到 0x9000:go

以上代码表示把ds:si处(物理地址0x7c00)的内容搬运到es:di(物理地址0x90000),一共搬运512字节，即主引导扇区把自己移动到了0x90000处。

对于movw指令，可以参考我的博文。

http://blog.csdn.net/longintchar/article/details/50949923

我的疑问是，Linus为什么没有清除DF标志呢？是不是设置DF=0会更严谨呢？

jmpi go,INITSEG段间跳转，INITSEG是段地址，go是偏移地址。这句话执行完，CPU就一下子跑到了0x9000:go处执行了。（下图中左边的蓝色箭头，点击图片可放大）



跳转后继续执行下面的指令，设置ds,es,ss和sp.

go: mov ax,cs

mov ds,ax

mov es,ax !ds=es=cs=0x9000

mov ss,ax

mov sp,#0xFF00

!es:sp = 0x9000:0xff00 ，栈的设置

加载 setup 模块到 0x90200

load\_setup:

mov dx,#0x0000 ! 驱动器号(DL)0，磁头号(DH)0

mov cx,#0x0002 ! 起始扇区号2, 磁道号0

mov bx,#0x0200 ! 偏移地址0x200

mov ax,#0x0200+SETUPLEN ! 功能号AH=0x02,AL=要读的扇区数目=SETUPLEN=4

int 0x13 ! read it

jnc ok\_load\_setup ! ok - continue

mov dx,#0x0000 !需要复位的驱动器号=DL=0

mov ax,#0x0000 !功能号AH=0

int 0x13 ! 复位磁盘

j load\_setup

以上代码利用INT 13H, AH=02H把setup模块从磁盘（2~5扇区）加载到0x90200后面。

注意：柱面号和磁头号都从0开始，扇区号从1开始。

INT 13H AH=02H：读扇区

此功能从磁盘上把一个或更多的扇区内容读进内存。这是一个低级功能，在一个操作中读取的全部扇区必须在同一条磁道上。

参数 说明

入口参数

AH =02H ，指明调用读扇区功能。

AL 要读的扇区数目，不允许使用读磁道末端以外的数值，也不允许使该寄存器为0。

DL 需要进行读操作的驱动器号，0表示软盘，80H表示硬盘。

DH 所读磁盘的磁头号。

CH 磁道号的低8位数（磁道号共10位）。

CL 低5位放入所读起始扇区号，位7-6表示磁道号的高2位。

ES:BX 读出数据的缓冲区地址。

返回参数

CF =0，操作成功；=1，操作失败。

AH 错误返回码。

AL 实际读到的扇区数。

INT 13H AH=00H：磁盘控制器复位

此功能用于复位磁盘（软盘和硬盘）。当磁盘I/O功能调用出现错误时，需要调用此功能。

参数 说明

入口参数

AH =00H，指明调用复位磁盘功能。

DL 需要复位的驱动器号。软盘：00H-7FH；硬盘：80H-FFH

返回参数

CF =0，操作成功；=1，则操作失败

AH 错误返回码。

获得磁盘驱动器参数（主要是每磁道的扇区数量）

ok\_load\_setup:

! Get disk drive parameters, specifically nr of sectors/track

mov dl,#0x00 !驱动器号为0，说明是软盘

mov ax,#0x0800 ! AH=8 is get drive parameters

int 0x13

mov ch,#0x00 !这里用不上软盘的最大磁道号，可以使CH=0

seg cs !把段超越前缀设置为cs,只影响下一条语句

mov sectors,cx

!保存每磁道最大扇区数。对于软盘，最大磁道号不会超过256，所以CH足以表示，CL[7:6]为0

!以上两句可以写为 mov cs:[sectors], cx

mov ax,#INITSEG

mov es,ax !因为上面ES的值被修改，所以令ES=0x9000

INT 13H AH=08H：读取驱动器参数

参数 说明

入口参数

AH =08H，读取驱动器参数

DL 驱动器号（如果是硬盘则[7]=1）

返回参数

CF 0-操作成功；1-操作失败

AH 错误返回码

BL 驱动器类型

CH 最大磁道号的[7:0]

CL[7:6] 最大磁道号的[9:8]

CL[5:0] 每磁道最大扇区数

DH 最大磁头数

DL 驱动器数量

ES:DI 指向软驱磁盘参数表

打印 “Loading system …”

mov ah,#0x03 !读光标的位置

xor bh,bh !bh=页号

int 0x10

我们主要是用行号（DH中）和列号（DL中）。

INT 10H AH=03H：获取光标位置和形状

参数 说明

入口参数

AH =03H,读光标的位置

BH 页号

返回参数

CH 行扫描开始

CL 行扫描结束

DH 行号

DL 列号

INT 10H AH=13H：在Teletype模式下显示字符串

参数 说明

入口参数

AH ＝13H，在Teletype模式下显示字符串

BH 页码

BL 属性（若 AL=00H 或 01H）

CX 要显示的字符串的长度

DH、DL 坐标（行、列）

ES:BP 指向要显示的字符串

AL 显示输出方式

返回参数

无

对于显示输出方式，解释如下：

取值 说明 字符串格式

0 字符串中只含显示字符，显示属性在BL中；显示后，光标位置不变 char1，char2，……，charN

1 字符串中只含显示字符，显示属性在BL中；显示后，光标位置跟随字符串改变 char1，char2，……，charN

2 字符串中含有显示字符和显示属性；显示后，光标位置不变 char1，attri1,char2，attri2,……，charN,attriN

3 字符串中含有显示字符和显示属性；显示后，光标位置跟随字符串改变 char1，attri1,char2，attri2,……，charN,attriN

mov cx,#24 ! 24个字符

mov bx,#0x0007 ! page 0, attribute 7 (normal)

mov bp,#msg1

mov ax,#0x1301 ! write string, move cursor

int 0x10

msg1:

.byte 13,10

.ascii "Loading system ..."

.byte 13,10,13,10

13是回车，10是换行。它们的区别如下表。

回车和换行

中文名称 英文名称 字母简写 ASCII码 来源

回车 carriage return CR 0x0D=13D “车”指的是纸车，它带着纸向左移动。在开始打第一个字之前，要把纸车拉到最右边，使弹簧收紧。随着打字的进行，弹簧把纸车推向左边。把纸车拉到最右边，叫做“回车”。

换行 line feed LF 0x0A=10D 换行的概念是，打字机左边有个”把手”，扳动一下把手，纸就会上移一行。

加载 system 到 0x10000

! we want to load the system (at 0x10000)

mov ax,#SYSSEG ! SYSSEG=0x1000

mov es,ax ! segment of 0x010000

call read\_it

call kill\_motor

3~5行，把system模块加载到0x10000。

第6行，关闭驱动器马达。

过程read\_it

这个过程的功能是把还未读取的扇区加载到es:0x0000处。注意：es必须是0x1000的整数倍，否则会陷入死循环。每读64KB，都会使es的值增加0x1000，当es=0x4000的时候，停止读取。

sread: .word 1+SETUPLEN !当前磁道已经读取的扇区数, 前面的1表示引导扇区bootsect.s

head: .word 0 ! current head，当前磁头号

track: .word 0 ! current track，当前磁道号

read\_it:

mov ax,es

test ax,#0x0fff !使ax与0xfff按位与，测试es是否为0x1000的整数倍

die: jne die !结果不为0（说明es不是0x1000的整数倍）则陷入死循环

xor bx,bx ! bx（作为段内偏移地址）清零

rp\_read:

mov ax,es

cmp ax,#ENDSEG ! 实际上求(ax-ENDSEG)

jb ok1\_read ! 当CF=1（ax<ENDSEG, 有借位）时跳转到ok1\_read

ret ! 当ax>=ENDSEG时返回（我认为不会出现大于的情况）

ok1\_read:

seg cs

mov ax,sectors ! 这两句相当于 mov ax, cs:[sectors]; 获得每磁道扇区数

sub ax,sread ! ax = ax - sread, 得出本磁道未读扇区数

mov cx,ax

shl cx,#9 ! cx乘以512，求出字节数

add cx,bx ! 以上3行相当于 cx = ax \* 512 + bx

! 假设再读ax个扇区，cx就是段内共读入的字节数

jnc ok2\_read ! 若cx < 0x10000（CF=0,没有进位）则跳转到ok2\_read

je ok2\_read ! 若cx = 0（ZF=1），说明刚好读入64KB，则跳转到ok2\_read

xor ax,ax ! ax = 0x0000

sub ax,bx ! 求bx对0x10000的补数，结果在ax中

shr ax,#9 ! 除以512,得到扇区数，AL作为参数，传给read\_track

ok2\_read:

call read\_track ！调用read\_track过程，用AL传参,读取AL个扇区到ES:BX

mov cx,ax ！cx是该次操作已经读取的扇区数

add ax,sread ！ax是当前磁道已经读取的扇区数

seg cs

cmp ax,sectors

jne ok3\_read ！如果当前磁道还有扇区未读，跳转到ok3\_read

mov ax,#1 ！说明当前磁道的扇区都已读完

sub ax,head ！ax = 1 - 磁头号

jne ok4\_read ！不为0则跳转到 ok4\_read，说明磁头号为0

inc track ！说明磁头号为1，磁道号增加1

ok4\_read:

mov head,ax !更新磁头号（如果是37行跳转过来，则 head=1；否则 head=0）

xor ax,ax !ax=0, 因为更换了磁道，所以当前磁道已读扇区数置0

ok3\_read:

mov sread,ax ！更新当前磁道已经读取的扇区数

shl cx,#9

add bx,cx ！更新偏移地址

jnc rp\_read ！没有进位，则跳转到rp\_read

mov ax,es ！有进位，说明BX达到了64KB边界

add ax,#0x1000

mov es,ax ！es增加0x1000

xor bx,bx ！bx = 0

jmp rp\_read ！继续读取

以上汇编代码看起来实在是费劲。为了便于理解，写成C语言伪代码如下：

void read\_it(es)//参数是es

{

if((es & 0xFFF) != 0) //es 必须是0x1000的倍数，否则进入死循环

while(1); //dead loop

bx = 0;

while(es < ENDSEG){

// 1. 看看要读多少个扇区，用ax表示

// 2. sread:本磁道已经读取的扇区数

ax = SECTORS - sread;

if((ax \* 512 + bx) > 0x10000){

ax = (0x10000 - bx) / 512;

}

read\_track(ax); //调用读扇区过程,al:要读的扇区数，es:bx->缓冲区

cx = ax; //该次操作读取的扇区数

ax += sread; //ax是本磁道已读取的扇区总数

if(ax==SECTORS){

//本磁道的扇区全部读完

if(head == 1){ //0和1磁头都已经读完，更新磁道

++track;

head = 0; //从0磁头开始

}

else{

head = 1; //切换到1磁头

}

ax = 0; //本磁道已读扇区数置为0

}

sread = ax; //更新本磁道已读扇区数

bx += cx \* 512; 更新偏移地址bx

if(bx == 0x10000)

{

//如果偏移地址到达0x10000，则更新es,并使bx=0

es += 0x1000;

bx = 0;

}

}

return;

}

过程read\_track

读取AL个扇区到ES：BX。此过程的入口参数是：

AL-要读的扇区数目

ES:BX-缓冲区地址

read\_track:

push ax

push bx

push cx

push dx

mov dx,track ！当前磁道号

mov cx,sread ！已经读取的扇区数

inc cx ！CL是起始扇区号

mov ch,dl ！CH是磁道号----

mov dx,head ！当前磁头号

mov dh,dl ！DH是磁头号

mov dl,#0 ！DL是驱动器号，0表示软盘

and dx,#0x0100 ！DH是磁头号，不是0就是1

mov ah,#2 ！功能号2，读扇区

int 0x13

jc bad\_rt ！CF=1，表示出错，复位磁盘

pop dx

pop cx

pop bx

pop ax

ret

bad\_rt: mov ax,#0 ！AH=0，磁盘复位功能

mov dx,#0 ！DL=0，驱动器号

int 0x13

pop dx

pop cx

pop bx

pop ax

jmp read\_track ！重新读取

过程kill\_motor

kill\_motor:

push dx

mov dx,#0x3f2 !软盘控制器的端口-数字输出寄存器端口，只写

mov al,#0 !驱动器A，关闭FDC，禁止DMA和中断请求，关闭马达

outb !将al的值写入端口dx

pop dx

ret

DOR（数字输出寄存器）

DOR是一个8位寄存器，他控制驱动器马达的开启、驱动器选择、启动/复位FDC以及允许/禁止DMA及中断请求。

位 Name Description

7 MOT\_EN3 Driver D motor：1-start；0-stop

6 MOT\_EN2 Driver C motor：1-start；0-stop

5 MOT\_EN1 Driver B motor：1-start；0-stop

4 MOT\_EN0 Driver A motor：1-start；0-stop

3 DMA\_INT DMA and IRQs； 1 enable; 0-disable

2 RESET 0= enter reset mode；1= normal operation

1 and 0 DRV\_SEL1, DRV\_SEL0 “Select” drive number for next access

确认根文件系统设备号

seg cs

mov ax,root\_dev !ax = ROOT\_DEV

cmp ax,#0

jne root\_defined !如果 ROOT\_DEV 不等于0则跳转到 root\_defined

seg cs

mov bx,sectors ! 取每磁道扇区数

mov ax,#0x0208 ! /dev/ps0 - 1.2Mb

cmp bx,#15 ! 判断每磁道扇区数是否等于15

je root\_defined ! 说明是1.2MB的软盘

mov ax,#0x021c ! /dev/PS0 - 1.44Mb

cmp bx,#18 ! 判断每磁道扇区数是否等于18

je root\_defined ! 说明是1.44MB的软盘

undef\_root:

jmp undef\_root ! 死循环

root\_defined:

seg cs

mov root\_dev,ax ! 将检查过的设备号保存到 root\_dev 中

在Linux中软驱的主设备号是2，次设备号 = type \* 4 + nr.

其中，nr等于0~3时分别对应软驱A、B、C、D；type是软驱的类型，比如2表示1.2MB，7表示1.44MB等。

因为是可引导的驱动器，所以肯定是A驱。对于1.2MB，设备号 = 2 << 8 + 2 \* 4 + 0 = 0x208；对于1.44MB，设备号 = 2 << 8 + 7 \* 4 + 0 = 0x21C.

.org 508

root\_dev:

.word ROOT\_DEV ！这里存放根文件系统所在设备号（init/main.c中会用）

ROOT\_DEV到底有何用，怎么用，这里先存疑，后面再探究。

跳转到 setup 去执行

jmpi 0,SETUPSEG !到此本程序就结束了。

段间跳转，跳转到0x9020:0x0000（setup.s程序开始处）去执行。

代码分析到这里，就差不多明白了。虽然是一个引导扇区，编译后只有512字节，可是涉及的知识点还真不少。真是太佩服Linus了，一个大学生就能写出这样的代码，实属出众。

---------------------

作者：ARM的程序员敲着诗歌的梦

来源：CSDN

原文：https://blog.csdn.net/longintchar/article/details/79323783

版权声明：本文为博主原创文章，转载请附上博文链接！