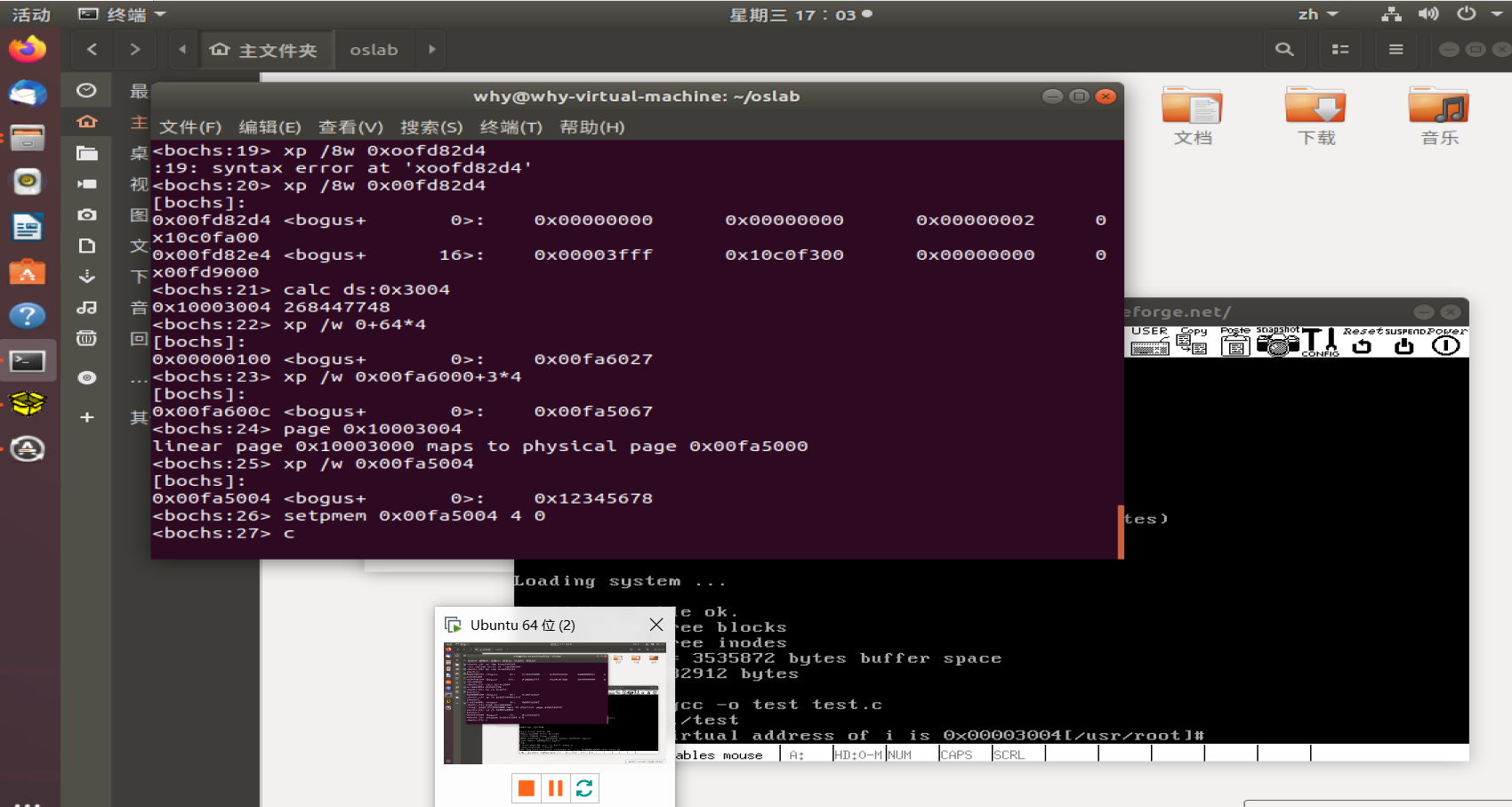
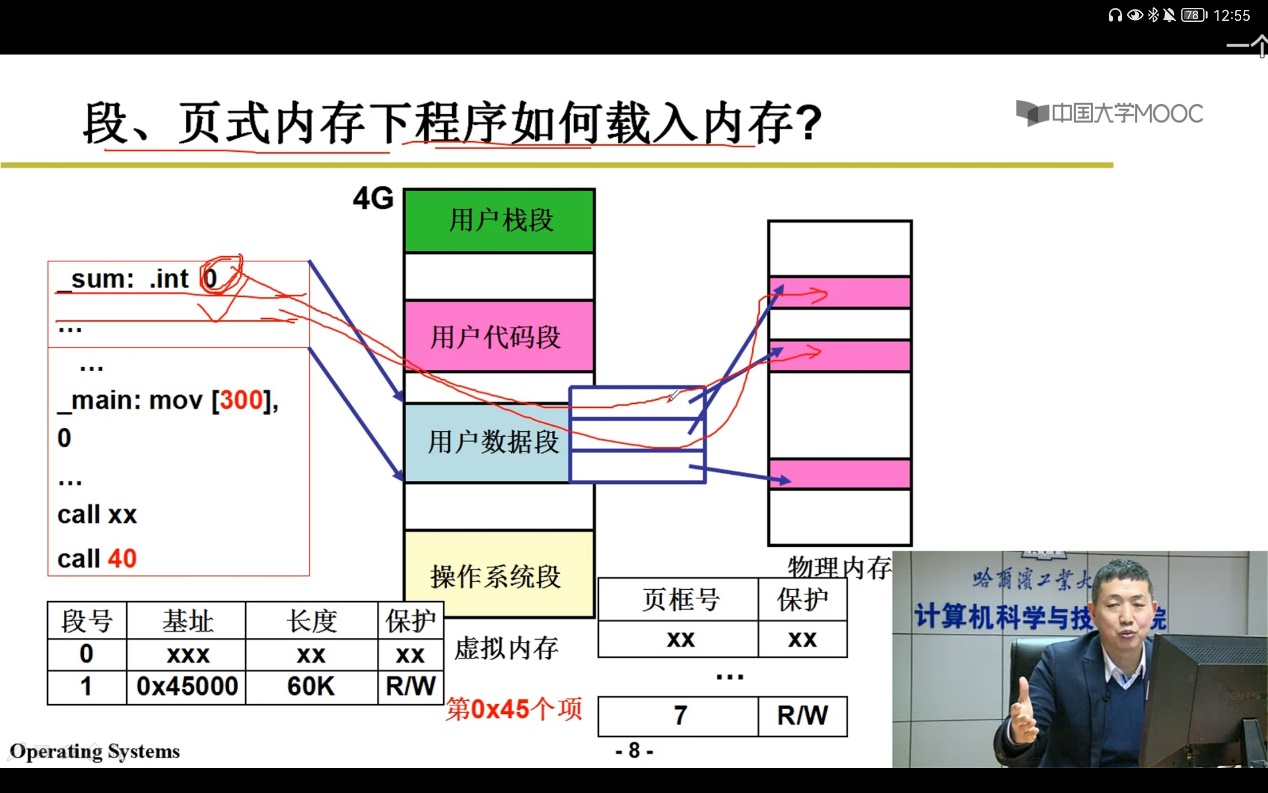
第一部分：

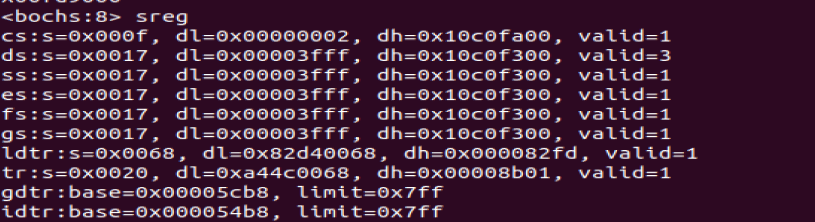
首先上结果



主要内容就是在linux-0.11下运行了一个死循环的程序，而我们要做的就是通过linux-0.11的./dbg-asm进行调试，找到循环进行比较的的那一步单步调试，然后根据ds寄存器里面的段选择子去查LTD表，找到段基址，然后根据段偏移（ds中的虚拟地址）去找到段地址，然后通过段地址去查页表，找到真实的物理地址，通过调式命令修改那个物理地址的值，最后可以看到bochs下的程序跳出循环。



实验流程及个人感悟：当我们操作系统给用户分一个地址时，这个地址并不是一个真实的物理地址，而是一个虚拟的地址，我们要获得这个物理地址，首先需要的就是查看各个寄存器的值，通过sreg查看

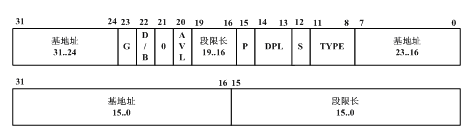


（在这里我们需要了解”s=”就是我们的段选择子：其中 RPL 是请求特权级，当访问一个段时，处理器要检查 RPL 和 CPL（放在 cs 的位 0 和位 1 中，用来表示当前代码的特权级），即使程序有足够的特权级（CPL）来访问一个段，但如果 RPL（如放在 ds 中，表示请求数据段）的特权级不足，则仍然不能访问，即如果 RPL 的数值大于 CPL（数值越大，权限越小），则用 RPL 的值覆盖 CPL 的值。

而段选择子中的 TI 是表指示标记，如果 TI=0，则表示段描述符（段的详细信息）在 GDT（全局描述符表）中，即去 GDT 中去查；而 TI=1，则去 LDT（局部描述符表）中去查。



“dl= ,dh=”就是段描述符：64位



可以看到，段描述符是一个 64 位二进制的数，存放了段基址和段限长等重要的数据。其中位 P（Present）是段是否存在的标记；位 S 用来表示是系统段描述符（S=0）还是代码或数据段描述符（S=1）；四位 TYPE 用来表示段的类型，如数据段、代码段、可读、可写等；DPL 是段的权限，和 CPL、RPL 对应使用；位 G 是粒度，G=0 表示段限长以位为单位，G=1 表示段限长以 4KB 为单位；）

既然我们已经知道了ds寄存器的值，段选择子，段描述符。那么就可以直接根据段选择子的值0x0017=0000000000010111（RPL为3，处于最低级别，ti等于1，去访问LTD表，偏移为3 也就是前13位，其实我们也可以从上面网课的图可以看出来，因为ds寄存器存储的时数据段，而数据段又在第三个位置，所以段偏移肯定就是3）偏移是3，我们通过LDT的段描述符可以知道LDT表的初地址，因为一个LDT表项占64位，然后通过偏移3可以知道ds所在位置的LDT表项，也就是ds的段描述符。（这里也可以通过查GDT表的段描述符知道GDT的初地址，然后通过LTD的段选择子来确定偏移，然后就可以知道LDT的段描述符） 知道了ds的段描述符（其实这里直接就能查看了，实验只是为了让我们了解查看段表的流程），我们就可以知道ds的段基址了，这就是 ds 段在线性地址空间（硬盘上的虚拟地址）中的起始地址。然后通过ds的值0x3004 段偏移，我们就确定了ds数据的线性地址，也就是上面网课上的用户数据段上的具体段地址。我们知道了这个段地址后，就需要通过查页表来确定真实物理地址了，于是用creg命令，查看cr3的值（页表初地址），

然后我们将具体的段地址分解位10，10，12位。前面10是页目录号，中间10是页表号，最后12是页内偏移。

这一过程由MMU自动完成，如果中间发生缺页中断也不会影响，最后就是通过地址总线给物理内存发一个地址！

分解可得 0x10003004 的页目录号是 64，页号 3，页内偏移是 4。

于是我们通过xp /w 0+64\*4 其中0是总页表起始地址，然后每个页表32位（4字节），就可以查到0x00faa027

其中的 027 是属性，显然 P=1，其他属性实验者自己分析吧。页表所在物理页框号为 0x00faa，即页表在物理内存的 0x00faa000 位置。从该位置开始查找 3 号页表项，得到（xp /w 0x00faa000+3\*4），也就是ds对应的页表的地址，我们查看这个页表地址。

得到0x00fa7067，其中后12位是属性值，我们只关注前20位，也就是页表地址（物理页框号就是0x00fa7000）。

然后我们通过页内偏移 4，加上去得到0x00fa7004,就得到了这个数据的真实物理地址。

参考：

[(131条消息) HIT Linux-0.11 实验七 地址映射与内存共享 实验报告\_laoshuyudaohou的博客-CSDN博客\_linux共享内存实验报告](https://blog.csdn.net/laoshuyudaohou/article/details/103843023)

总结：汇编中查的地址是一个段偏移，我们拿到这个段偏移后需要先查段描述符，找到基地址，然后组合成段地址，通过段地址去查页表（每个页表项4字节，通过分解成10+10+12），知道页目录号，页框号，业内偏移。然后我们通过查找页表出地址，找到页目录号首地址，然后通过首地址偏移页框号\*4个字节找到该页框对应的物理地址，最后通过偏移确定物理地址，如果发现需要找的页不在物理内存中（因为虚拟地址远大于物理地址，所以肯定有很多页不在内存上而在磁盘上），就会引发缺页中断（首先在内存中申请一个空闲页，然后将当前执行程序未在内存找到的这一页加载到内存中找到的空闲页中，最后建立映射，也就是修改页表，因为段地址到页表项的映射是不变的，所以我就需要修改就是页表项中的值，也就是将某一个页表项中的物理地址修改为我们想要的物理地址），将cpu上一个寄存器的某一位置为1，将这一页加载到内存后继续执行（这些都是有MMU完成）

