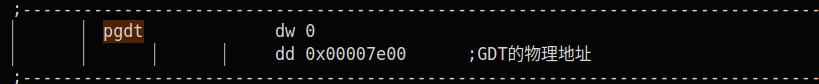
code13-1

主引导程序。首先从下图看到内存布局

图示

AI 生成的内容可能不正确。

pgdt记录了GDT的物理地址，这里将GDT放在0x7e00，描述符界限初始为0。



由于当前仍然处在实模式下，所以ds中的值应当是0x7e0，段内起始偏移地址为0。

之后按下图内容创建保护模式下的描述符，注意这里每个描述符占用了8个字节。

表格

AI 生成的内容可能不正确。

这里再将描述符的分布图列出来

图片包含 表格

AI 生成的内容可能不正确。

其中关注bit23，为1表示以4KB为粒度。

在初始化完4个描述符之后，计算出描述符表界限为8\*5-1=39，回填到pgdt中。注意这里标号pgdt指向的内存区域是主引导程序内。之后使用ldgt加载描述符表。

进入到保护模式之后，首先加载数据段选择子和堆栈段选择子，初始化堆栈指针为0。

屏幕上有字

AI 生成的内容可能不正确。

之后通过read\_hard\_disk\_0读取扇区内容。每次都将ebx寄存器内容增加512，目的是指向下一个内存块。

先调用一次read\_hrad\_disk\_0读取第一个扇区，从edi也就是0x40000，即系统核心程序的起始位置读出程序大小，除以512。

1. 用系统核心程序的大小除以512，如果刚好为512的倍数，放在edx的余数为0，从eax的倍数减去1(因为已经读取了一个扇区)。然后继续读取剩余的扇区直到读完整个内核。
2. 用系统核心程序的大小除以512，发现没有除尽，即放在edx的余数不为0，之后跳转@1，首先考虑到程序大小不满足512字节的情况(也就是eax为0)，这时候直接跳转setup加载程序即可
3. 如果(2)中发现程序大小大于512字节，并且还有要读的内容，此时正常继续读取即可，继续读取的次数就是eax中记录的商(因为余数的存在，不用-1)，之后循环读完即可。

屏幕上有字

AI 生成的内容可能不正确。

最后进入到setup，通过数据段选择子，确保能够访问整个4GB空间。由于DS描述符高速缓存器中的32为线性基地址为0x0，mov esi,[0x7c00+pgdt+0x02]指令将esi赋值为pgdt中的0x40000地址。之后就是通过访问系统核心程序的各段汇编地址构造对应的段描述符，这里封装了make\_gdt\_descriptor接口，最后再将新的GDT界限8\*8-1=63写回到pgdt指示的描述符表界限中。

表格

AI 生成的内容可能不正确。

之后跳转code13-2中偏移0x10指示的核心代码段入口点执行系统核心程序。从531行开始执行，其中涉及到公共例程段以及cpuid指令，cpuid指令用于返回处理器的标识和特性信息。cpuid指令执行后，处理器将返回的信息放在EAX/EBX/ECX/EDX中。之后要开始用户程序的加载和重定位。

文本

AI 生成的内容可能不正确。

先跳转code13-3看用户程序的结构，首先是文件头

图形用户界面, 文本

AI 生成的内容可能不正确。

在文件头中，偏移0处是一个双字，指示了用户程序的大小，以字节为单位。

偏移0x4的双字是头部的长度，以字节为单位。

偏移0x8的双字是为栈保留，内核分配了栈空间之后，会把栈段的选择子写到这里。

偏移0xc的双字是要求分配的栈大小，以4KB为单位，填1要求4KB，填2要求8KB，以此类推。

偏移0x10的双字是用户程序入口点的32位偏移地址。

偏移0x14的双字，是用户代码段起始汇编地址。内核完成对用户程序的加载和重定位之后，将把该段的选择子回填到这里(仅占用低字部分)。这样和0x10就组成了一个6字节的入口点，内核从这里转移控制到用户程序。

偏移0x18的双字是用户代码段长度。

偏移0x1c的双字是用户数据段的起始汇编地址。内核完成用户程序的加载和重定位之后，将把该段的选择子回填到这里(仅占用低字部分)。

偏移0x20的双字是数据段的长度。

操作系统提供的编程接口就是API，这是一大堆例程，需要的时候直接调用即可。

在code13-3中，内核要求，用户程序必须在偏移0x28的地方构造一个表格，并在表格中列出要用到的符号名(在高级语言里就是库函数名)。每个符号名长度是256字节，不足用0x0填充。用户程序加载之后，内核会分析这个表格，并将每一个符号名替换成相应的内存地址，这就是过程的重定位。作者在这里将该表格称为“符号-地址检索表”(Symbol-Address Lookup Table， SALT)。

之后用times 256-($-PrintString)表达式计算出伪指令db的重复次数，这里$代表当前行的汇编地址。