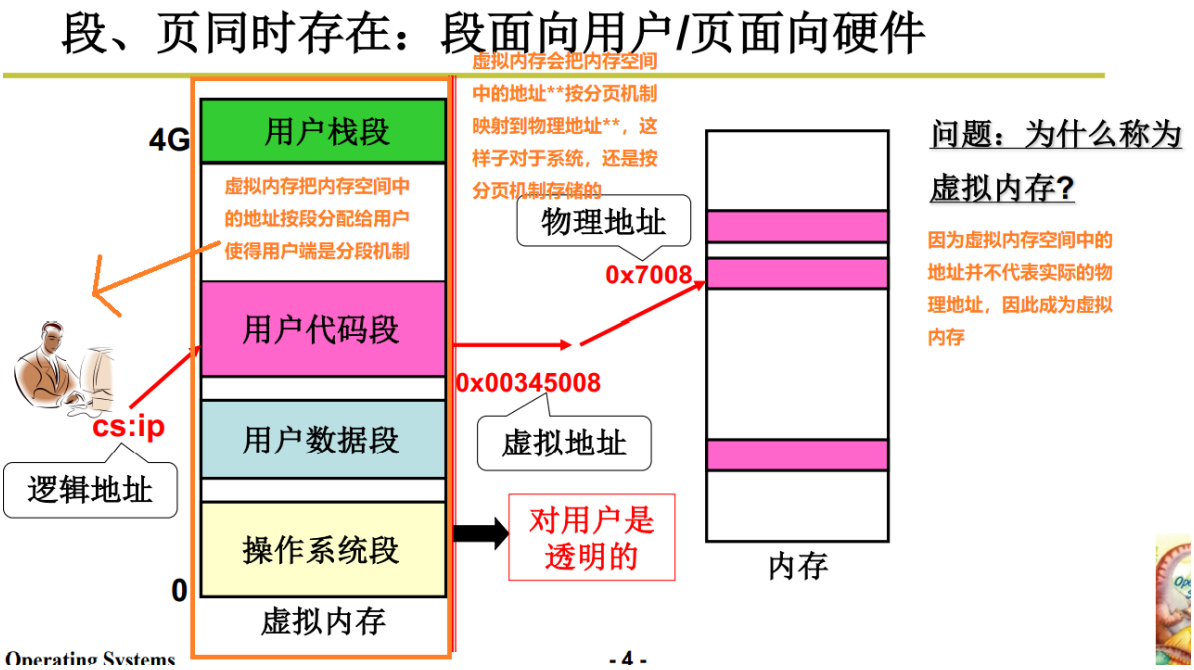
哈工大操作系统-L23段页结合的实际内存管理

- 用户用分段机制书写程序；系统用分页机制管理内存。
- 如何结合为本课核心。

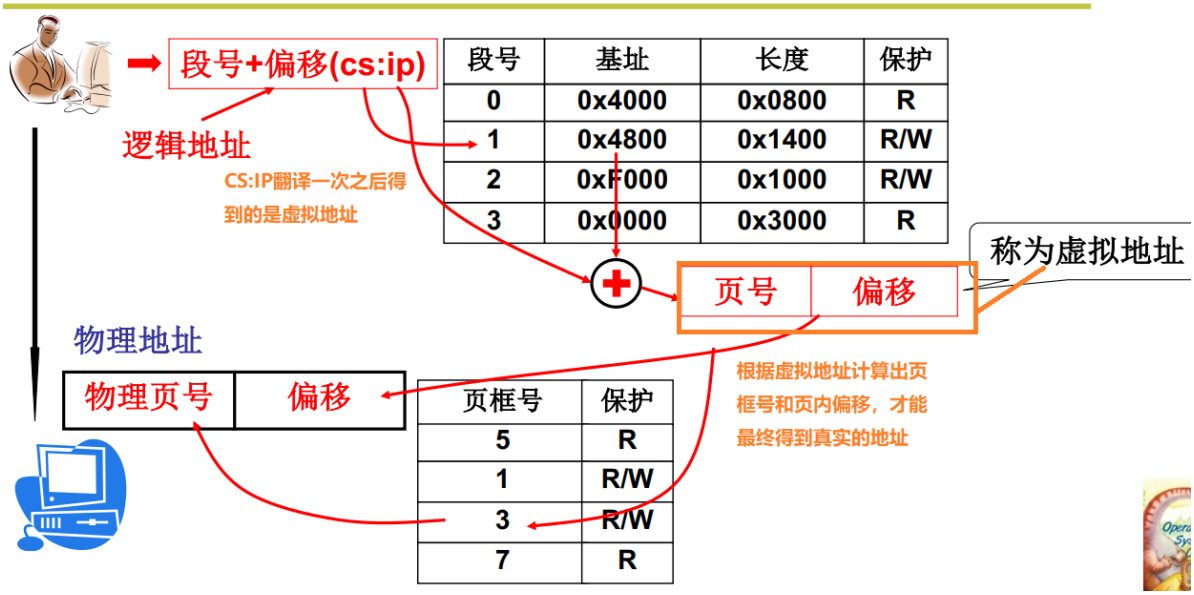
1.虚拟内存--联系段和页的中间桥梁



- 虚拟内存把内存空间中的地址**按段分配给用户**使得**用户端是分段机制**
- 虚拟内存会把内存空间中的地址**按分页机制映射到物理地址**，这样子**对于系统，还是按分页机制存储的**
- 因为虚拟内存空间中的地址并不代表实际的物理地址，因此称为虚拟内存
- 对用户端，用户感觉不到虚拟内存的存在，因此用户端会认为是分段机制的
- 对于物理内存而言，映射也使得实际的存取是按页的，也支持系统的分页机制

2.段、页同时存在时的重定位(地址翻译)

段、页同时存在是的重定位(地址翻译)

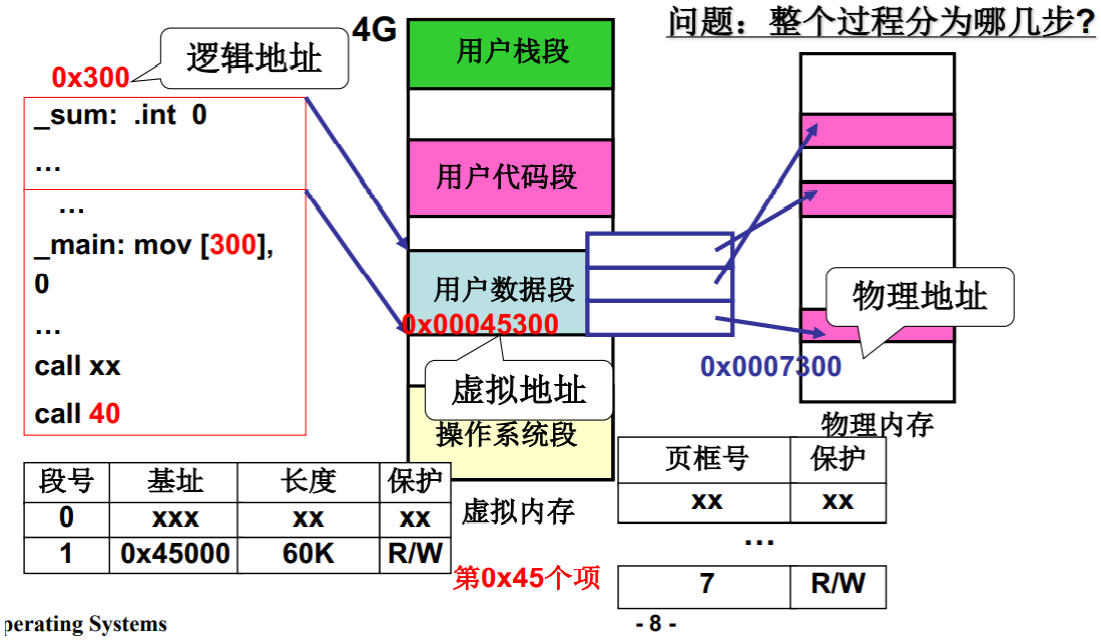


经历两重的地址翻译：

- 分段的地址翻译--从用户端到虚拟地址空间
- 分页的地址翻译--从虚拟地址空间到物理地址

3.段页内存的实际实现

段、页式内存下程序如何载入内存?



- 先在虚拟内存中，割出用户程序所需的各个段。(如何割？分区适配算法)
- 虚拟内存的各个段会被分割成页，放置在不同的页框。
- 段表记录用户程序的段信息
- 页表记录虚拟内存段分页的信息

3.1分配虚拟内存，建立段表

故事从fork()开始　分配虚存、建段表

- fork()→sys_fork→copy_process的路都已经走过了

```
在linux/kernel/fork.c中
int copy_process(int nr, long ebp,...)
{    ...
     copy_mem(nr, p); ...
```

的确是进程带动内存!

- 现在开始分析当时那个神秘的copy_mem了

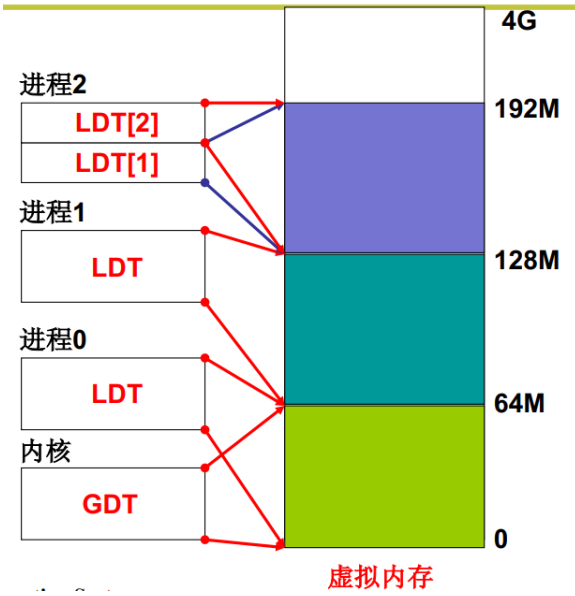
```
int copy_mem(int nr, task_struct *p)
{
    unsigned long new_data_base;
    new_data_base=nr*0x4000000;   //64M*nr
    set_base(p->ldt[1],new_data_base);
    set_base(p->ldt[2],new_data_base);
```

虚拟内存的基址

段表

p是什么?进程切换跟着切换

进程0、进程1、进程2的虚拟地址



- 每个进程的代码段、数据段都是一个段
- 每个进程占64M虚拟地址空间，互不重叠

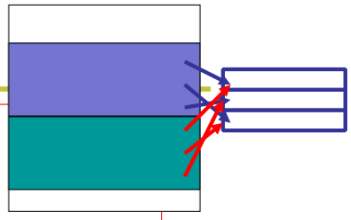
问题：这意味着什么样的简化?

- LDT1应是记录程序段在虚拟内存的基址，LDT2应该是记录程序段结束的地址，初始化为基地址。
- 每个进程的代码段、数据段都是一个段
- **每个进程占据64MB的虚拟地址空间，互不重叠**
  - 互不重叠意味着，页表可以共用一套?
  - 因为是以虚拟地址为页号进行页框的查找

3.2 分配内存，建页表

页表为虚拟内存到物理内存的映射

接下来应该是什么了？分配内存、建页表



```
int copy_mem(int nr, task_struct *p)
{  unsigned long old_data_base;
   old_data_base=get_base(current->ldt[2]);
   copy_page_tables(old_data_base,new_data_base,data_limit);
```

```
int copy_page_tables(unsigned long from,unsigned long to, long size)
{  from_dir = (unsigned long *)((from>>20)&0xffc);
   to_dir = (unsigned long *)((to>>20)&0xffc);
   size = (unsigned long)(size+0x3fffff)>>22;
   for(; size-->0; from_dir++, to_dir++){
      from_page_table=(0xfffff000&*from_dir);
      to_page_table=get_free_page();
```

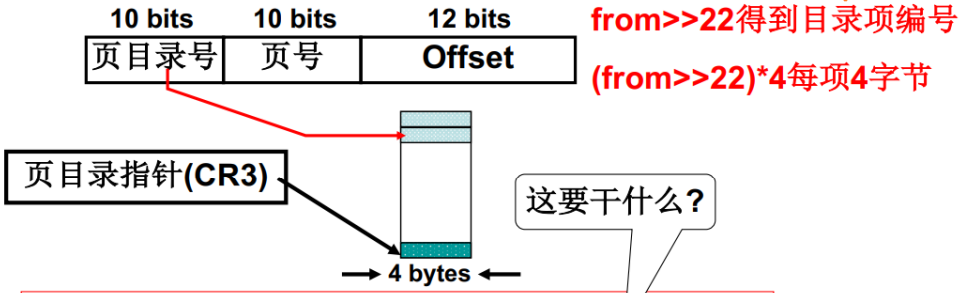
父进程创建子进程，如果共用存储空间，则是不需要给子进程分配实际内存的。
但是由于需要给子进程分配虚拟内存，因此需要建立新的页表

这里的from_dir，to_dir是什么？

```
from_dir = (unsigned long *)((from>>20)&0xffc);
to_dir = (unsigned long *)((to>>20)&0xffc);
size = (unsigned long)(size+0x3fffff)>>22;
```

因此右移20位然后最后两位变0相当于右移22位*4的效果

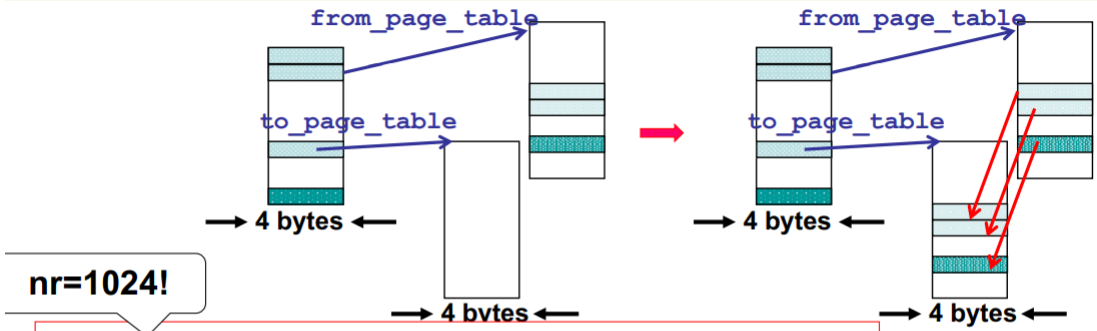
- from是？32位虚拟地址，这个地址的格式是否还记得？



因为右移动22位之后找到页目录号即页表的地址，然后需要乘4，才能得到下一个页表的地址。
因为是以字节寻址，而每个页表地址长4字节。
所以该地址要乘4才能得到下一个地址。

```
for(; size-->0; from_dir++, to_dir++){
      from_page_table=(0xfffff000&*from_dir);
```

接下来干什么应该猜也猜的到…

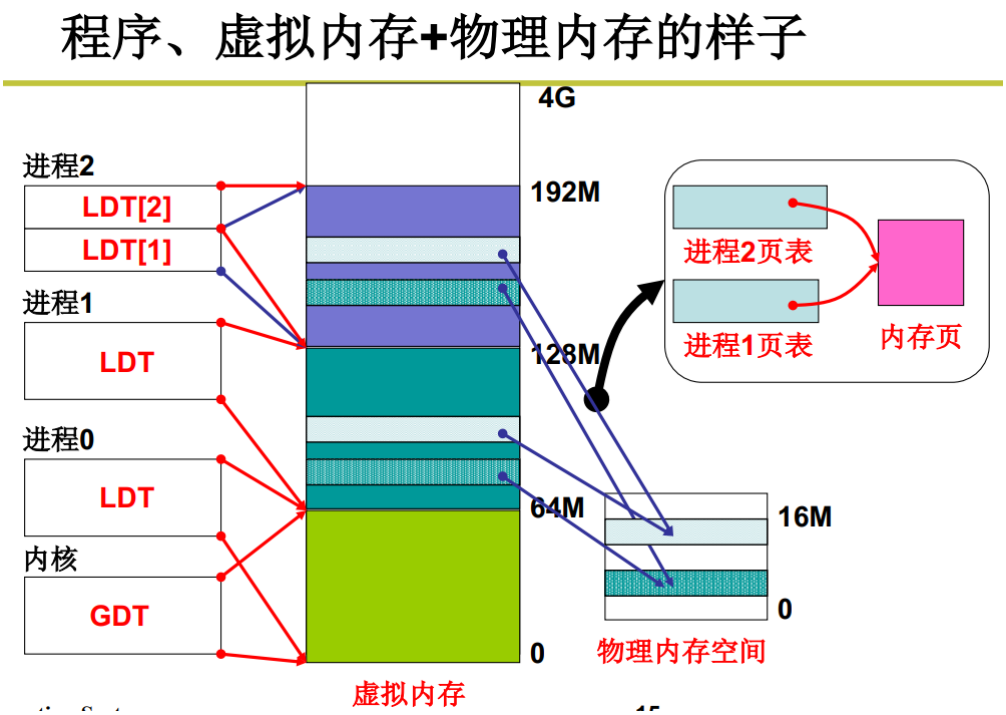


nr=1024!

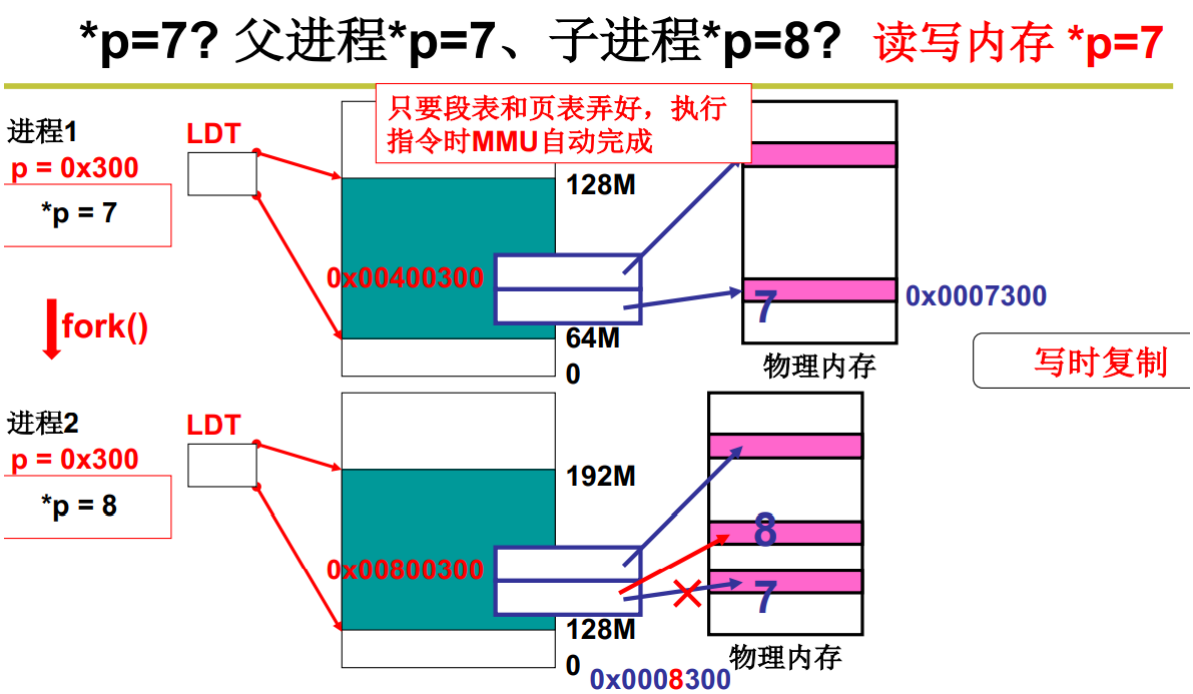
```
for(;nr-->0;from_page_table++,to_page_table++){
   this_page = *from_page_table;
   this_page&=~2;//只读
   *to_page_table=this_page;
   *from_page_table=this_page;
   this_page -= LOW_MEM; this_page >>= 12;
   mem_map[this_page]++; }
```

把from page table所指的内容复制给to page table

3.3结果



3.4最后，MMU自动执行从逻辑地址到物理地址的转换



- 在子进程写*p=8时，我们不会再在原来的7300处保存这个P了。因为父进程的数据对子进程是只读的。
- 子进程会创建一个新的区块，去保存我们的指针p=8。即，翻译指针p时，尽管逻辑地址，以及由逻辑地址到虚拟内存地址都是与父进程一致的，但在虚拟地址映射到物理地址时，因为父进程的指针p是不能修改的，因此我们会把子进程的指针P的虚拟地址映射到新的物理地址，然后保存指针P的值。