1.实模式下的寻址⽅式以及实模式的缺陷

实模式下，段在内存中固定的位置（物理地址 = 段值\*16 + 偏移 ）；通过改变段寄存器的值，我们可以随心所欲的访问内存任何一个单元，而丝毫不受到限制，不能对内存访问加以限制，也就谈不上对系统的保护。因此，在实模式下是无法构造现代意义的操作系统的。

内存中每个字节的地址能由不止一个的段基址加偏移表示，分段地址之间的比较将复杂化

2. 保护模式下的寻址过程：  
• 段寄存器中存储的是什么?GDT 是什么?LDT 是什么? 如何区分  
LDT 和 GDT? LDT 和 GDT 的区别是什么? 如何定位到 Descriptor?Descriptor 的内容有哪些?  
• GDTR 中的内容是什么?LDTR 中存储的是什么? 为什么 LDT 要放  
在 GDT 中?

1. 根据指令的性质来确定使用哪个段寄存器，例如，放在转移指令中 的地址在代码段，而取数据的指令中的地址在数据段；
2. 根据段寄存器的内容(选择子)，首先判断描述符是在GDT中还是在 LDT中，如果是在GDT中，根据GDTR以及该段寄存器的内容找到相应 的“描述符”；如果是在LDT中，根据LDTR（选择子）以及GDTR的 内容找到LDT的描述符，得到LDT的地址，然后再根据段寄存器内容 找到相应的“描述符”；
3. 从描述符中得到基地址；
4. 将指令中发出的地址作为位移，与描述符中界限相比，看是否越界；
5. 将指令的性质与描述符中的访问权限来确定是否越权；
6. 将指令中发出的地址作为位移，与基地址相加得出实际的“物理地 址”。

段寄存器中是一个选择子

* GDT，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。

LDT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

* 所有的描述符均为8个字节（0-7)，由第5个字节说明描述符的类型，类型不同，描述符的结构也有所不同。
* 内容有段的起止地址、界限、属性
* GDTR用于表示GDT在内存中的段地址和段限，是一个48位的寄存器，其中32位表示段地址，16位表示段限（最大 64K，每个描述符8字节，故最多有64K/8=8K个描述符）
* LDTR用于表示LDT在内存中的位置， LDT本身也是一种数据段，它必须有一 个描述符，且该描述符必须放在GDT中，因此LDTR使用了与DS、ES、CS等相同的机制，其中只存放一个“选择子”，通过查GDT表获得LDT的真正内存地址
* GDT表只有一个，是固定的；而LDT表每个任务就可以有一个，因此有多个，并且由于任务的个数在不断变化其数量也在不断变化。

如果只有一个LDTR寄存器显然不能满足多个LDT的要求。因此intel的做法是把它放在GDT中。

3. 选择⼦的作⽤：  
• 选择⼦是什么？它的值存放在哪⾥？• 选择⼦⾥⾯的内容有哪些？  
• 为什么偏移地址⼤⼩是 13 位？

将段寄存器与段联系起来，用于找到所对应的描述符在描述符表中的位置

选择子是一个2字节的数，共16位，最低2位表示RPL（请求特权等级），第3位表示查表是利用GDT（全局描述符表）还是LDT（局部描述符表）进行，最高13位给出了 所需的描述符在描述符表中的地址。

13位正好足够寻址8K项

4. 描述符的作⽤：

描述各种数据段，给出基址进行地址转换，以及该段的范围防止越界，起到保护的作用

5. GDTR/LDTR 的作⽤：

用于指示GDT/LDT在内存中的位置。

GDTR 的内容是什么？

表示GDT在内存中的段地址和段限，是一个48位的寄存器，其中32位表示段地址，16位表示段限（偏移量）

LDTR 的内容是什么？

表示LDT在内存中的位置， LDT本身也是一种数据段，它必须有一个描述符，且该描述符必须放在GDT中，因此LDTR使用了与DS、ES、CS等相同的机制，其中只存放一个“选择子”，通过查GDT表获得LDT的真正内存地址

6. 根⽬录区⼤⼩⼀定么? 扇区号是多少? 为什么？

不一定，19，之前为引导扇区fat1、fat2的扇区

根目录大小为：BPB\_RootEntCnt（根目录文件最大数）\*32字节

7. 数据区第⼀个簇号是多少? 为什么?

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，所以数据区就起始于簇2

8. FAT 表的作用?

FAT项的值代表文件的下一个簇号

值大于或等于0xFF8，表示当前簇已经是本文件的最后一个簇

值为0xFF7，表示它是一个坏簇

9. 解释静态链接的过程。

链接器在链接的时候，会根据所引用的符号自动去相应的模块查找函数的地址，然后将主程序模块中所有引用到函数的指令重新修正，让它们的目标地址为真正的函数的地址。在编译链接时直接将需要的执行代码拷贝到调用处、、

相似段合并，重定位

缺点：使用静态链接生成的可执行文件体积较大，包含相同的公共代码，造成浪费

10. 解释动态链接的过程。

只建立一个引用的接口，而真正的代码和数据存放在另外的可执行模块中，在运行时再装入。使用这种方式的程序并不在一开始就完成动态链接，而是直到真正调用动态库代码时，载入程序才计算(被调用的那部分)动态代码的逻辑地址。

动态链接器自举，装载共享对象，重定位和初始化

缺点：这种方式使程序初始化时间较短，但运行期间的性能比不上静态链接的程序

11. 静态链接相关 PPT 中为什么使用 ld 链接而不是 gcc。

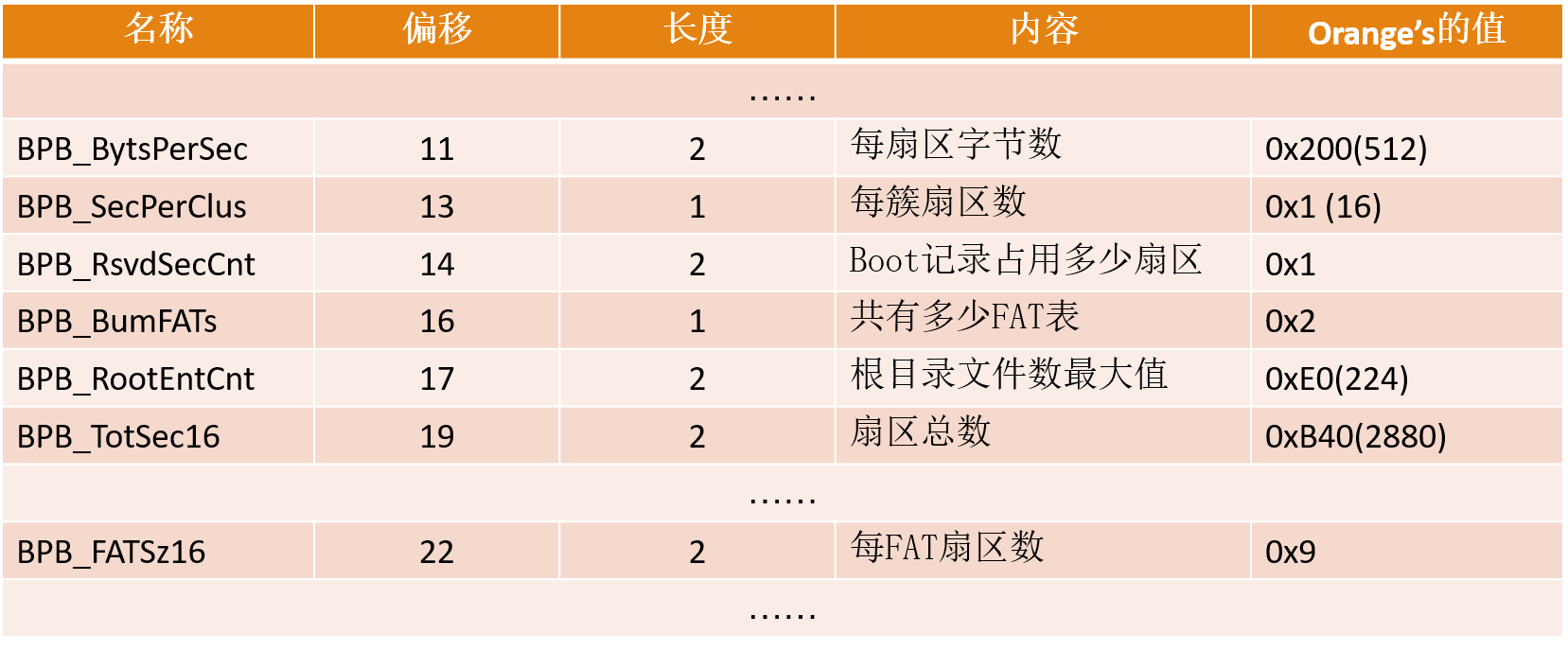
gcc叫做gcc工具链，这个工具链包括很多工具，其中用于链接的叫做ld.用 gcc链接时会根据选项不同调用不同的工具，如果到了链接阶段就会调用ld这个工具，，就是说gcc是工具的入口，实际上执行连接的还是ld

12. linux 下可执行文件的虚拟地址空间默认从哪里开始分配。

进程从0x08048000开始

2.2 实验相关内容

1. BPB 指定字段的含义



2. 如何进入子目录并输出 (说明方法调用)

3. 如何获得指定文件的内容, 即如何获得数据区的内容 (比如使用指针等)

4. 如何进行C 代码和汇编之间的参数传递和返回值传递

栈

5. 汇编代码中对 I/O 的处理方式, 说明指定寄存器所存值的含义

6. 可以要求解释某些看不懂的代码 (我看不懂的话