Boot ----- 14040255 -------7c40 ---- mov ax,cs

Int 10 ---- 14040265

14042615

jmp loader ---- 15735412

18014853

18247597

18248093------dispmeminfo

18347832

18253120

18253208-----

18277230

18277838 ------ initkernel --------0008:0009038e

18342507

18342863------jmp kernel (0008:30400)-----0008:00090393

18342867 ------call cstart (0008:30718) ------0008:00030416

18346494 ----- ret

18346501---- jmp kernel\_main(0008:000307a6)------0008:00030439

18346506 -------call disp\_str(0008:00031430)----0008:000307b3

18346773 -------ret ------0008:0003146d

18347659 --- ret

18347660 ---call restart (0008:000306de)----0008:3098c

GDTR—

cr0

31 |30 |29 |。。。 |18 |17 |16 |。。。 |5 |4 |3 |2 |1 |0

PG |CD |NW | |AM| |WP | |NE |ET |TS |EM |MP |PE

PE=0时，CPU运行于实模式。

PE=1时，CPU运行于保护模式。

lgdt [GdtPtr]

GdtPtr:

BYTE5 BYTE4 BYTE3 BYTE2 |BYTE1 BYTE0

32位基地址 |16位界限

选择子(Selector)

BYTE1 |BYTE0

15 14 13 12 11 10 9 8 7 6 5 4 3 |2 |1 0

描述符索引 |TI |RPL

TI=0时，系统从当前的GDT中寻找描述符。

TI=1时，系统从当前的LDT中寻找描述符。

BYTE7 |BYTE6 BYTE5 |BYTE4 BYTE3 BYTE2 |BYTE1 BYTE0

段基址2 |属性 |段基址1 |段界限1

BYTE6 |BYTE5

7 |6 |5 |4 |3 2 1 0 |7 |6 5 |4 |3 2 1 0

G |D/B|0 |AVL|段界限 2 |P |DPL |S |TYPE

TYPE

3 2 1 0

0数据段 0 0 0

1代码段 1向下扩展 1写 1已访问

一致代码 读

3 2 1 0

0：286 0：TSS 0 0 0 - 《未定义》

0 0 0 1 1 - 可用286TSS

0 0 1 0 2 - LDT

0 0 1 1 3 - 忙的286TSS

0 1：门 0 0 4 - 286调用门

0 1 0 1 5 - 任务门

0 1 1 0 6 - 286中断门

0 1 1 1 7 - 286陷阱门

1：386 0：TSS 0 0 8 - 《未定义》

1 0 0 1 9 - 可用386TSS

1 0 1 0 A - 《未定义》

1 0 1 1 B - 忙的386TSS

1 1：门 0 0 C - 386调用门

1 1 0 1 D - 《未定义》

1 1 1 0 E - 386中断门

1 1 1 1 F - 386陷阱门

G --- 段界限颗粒度，G=0时段界限颗粒度为字节；

G=1 时段界限颗粒度为4KB.

D/B --- 在可执行代码段描述符中，------------------------------这一位叫D位：

D=0时，在默认情况下使用16位地址及16位或8位操作数。

D=1时，在默认情况下使用32位地址及32位或8位操作数。

在向下扩展数据段描述符中，---------------------------这一位叫B位：

B=0时，段的上部界限为64KB。

B=1时，段的上部界限为4GB。

在描述堆栈段(由ss寄存器指向的段)的描述符中，这一位叫D位：

D=0时，隐式的堆栈访问指令(push,pop,call)使用16位堆栈指针寄存器sp。

D=1时，隐式的堆栈访问指令(push,pop,call)使用16位堆栈指针寄存器sp。

AVL --- 保留

P --- P＝0时表示段在内存中不存在

P＝1时表示段在内存中存在

DPL --- 描述符特权级

S --- S＝0时表明描述符是系统段/门描述符

S＝1时表明描述符是数据段/代码段

调用门描述符：

BYTE7 BYTE6 |BYTE5 BYTE4 |BYTE3 BYTE2 |BYTE1 BYTE0

偏移 |属性 |选择子 |偏移

BYTE5 |BYTE4

7 |6 5 |4 |3 2 1 0 |7 |6 |5 |4 3 2 1 0

P |DPL |S |TYPE |0 |0 |0 |ParamCount

ELF

ELF头(ELF header)

程序头表(Program header table)

节(Sections)

节头表(Section header table)

ELF header

#define EI\_NIDENT

Typedef struct{

unsigned char e\_ident[EI\_NIDENT]; 0 16

ELF文件的字符，以及与机器无关的信息

Elf32\_Half e\_type; 16 2

文件的类型

Elf32\_Half e\_machine; 18 2

运行该程序需要的体系结构

Elf32\_word e\_version; 20 4

确定文件的版本

Elf32\_Addr e\_enrty; 24 4

程序的入口地址

Elf32\_off e\_phoff; 28 4

Program header table在文件中的偏移量

Elf32\_off e\_shoff; 32 4

Section header table 在文件中的偏移量

Elf32\_word e\_flags; 36 4

对IA32而言，此项为0.

Elf32\_Haif e\_ehsize; 40 2

ELF header 大小

Elf32\_Haif e\_phentsize; 42 2

Program header table中每一个条目的大小

Elf32\_Haif e\_phnum; 44 2

Program header table中有多少个条目

Elf32\_Haif e\_shentsize; 46 2

Section header table中每一个条目的大小

Elf32\_Haif e\_shnum; 48 2

Section header table 中有多少个条目

Elf32\_Haif e\_shstrndx; 50 2

包含节名称的字符串表是第几个节（从0开始数）

}Elf32\_Ehdr;

名称 大小 对齐 用途

Elf32\_Addr 4 4 无符号程序地址

Elf32\_Half 2 2 无符号中等大小整数

Elf32\_Off 4 4 无符号文件偏移

Elf32\_Sword 4 4 有符号大整数

Elf32\_Word 4 4 无符号大整数

Unsigned char 1 1 无符号小整数

Program header

Typedef struct{

Elf32\_Word p\_type; 0 4

当前Program header所描述的段的类型

Elf32\_Off p\_offset; 4 4

段的第一个字节在文件中的偏移

Elf32\_Addr p\_vaddr; 8 4

段的第一个字节在内存中的虚拟地址

Elf32\_Addr p\_paddr; 12 4

在物理地址定位相关的系统中，此项是为物理地址保留

Elf32\_Word p\_filesz; 16 4

段在文件中的长度

Elf32\_Word p\_memsz; 20 4

段在内存中的长度

Elf32\_Word p\_flags; 24 4

与段相关的标志

Elf32\_Word p\_align; 28 4

段在文件以及内存中如何对齐

}Elf32\_Phdr

Section header table

Typedef struct{

Elf32\_Word sh\_name; 0 4

Elf32\_Word sh\_type; 4 4

Elf32\_Word sh\_flags; 8 4

Elf32\_Addr sh\_addr; 12 4

Elf32\_Off sh\_offset; 16 4

Elf32\_Word sh\_size; 20 4

Elf32\_Word sh\_link; 24 4

Elf32\_Word sh\_info; 28 4

Elf32\_Word sh\_addralign; 32 4

Elf32\_Word sh\_entsize; 36 4

}Elf32\_Shdr

kernel.bin

ELF header

e\_ident[EI\_NIDENT];0 16 0000 0000 0000 0000 0001 0101 464c 457f

ELF文件的字符，以及与机器无关的信息

e\_type; 16 2 0002

文件的类型

e\_machine; 18 2 0003

运行该程序需要的体系结构

e\_version; 20 4 0000 0001

确定文件的版本

e\_enrty; 24 4 0003 0400

程序的入口地址

e\_phoff; 28 4 0000 0034

Program header table在文件中的偏移量

e\_shoff; 32 4 0000 20f8

Section header table 在文件中的偏移量

e\_flags; 36 4 0000 0000

对IA32而言，此项为0.

e\_ehsize; 40 2 0034

ELF header 大小

e\_phentsize; 42 2 0020

Program header table中每一个条目的大小

e\_phnum; 44 2 0003

Program header table中有多少个条目

e\_shentsize; 46 2 0028

Section header table中每一个条目的大小

e\_shnum; 48 2 0008

Section header table 中有多少个条目

e\_shstrndx; 50 2 0007

包含节名称的字符串表是第几个节（从0开始数）

Program header table 1 0x0034 ~ 0x0053

p\_type; 0 4 0000 0001

当前Program header所描述的段的类型

p\_offset; 4 4 0000 0000

段的第一个字节在文件中的偏移

p\_vaddr; 8 4 0003 0000

段的第一个字节在内存中的虚拟地址

p\_paddr; 12 4 0003 0000

在物理地址定位相关的系统中，此项是为物理地址保留

p\_filesz; 16 4 0000 1f18

段在文件中的长度

p\_memsz; 20 4 0000 1f18

段在内存中的长度

p\_flags; 24 4 0000 0005

与段相关的标志

p\_align; 28 4 0000 1000

段在文件以及内存中如何对齐

Program header table 2 0x0054 ~ 0x0073

p\_type; 0 4 0000 0001

当前Program header所描述的段的类型

p\_offset; 4 4 0000 2000

段的第一个字节在文件中的偏移

p\_vaddr; 8 4 0003 2000

段的第一个字节在内存中的虚拟地址

p\_paddr; 12 4 0003 2000

在物理地址定位相关的系统中，此项是为物理地址保留

p\_filesz; 16 4 0000 009c

段在文件中的长度

p\_memsz; 20 4 0001 9748

段在内存中的长度

p\_flags; 24 4 0000 0006

与段相关的标志

p\_align; 28 4 0000 1000

段在文件以及内存中如何对齐

Program header table 3 0x0074 ~ 0x0093

p\_type; 0 4 6474 e551

当前Program header所描述的段的类型

p\_offset; 4 4 0000 0000

段的第一个字节在文件中的偏移

p\_vaddr; 8 4 0000 0000

段的第一个字节在内存中的虚拟地址

p\_paddr; 12 4 0000 0000

在物理地址定位相关的系统中，此项是为物理地址保留

p\_filesz; 16 4 0000 0000

段在文件中的长度

p\_memsz; 20 4 0000 0000

段在内存中的长度

p\_flags; 24 4 0000 0007

与段相关的标志

p\_align; 28 4 0000 0010

段在文件以及内存中如何对齐

Section header table1

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 0000

Elf32\_Word sh\_type; 4 4 0000 0000

Elf32\_Word sh\_flags; 8 4 0000 0000

Elf32\_Addr sh\_addr; 12 4 0000 0000

Elf32\_Off sh\_offset; 16 4 0000 0000

Elf32\_Word sh\_size; 20 4 0000 0000

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0000

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Section header table2

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 000b .text

Elf32\_Word sh\_type; 4 4 0000 0001

Elf32\_Word sh\_flags; 8 4 0000 0006

Elf32\_Addr sh\_addr; 12 4 0003 0400

Elf32\_Off sh\_offset; 16 4 0000 0400

Elf32\_Word sh\_size; 20 4 0000 1292

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0010

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Section header table3

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 0011 .rodata

Elf32\_Word sh\_type; 4 4 0000 0001

Elf32\_Word sh\_flags; 8 4 0000 0002

Elf32\_Addr sh\_addr; 12 4 0003 16a0

Elf32\_Off sh\_offset; 16 4 0000 16a0

Elf32\_Word sh\_size; 20 4 0000 05e0

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0020

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Section header table4

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 0019 .eh\_frame

Elf32\_Word sh\_type; 4 4 0000 0001

Elf32\_Word sh\_flags; 8 4 0000 0002

Elf32\_Addr sh\_addr; 12 4 0003 1c80

Elf32\_Off sh\_offset; 16 4 0000 1c80

Elf32\_Word sh\_size; 20 4 0000 0298

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0004

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Section header table5

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 0023 .data

Elf32\_Word sh\_type; 4 4 0000 0001

Elf32\_Word sh\_flags; 8 4 0000 0003

Elf32\_Addr sh\_addr; 12 4 0003 2000

Elf32\_Off sh\_offset; 16 4 0000 2000

Elf32\_Word sh\_size; 20 4 0000 009c

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0020

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Section header table6

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 0029 .bss

Elf32\_Word sh\_type; 4 4 0000 0008

Elf32\_Word sh\_flags; 8 4 0000 0003

Elf32\_Addr sh\_addr; 12 4 0003 20a0

Elf32\_Off sh\_offset; 16 4 0000 209c

Elf32\_Word sh\_size; 20 4 0001 96a8

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0020

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Section header table7

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 002e .comment

Elf32\_Word sh\_type; 4 4 0000 0001

Elf32\_Word sh\_flags; 8 4 0000 0030

Elf32\_Addr sh\_addr; 12 4 0000 0000

Elf32\_Off sh\_offset; 16 4 0000 209c

Elf32\_Word sh\_size; 20 4 0000 0024

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0001

Elf32\_Word sh\_entsize; 36 4 0000 0001

}Elf32\_Shdr

Section header table8

Typedef struct{

Elf32\_Word sh\_name; 0 4 0000 0001 .shstrtab

Elf32\_Word sh\_type; 4 4 0000 0003

Elf32\_Word sh\_flags; 8 4 0000 0000

Elf32\_Addr sh\_addr; 12 4 0000 0000

Elf32\_Off sh\_offset; 16 4 0000 20c0

Elf32\_Word sh\_size; 20 4 0000 0037

Elf32\_Word sh\_link; 24 4 0000 0000

Elf32\_Word sh\_info; 28 4 0000 0000

Elf32\_Word sh\_addralign; 32 4 0000 0001

Elf32\_Word sh\_entsize; 36 4 0000 0000

}Elf32\_Shdr

Gdtr ---------0009013d --------1f

31

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

BASE:00 00 00 00

LIMIT:0 00 00

FF FF 00 00 00 9B CF 00

00 CF 9B 00 00 00 FF FF

BASE:00 00 00 00

LIMIT:F FF FF

FF FF 00 00 00 93 CF 00

00 CF 93 00 00 00 FF FF

BASE:00 00 00 00

LIMIT:F FF FF

FF FF 00 80 0B F3 00 00

00 00 F3 0B 80 00 FF FF

BASE:00 0B 80 00

LIMIT:0 FF FF

1F 00 3D 01 09 00

00 09 01 3D 00 1F

BASE: 0X0009013D

LIMIT: 001F