问题清单二

2019 年 11 月 10 日
22:29

1. 什么是实模式，什么是保护模式?

（1）早期 8088CPU 时期，有 20 位的地址线，8 个 16 位通用寄存器和 4 个 16 位段寄存器。实模式就是用基地址加偏移量就可以直接拿到物理地址的模式。它通常需要用下面的这种格式来表示：(段基址：段偏移量)

其中第一个字段是段基址，它的值是由段寄存器提供的(一般来说，段寄存器有 6 种，分别为 cs，ds，ss，es，fs，gs，这几种段寄存器都有自己的特殊意义，这里不做介绍)。

第二字段是段内偏移量，代表你要访问的这个内存地址距离这个段基址的偏移。它的值就是由通用寄存器来提供的，所以也是 16 位。那么两个 16 位的值如何组合成一个 20 位的地址呢？CPU 采用的方式是把段寄存器所提供的段基址先向左移 4 位。这样就变成了一个 20 位的值，然后再与段偏移量相加。

即：

物理地址 = 段基址<<4 + 段内偏移

缺点：实模式非常不安全。没有内存访问保护（想读就读，想写就写）；无法支持多任务（一切都是真实地址）

（2）保护模式就是不能直接拿到物理地址的模式。

需要进行地址转换。保护模式是在 80286 时提出，在 80386 上大成。具体就是 80286 提了的段式内存管理和 80386 提出的页式内存管理。所以所谓的保护物理内存就是段页式内存管理方式。

使用了段式内存管理之后，cpu 使用的就是逻辑地址，要经过**段描述符表**翻译才能访问实际的物理地址，而段描述符表只有系统内核才能修改。这就保证了一个进程只能访问内核分配给他的段上的物理内存。

是现代操作系统的主要模式

2. 什么是选择子?

选择子是描述符在描述符表的相对偏移，值在段寄存器里。

（1）选择子共 16 位，放在段选择寄存器里

（2）低 2 位是 RPL，表示请求特权级，即以什么样的权限去访问段

（3）第 3 位表示选择 GDT 方式还是 LDT 方式，其中 TI=0 找 GDT 表，TI=1 找 LDT 表

（4）高 13 位表示在描述符表中的偏移（故描述符表的项数最多是 2 的 13 次方），即索引号，

3. 什么是描述符?

全局描述符表中含有一个个表项，每一个表项称为段描述符。而段寄存器在保护模式下存放的便是相当于一个数组索引的东西，通过这个索引，可以找到对应的表项。段描述符存放了段基址、段界限、内存段类型属性(比如是数据段还是代码段,注意一个段描述符只能用来定义一个内存段)等许多属性,具体信息见下图：

BASE: 段基址，由上图中的两部分(BASE 31-24 和 BASE 23-16)组成

G：LIMIT的单位，该位 0表示单位是字节，1表示单位是 4KB

D/B: 该位为 0 表示这是一个 16 位的段，1 表示这是一个 32 位段

AVL: 该位是用户位，可以被用户自由使用

LIMIT: 段的界限，单位由 G 位决定。数值上（经过单位换算后的值）等于段的长度（字节）- 1

P: 段存在位，该位为 0 表示该段不存在，为 1 表示存在

DPL：段权限

S: 该位为 1 表示这是一个数据段或者代码段。为 0 表示这是一个系统段（比如调用门，中断门等）

TYPE: 根据 S 位的结果，再次对段类型进行细分

其中，段界限表示段边界的扩张最值，即最大扩展多少或最小扩展多少，用 20 位来表示，它的单位可以是字节，也可以是 4KB，这是由 G 位决定的(G 为 1 时表示单位为 4KB)。

实际段界限边界值=(描述符中的段界限+1)*（段界限的单位大小(即字节或 4KB))-1，如果偏移地址超过了段界限，CPU 会抛出异常。

全局描述符表位于内存中，需要用专门的寄存器指向它后， CPU 才知道它在哪里。这个专门的寄存器便是 GDTR(一个 48 位的寄存器),专门用来存储 GDT 的内存地址及大小。

4. 什么是 GDT，什么是 LDT?
(1)GDT：全局描述符表，是全局唯一的。存放一些公用的描述符、和包含各进程局部描述符表首地址的描述符。另外，每一个 LDT 自身作为一个段存在，它们的段描述符被放在 GDT 中。
Global Descriptor Table,是一张存放 Descriptor 的表，可在全局
内访问,所有进程想要访问全局可见的段时，从 GDT 查询，有且只有一个。
进程从 GDTR 寄存器中获得 GDT 的位置，向它发起查询。具体见 3
(2)LDT：局部描述符表，每个进程都可以有一个。存放本进程内使用的描述符。
LDT(Local)与 GDT 相同，但是不是全局的，对于某个进程，它只知
道它自己的 LDT。每个进程有自己的 LDT，访问自己的段时从 LDT 查询。
进程从 LDTR 寄存器中获得 LDT 的位置，向它发起查询

5. 请分别说明 GDTR 和 LDTR 的结构。
(1)GDTR：48 位寄存器，高 32 位放置 GDT 首地址，低 16 位放置 GDT 限长（限长决定了可寻址的大小即表的大小，注意低 16 位放的不是选择子）
(2)LDTR：16 位寄存器，放置一个特殊的选择子，用于查找当前进程的 LDT 首地址。

6. 请说明 GDT 直接查找物理地址的具体步骤。
（1）给出段选择子（放在段选择寄存器里）+偏移量
（2）若选择了 GDT 方式，则从 GDTR 获取 GDT 首地址，用段选择子中的 13 位做偏移，拿到 GDT 中的描述符
（3）如果合法且有权限，用描述符中的段首地址加上（1）中的偏移量找到物理地址。寻址结束。
7. 请说明通过 LDT 查找物理地址的具体步骤。
（1）给出段选择子（放在段选择寄存器中）+偏移量

（2）若选择了 LDT 方式，则从 GDTR 获取 GDT 首地址，用 LDTR 中的偏移量做偏移，拿到 GDT 中的描述符 1
（3）从描述符 1 中获取 LDT 首地址，用段选择子中的 13 位做偏移，拿到 LDT 中的描述符 2
（4）如果合法且有权限，用描述符 2 中的段首地址加上（1）中的偏移量找到物理地址。寻址结束。

8. 根目录区大小一定么？扇区号是多少？为什么？

不一定，通过 BPB_RootEntCnt（根目录文件数的最大值，比如 0xE0）来计算根目录的大小，具体计算公式是

根目录区占用扇区数 = RootDirSectors = (BPB_RootEntCnt*0x20(一个目录项占 0x20 个字节))/BPB_BytesPerSec（一个扇区的字节数，比如 0x200）

典型值：19

不一定。扇区号=1+9*2=19
根目录条目不一定。引导分区占一个扇区，每个 fat 占 9 个扇区

9. 数据区第一个簇号是多少？为什么？
数据区就起始于簇 2。
在 1.44M 软盘上，FAT 前三个字节的值必须是固定的，分别是 0xF0、0xFF、0xFF，用于表示这是一个应用在 1.44M 软盘上的 FAT12 文件系统。本来序号为 0 和 1 的 FAT 表项应该对应于簇 0 和簇 1，但是由于这两个表项被设置成了固定值，簇 0 和簇 1 就没有存在的意义了。

10. FAT 表的作用？
文件分配表被划分为紧密排列的若干个表项，每个表项都与数据区中的一个簇相对应，而且表项的序号也是与簇号一一对应的。用于寻找下一个簇号

11. 解释静态链接的过程。
静态链接是由链接器在链接时将库的内容加入到可执行程序中的做法。
在编译 main.c 的时候，编译器还不知道 printf 函数的地址，所以在编译阶段只是将一个“临时地址”放到目标文件中，在链接阶段，这个“临时地址”将被修正为正确的地址，这个过程叫重定位。所以链接器还要知道该目标文件中哪些符号需要重定位，这些信息是放在了重定位表中。
在链接的时候，我们需要告诉链接器需要链接的目标文件和库文件（默认 gcc 会把标准库作为链接器输入的一部分）。链接器会根据输入的目标文

件从库文件中提取需要目标文件。
知道了这些信息后，链接器就可以开始工作了，分为两个步骤：1）合并相似段，把所有需要链接的目标文件的相似段放在可执行文件的对应段中。2）重定位符号使得目标文件能正确调用到其他目标文件提供的函数。

12. 解释动态链接的过程。
所谓动态链接就是在运行的时候再去链接。
从动态库的角度来看，动态库像普通的可执行文件一样，有其代码段和数据段。为了使得动态库在内存中只有一份，需要做到不管动态库装载到什么位置，都不需要修改动态库中代码段的内容，从而实现动态库中代码段的共享。而数据段中的内容需要做到进程间的隔离，因此必须是私有的，也就是每个进程都有一份。因此，动态库的做法是把代码段中变化的部分放到数据段中去，这样代码段中剩下的就是不变的内容，就可以装载到虚拟内存的任何位置。那代码段中变化的内容是什么，主要包括了对外部函数和变量的引用。
动态库是把地址相关的内容放到了数据段中来实现地址无关的代码，从而使得动态库能被多个进程共享。
动态链接生成的可执行文件运行前，系统会首先将动态链接库加载到内存中。当所有的库都被加载进来以后，类似于静态链接，动态链接器从各个动态库中可以知道每个库都提供什么函数（符号表）和哪些函数引用需要重定位（重定位表），然后修正.got 和.got.plt 中的符号到正确的地址，完成之后就可以将控制权交给可执行文件的入口地址，从而开始执行我们编写的代码了。

13. 静态链接相关 PPT 中为什么使用 ld 链接而不是 gcc
用 gcc 的话有可能去调 C 库，使程序环境变得复杂，所以用 ld
gcc 会自动链接操作系统的库，有时我们并不想这么操作。ld 就老实多了，只会链接参数中所规定的。

14. linux 下可执行文件的虚拟地址空间默认从哪里开始分配。
linux 下，ELF 可执行文件默认从地址 0x08048000 开始分配