2021/04/28_Windows32位内核_第6课_文件和注册表的操作、操作系统的内存管理

笔记本:	Windows32位内核
创建时间:	2021/4/28 星期三 15:21
作者:	ileemi

文件和注册表的操作

关于文件路径:

驱动程序没有写 "Create" 回调函数，三环程序使用 "CreateFile" 就打不开该驱动设备（有文件句柄才能去控制驱动设备）。

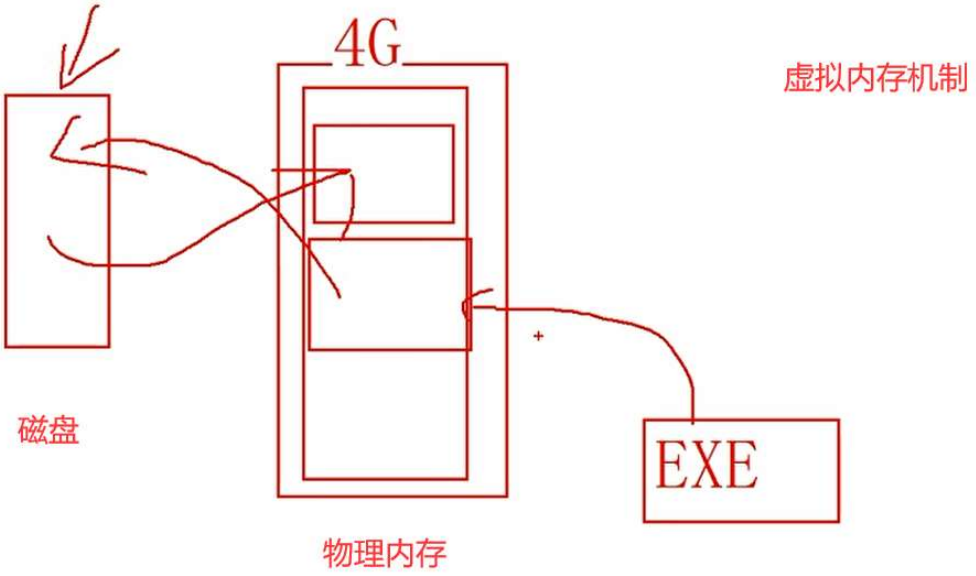
申请内存：ExAllocatePoolWithTool

内核中的内存分为：分页内存和非分页内存（存放操作系统代码）。

分页内存：允许这块内存在不使用时交换到磁盘中，提高内存使用效率。编写软件的时候，尽量申请分页内存。不合操作系统代码抢占内存资源。
非分页内存：不允许这块内存在不使用时交换到磁盘中，操作系统的代码一般防止在非分页内存中。

定义 "PAGE" 代码段，将函数代码放置到分页内存中（Ring0中）三环代码不能放置到非分页内存中，只能放置到分页内存中。
#pragma alloc_text("PAGE", xxx)
#pragma alloc_text("PAGE", xxx, xxx, xxx)

当函数调用完，就将该函数代码进行释放
#pragma alloc_text("INIT", xxx)



在C盘中有对应的文件存放（pagefile.sys）物理内存中暂时不用的数据。

STATUS_PENDING：当驱动被操作的时候，回调函数被调用，没有完成请求就会出现 PENDING 状态。操作系统会将请求者（线程）挂起，当请求完成时，恢复执行。

注册回调APC函数，查询不到文件信息，线程被切换的时候，被调用。

定位Ring3 "ReadFile" 函数在内核中的位置，进行hook。

内存管理

- IA-32架构的内存管理设施分为两部分：分割和分页。分割提供了一种隔离各个代码、数据和堆栈模块的机制，以便多个程序（或任务）可以在同一处理器上运行，而不会相互干扰。页面提供了一种实现传统的需求分页虚拟内存系统的机制，其中程序执行环境的部分根据需要映射到物理内存中。页面也可以用于在多个任务之间的隔离。在受保护模式下操作时，必须使用某种形式的分割。没有模式位来禁用分割。然而，使用分页功能是可选的。
- 分割提供了一种机制，可以将处理器的可寻址内存空间（称为线性地址空间）划分为较小的受保护地址空间，称为段。段可以用于保存程序的代码、数据和堆栈，或者保存系统数据结构（如TSS或LDT）。如果处理器上运行多个程序（或任务），可以为每个程序分配自己的段。然后，处理器强制执行这些段之间的边界，并确保一个程序不会通过写入另一个程序的段来干扰另一个程序的执行。分割机制还允许段的类型，以便限制对特定类型的段执行的操作。
- 系统中的所有数据段都包含在处理器的线性地址空间中。要在特定段中找到字节，必须提供逻辑地址（也称为远指针）。一个逻辑地址由一个段选择器和一个偏移量组成。段选择器是一个段的唯一标识符。除此之外，它还提供了一个从描述符表(如全局描述符表，GDT)到一个称为段描述符的数据结构的偏移量。每个段都有一个段描述符，它指定段的大小、段的访问权限和特权级别、段类型以及段的第一个字节在线性地址空间中的位置（称为段的基本地址）。逻辑地址的偏移部分被添加到段的基本地址中，以便定位段内的一个字节。因此，基地址加上偏移量在处理器的线性地址空间中形成一个线性地址。

内存进程的保护：

1. 权限
2. 内存保护属性
3. 内存隔离（每个任务都有自己的内存区域）

操作系统负责提供数据，CPU负责对数据进行检测。

内存管理表（在物理内存条）：
CPU通电在实模式，只能访问物理地址。在保护模式下CPU不能直接访问物理地址，需要通过虚拟地址（可以是内存管理表的下标）到内存管理表中定位对应的物理地址后，再去访问（取决于当前地址的属性等）。内存管理表（分段表）在物理内存条中的物理地址需要告诉CPU。CPU访问内存只会访问物理地址。

保护模式下需要注意的三个地址：

- 线性地址（linear address）
- 逻辑地址（logical address）：段寄存器:偏移 cs:[100]
- 物理地址（physical address）

虚拟地址：对CPU来说可以理解为偏移

CPU 处理汇编指令（逻辑地址），通过逻辑地址查询内存管理表得到的就是线性地址。当CPU没有做任何限制的情况下，线性地址等价于物理地址。在 Inter CPU 的设计中，允许做两层表，可通过线性地址 --> 分页表（用来快速交换内存） --> 物理地址。

逻辑地址 --> 分段表 --> 线性地址 --> 分页表 --> 物理地址
逻辑地址 --> 分段表 --> 线性地址 (物理地址)

分段表的模式

基本的平面模型：

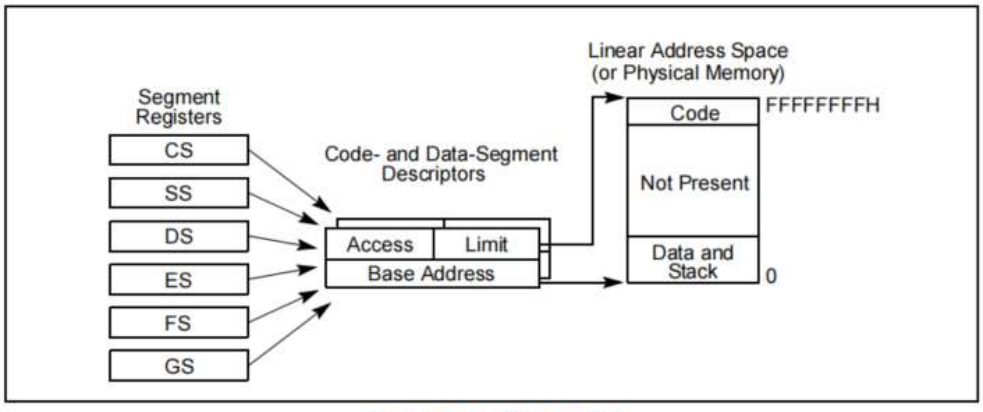


Figure 3-2. Flat Model

受保护的平面模型：

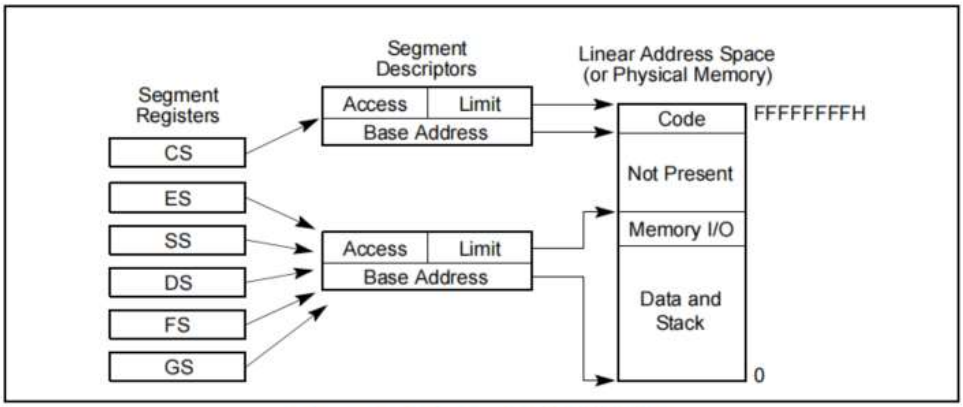


Figure 3-3. Protected Flat Model

字段描述符

> 段描述符是GDT或LDT中的一种数据结构，它向处理器提供段的大小和位置，以及访问控制和状态信息。段描述符通常由编译器、链接器、加载器、操作系统或执行程序创建，但不是应用程序。图3-8说明了所有类型的段描述符的一般描述符格式。

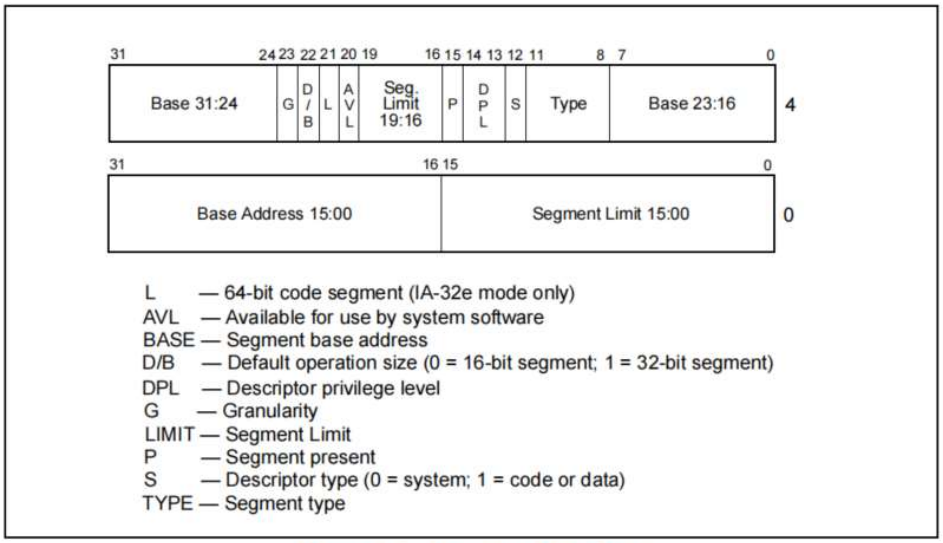


Figure 3-8. Segment Descriptor

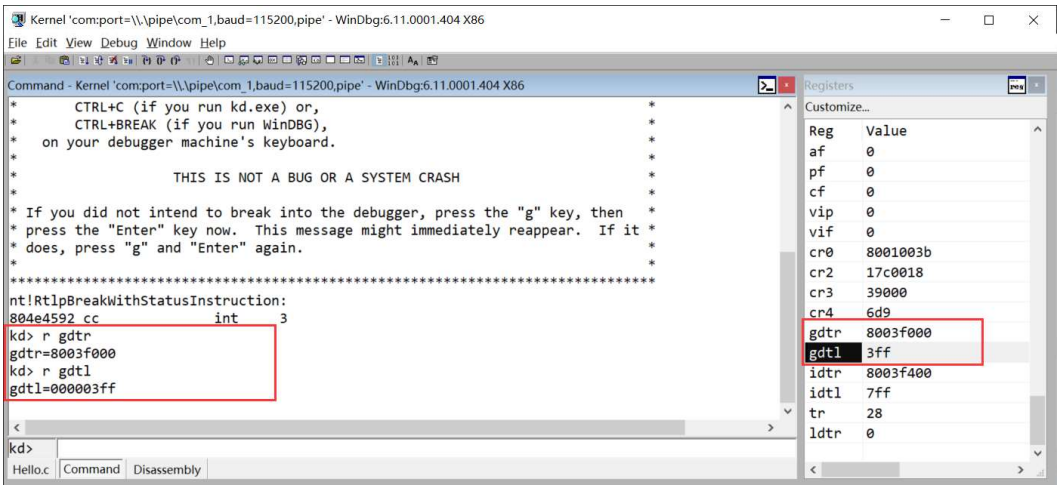
G (granularity) flag：确定数据段限制字段的缩放大小。当粒度标志被清除时，段限制以字节单位解释；当设置标志时，段限制以4-KByte单位解释。（此标志不会影响基本地址的粒度，它总是字节粒度。）当设置粒度标志时，在根据段限制检查偏移时，不会测试偏移的十二个最小重要的位。例如，当设置粒度标志时，0的限制会导致0到4095的有效偏移。

L (64-bit code segment) flag：在IA-32e模式下，段描述符的第二双字的第21位表示代码段是否包含本机64位代码。值为1，表示此代码段中在64位模式中执行的指令。值0表示此代码段中的指令是在兼容性模式下执行。如果设置了L位，则必须清除D位。当不在IA-32e模式下或对于非代码段时，位21被保留，并且应该始终设置为0。

AVL（Available and reserved bits）：段描述符的第二个双字的第20位可供系统软件使用。

段描述表

> 通过寄存器 "gdtr" 获取段描述表起始地址，通过 gdtl 获取段描述表的大小。再Windbg中通过 "db 8003f000 l3ff" 查看对应的内存数据。



`dq gdtr/addr`：

```
kd> dq gdtr
8003f000  00000000`00000000 00cf9b00`0000ffff
8003f010  00cf9300`0000ffff 00cffb00`0000ffff
8003f020  00cff300`0000ffff 80008b04`200020ab
8003f030  ffc093df`f0000001 0040f300`00000fff
8003f040  0000f200`0400ffff 00000000`00000000
8003f050  80008955`22000068 80008955`22680068
8003f060  00009302`2f40ffff 0000920b`80003fff
8003f070  ff0092ff`700003ff 80009a40`0000ffff
```

无效项

通过 `dg` 指令可以快速判断代码段/数据段的描述符信息，使用方法：`dg` + 相对于GDT表首地址的偏移，以第三项 "0000ffff 00cf9300" 为例，它距离GDT表首16个字节，也就是两个GDT表长度，输入 `dg 10` 即可：

```
kd> dg 10
                                   P Si Gr Pr Lo
Sel     Base     Limit     Type    l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0010 00000000 ffffffff Data RW Ac 0 Bg Pg P  Nl 00000c93
```