# 启动

切换到保护之后无法使用系统调用

引导扇区512B会将core.bin部分拷贝到IMGCORE\_LOADADDR\_BASE: IMGCORE\_LOADADDR\_OFFSET的上，注意由于16模式下所以最终地址实际是IMGCORE\_LOADADDR\_BASE\*16+IMGCORE\_LOADADDR\_OFFSET的位置。然后跳转到该处开始执行，这个位置放置了一段搬移自身到0地址、跳转过去开始执行并切换到保护模式的汇编代码。

# 内存分配

## 段描述符表

通过一个特定的段.gdt.data来存放全局描述符表，同时在链接脚本中定义一个符号指向这个地址，后续C代码中通过该符号操作gdt

通过一个特定的段.idt.data来存放全局描述符表，同时在链接脚本中定义一个符号指向这个地址，后续C代码中通过该符号操作idt

# 附录

## 低端内存空间分布

|  |  |
| --- | --- |
| From:0xA0000 |  |
| From:0x9A000 | Resered for BIOS EBDA. |
| From:0x98000 | Command Line |
| From:0x90200 |  |
| From:0x90000 |  |
| From:0x10000 |  |
| From:0x01000 | Boot loader. |
| From:0x00800 | Reserved for MBR/BIOS. |
| From:0x00600 |  |
| From:0x00000 | Bios use only. |

图表 1 低端内存分布图

## Grub2的实现参考

以下描述均是针对grub2.00版本的代码。

### 启动代码

$(GRUB\_SRC)/grub-core/boot/i386/pc/boot.S

// 该文件生成的boot.img被存储在0头0道1扇区

$(GRUB\_SRC)/grub-core/boot/i386/pc/diskboot.S

// 该文件生成的diskboot.img被存储在0头0道2扇区

boot.img加载diskboot.img并跳转至那里，由diskboot.img将kernel.img加载进来执行，它是grub2的主要部分。

## 反汇编实模式x86

objdump –d core.elf –M i8086

## 镜像文件

制作hd镜像，作为bochs运行的硬盘。这里需要注意的时，如果想镜像文件作为一个普通硬盘设备在host系统中使用(例如分区，文件系统制作，挂载写入等等操作)，需要借助loop文件这种抽象设备，在linux中使用losetup来将一个文件映射到一个loop设备。

# 可以先用-f参数找到一个空闲的loop虚拟磁盘文件

losetup –f

# 使用该命令可以将sys.img文件和一个loop虚拟磁盘建立映射

losetup /dev/loop0 sys.img

# 使用该命令可以将已经建立映射的loop虚拟磁盘脱离

losetup -d /dev/loop0

对于建立映射的loop文件，可以当作普通来处理了，比如使用fdisk来为其分区(操作之后就会在head0 cyl0 sector1上的0x1BE偏移上写磁盘分区列表了)，然后可以在分区上建立文件系统。这里需要使用一个工具kpartx，该工具将其中虚拟文件中的分区映射到/dev/mapper/目录下，作为一个设备(例如/dev/mapper/loop0p1这个文件，就是通过kpartx工具建立的)

# 使用该命令可以将loop0中的子分区映射成/dev/mapper目录下的单独文

# 件，这样就可以直接使用该文件来制作文件系统或者挂载操作

kpartx -a /dev/loop0

# 使用该命令可以将/dev/mapper目录下子分区映射文件解除

Kpartx –d /dev/loop0

通过kpartx在/dev/mapping/下的建立的分区映射文件(loop0p1)是一个link，这个link映射到/dev下的一个块设备文件，这个块设备可以直接用来制作文件系统(mkfs.ext4命令可以直接建立ext4文件系统)，也直接挂载到某个路径。

注意到可以使用virtualbox的工具将一个raw格式的镜像文件转换成virtualbox可以使用的磁盘文件\*.vdi，创建虚拟机使用该文件即可启动了。可以可以使用如下的命令创建磁盘文件：

vboxmanage convertfromraw sys.img sys.vdi -format VDI