Linux内核-实战

源自**技术简说**第8讲

逻辑地址,虚拟地址,线性地址和物理地址

讲师：杨文川

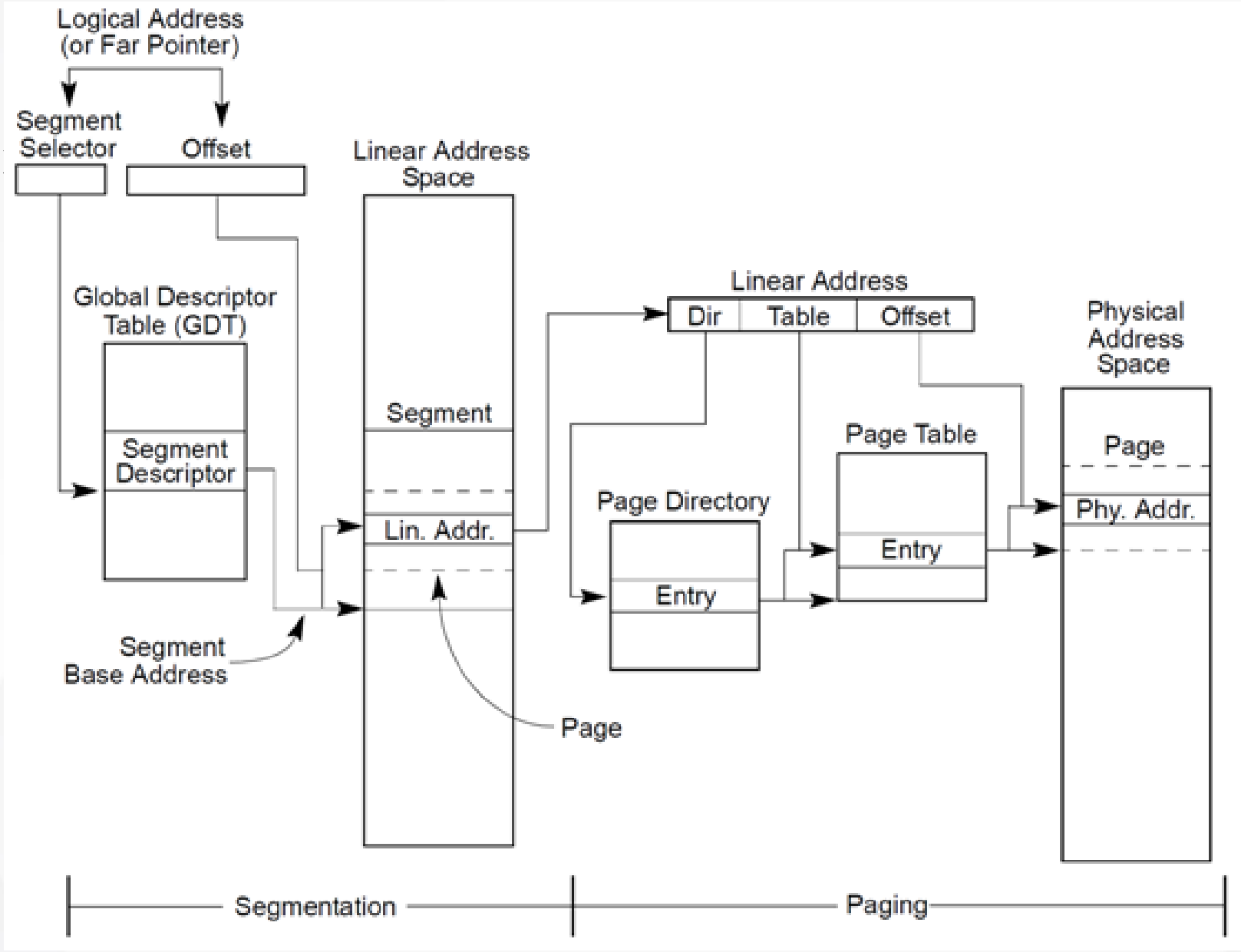
内容

- 1 名词解释
- 2 x86的段页式内存管理机制
- 3 逻辑地址、虚拟地址和线性地址的关系

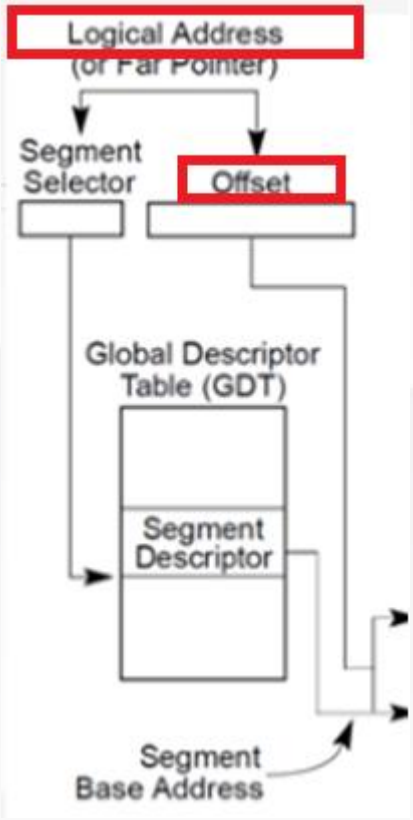
引言

- Linux内核关于地址有四个概念：逻辑地址、虚拟地址、线性地址和物理地址。
- 物理地址比较好了解，但是逻辑地址、虚拟地址、线性地址的区别到底是什么？
- 下面就来详细介绍。
- 中英文对应关系：
  - 逻辑地址 --- logical address;
  - 虚拟地址 --- virtual address;
  - 线性地址 --- linear address;
  - 物理地址 --- physical address;
- 这四个地址是体系相关的，以x86 CPU为例进行解释。

一、名词解释

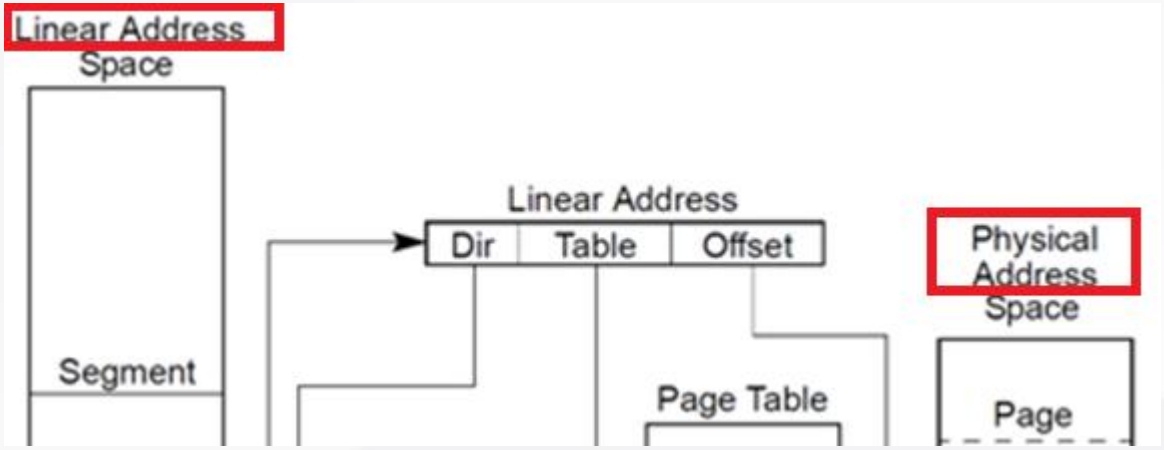
- 
Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Linear Address
Dir
Table
Offset
Physical
Address
Space
Page Table
Page
Phy. Addr.
Page Directory
Entry
Entry
Segment
Base Address
Page
Segmentation
Paging

- x86 CPU 段页式内存管理机制
- 1.**逻辑地址**就是左上角的Logical Address。
- 逻辑地址是由一个段选择符，加上一个指定
- 段内相对地址的偏移量(**Offset**)组成的，
- 表示为
- [段选择符：段内偏移量]，例如：[CS：EIP]



- 2.**虚拟地址**，就是上面逻辑地址的**段内偏移Offset**。所以：
- 逻辑地址可以表示为 [段标识符：**虚拟地址**]
- 驱动代码或者应用程序中，所用的地址就是**虚拟地址**，比如以下程序中指针p的输出：

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
        int *p;
        p = (int*)malloc(sizeof(int));
        printf("addres is %p\n", p);
        free(p);
        return;
}
```



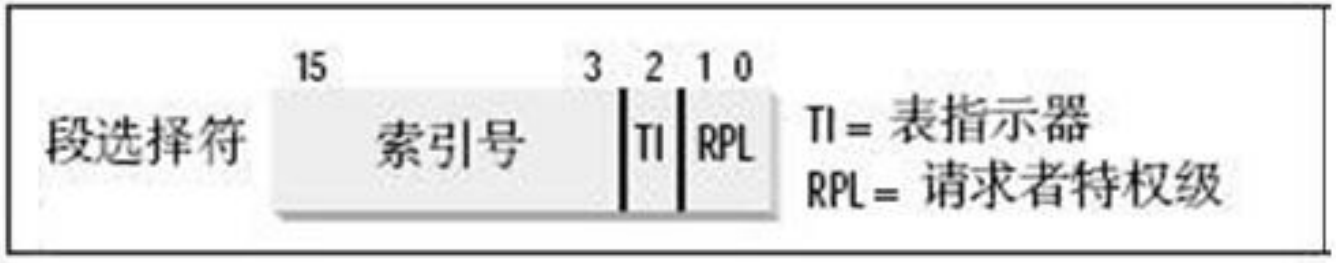
- 3.**线性地址**就是Linear Address，线性地址是平坦的统一地址空间。intel x86 中，线性地址是由逻辑地址经过段页式转换得到的
- 4.**物理地址**就是最右边的Physical Address。
- 物理地址就是物理内存的地址。注意在做页表转换的时候，这里存的是物理内存块的编号，不是真正的物理地址。
- 内核把物理内存按照4K大小编号，考虑k到物理内存的起始地址是固定的，所以从内存编号的序号，就可以算出该编号对应的物理内存块的起始地址了。

- 例如：
- 物理内存起始地址为0x50000000,
  - 那么编号为0的物理块的起始地址为：0x50000000
  - 编号为1的物理块的起始地址为 0x50001000
  - 以此类推….
- 所以，根据物理内存块的编号，就可以转换得到该物理内存块的起始地址，也叫做物理内存的基地址。
- 了解这一点非常重要，后续做页表映射的时候会用到。

二、x86的段页式内存管理机制

- 还是上面的那张图，除了表达了这三个地址之外，我们还可以看出从逻辑地址转换成最终的物理地址，要经历两个过程：
- 1. **逻辑地址转换为线性地址**
- 在 Intel 平台下，逻辑地址(logical address)是 selector:offset 这种形式，selector 可以是代码段或者数据段，offset 是段内偏移。
- 如果用 selector 去 GDT(全局描述符表) 里拿到 segment base address(段基址) 然后加上 offset(段内偏移)，这就得到了 linear address。我们把这个过程称作**段式内存管理**。

- 总之：逻辑地址转换为线性地址的详细过程是这样的：
  - (1)先从段选择符(selector)中得到段描述符；
  - (2)从段描述符中得到段基地址；
  - (3)线性地址=上一步得到的段基地址+段内偏移(也就是上文说的虚拟地址)
- 我们来看看端选择符(selector)的组成：
- 13位的索引号对应到段描述符表中的位置，而T1字段表示使用的是哪个段描述符表。

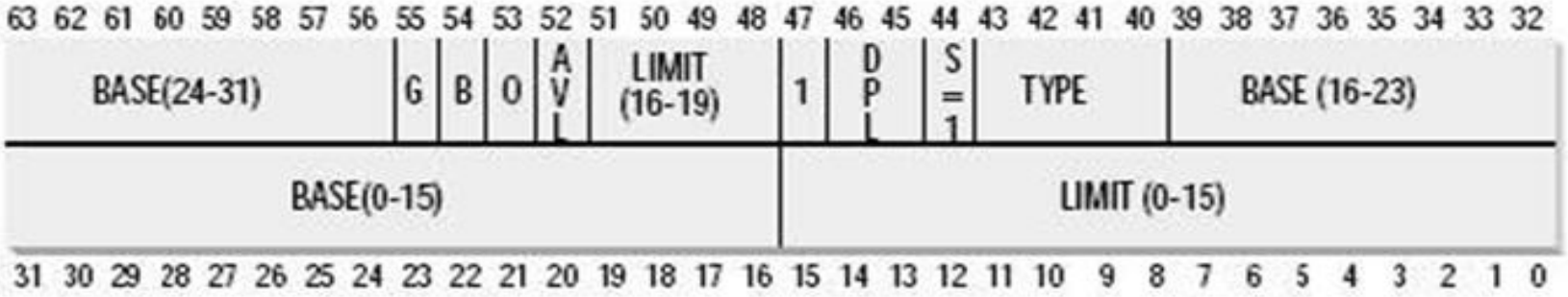


- Intel设计的本意是：
  - 一些全局的段描述符，就放在“全局段描述符表(GDT)”中
  - 一些局部的，例如每个进程自己的，就放在“局部段描述符表(LDT)”中。
- 所以，通过T1字段就可以选择是使用GDT，还是使用LDT了。GDT或者LDT里的内容就是一个一个的段描述符，段描述符的组成如后图：

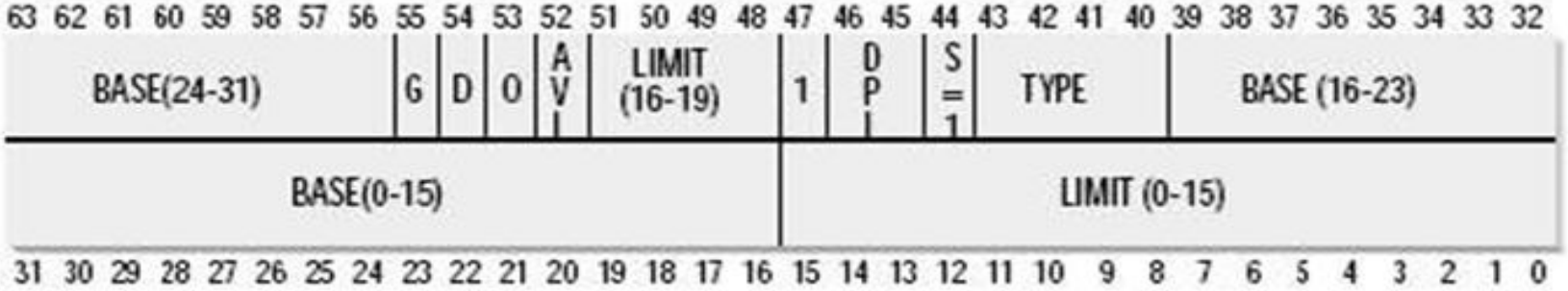
- 这些描述貌似很复杂，不过，我这里只关心一样，就是Base字段，它描述了一个段的的基地址。
- 得到了这个基地址，然后再加上段内偏移offset，就得到了最终的线性地址。

图

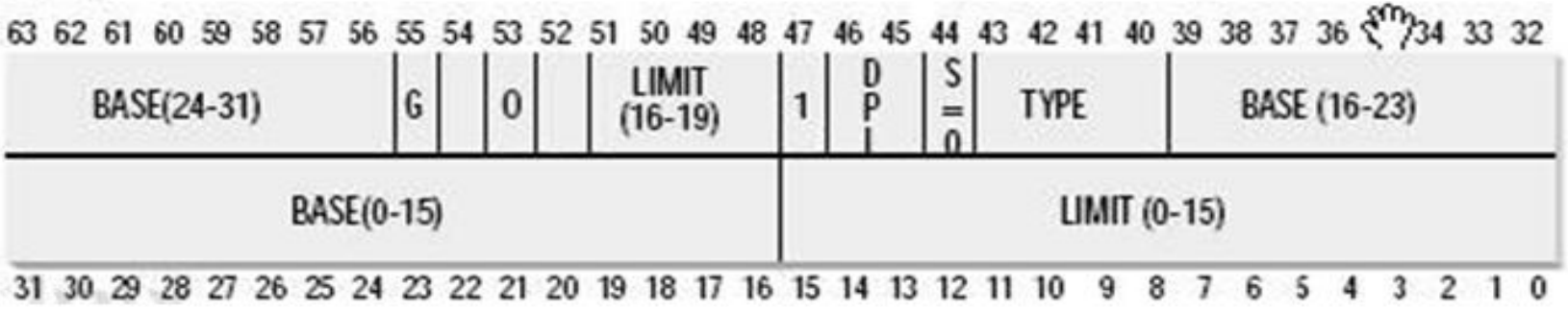
数据段描述符



代码段描述符



系统段描述符



- 2.**线性地址转换为物理地址**
- 如果再把线性地址切成三段，用前两段分别作为索引去PGD、Page Table里查表，会先得到一个页目录表项、
- 然后会得到一个页表项(Page Table Entry)，那里面的值就是一个物理内存块的起始地址(其实是物理内存编号)，
- 把它加上 linear address 切分之后第三段的内容(又叫页内偏移)就得到了最终的 physical address。
- 我们把这个过程称作**页式内存管理**。
- 所以，x86采用的是**段页式内存管理**的方式。

三、Linux内核中逻辑地址、虚拟地址和线性地址的关系

- 上一部分提到了x86采用了段页式内存管理。
- 按照 Intel 的设计，段式内存管理中的段类型分为三种：
- 　　代码段、数据段、系统段(TSS之类的)，实在麻烦。
- 我们只靠页式内存管理，就已经可以完成Linux内核需要的所有功能，根本不需要段映射，但是段映射又关不掉。
- 于是，Linux内核将所有类型的段的 segment base address 都设成0(包括内核数据段、内核代码段、用户数据段、用户代码段等)。
- 那么这样一来所有段都重合了，也就是不分段了，此外由于段限长是地址总线的寻址限度，所以这也就相当于所有段内空间跟整个线性空间重合了。

- 这样逻辑地址也就简化为了段内的偏移量(逻辑地址=虚拟地址)。
- 由于段基地址变为了0，那么：
- 　线性地址=逻辑地址=虚拟地址。
- 所以，在x86 linux内核里，
- 逻辑地址、虚拟地址、线性地址，这三个地址是一致的

谢　谢