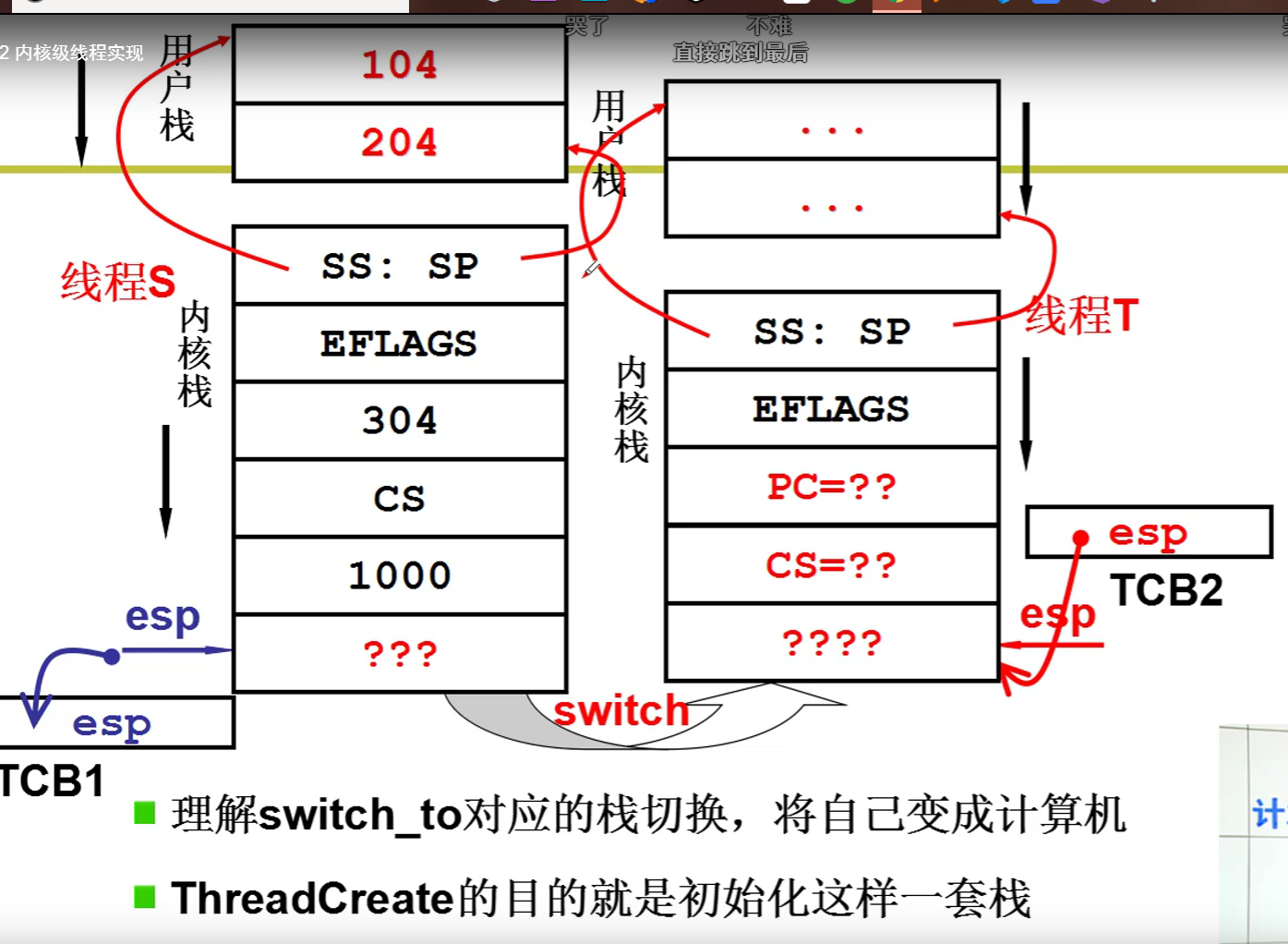
**这部分也就是操作系统核心思想 李治军老师b站课程 12讲，这部分书上没有细致讲解，但他的确非常重要。**

**大致原理可以看书，书上给出了详细的思路**

**流程思路（还是那5步操作） 这个阶段我们会去源码去剖析，见证这个具体过程**

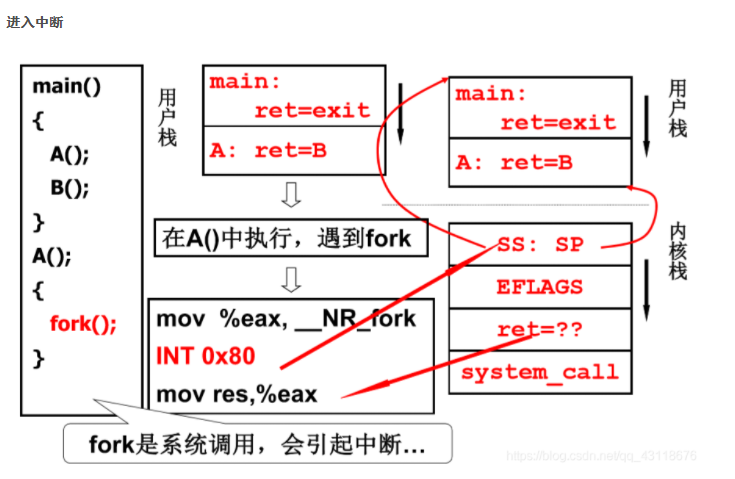


**第一步**

我们是从用户态切换到内核态，既然是进入内核，那么我们就无法避免中断切换权限，根据IDE表查找实际内核代码位置（中断开始，各种初始化），以Iwrite为例，我们需要找出他实际位置，根据IDE找出了他的位置，然后就是调用system\_call 找到Iwrite内存位置，这里其实是一个数组，存储了一些内核中函数的位置，system\_call功能就是从这个数组找出我们需要的函数位置。（这里讲述中断开始和system\_call的关系）

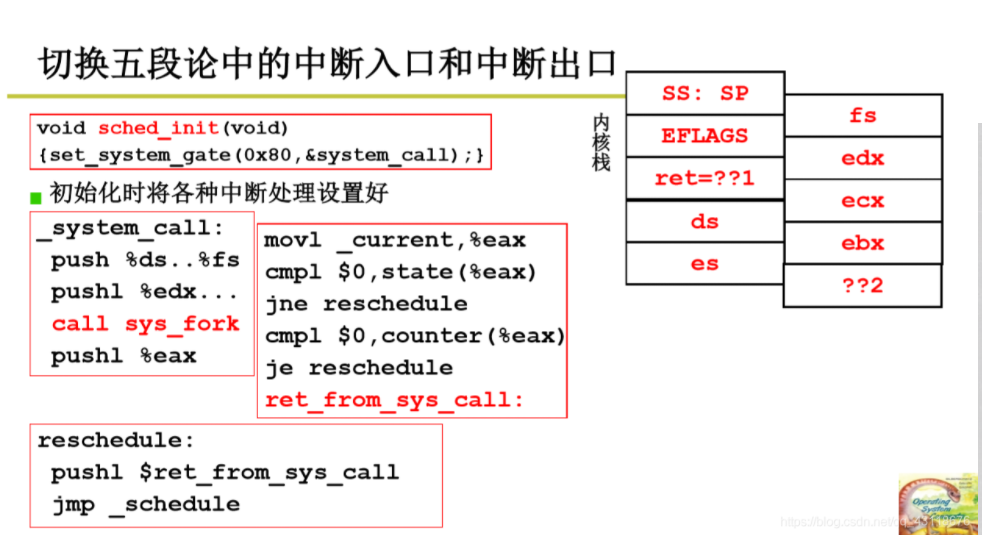
fork（）用于进入内核执行相应代码

main 函数执行到a（），a中遇到fork函数 ，fork中的int 0x80 中断开始 ，只有完成system\_call找到真正内核代码位置，我们才算进入内核，所以int 0x80 中断开始 存储的前两个部分（ss ：sp）毋庸置疑就是用户态时期代码信息，ret存储了INT 0x80下一条代码的信息（记住这一点）



in 0x80 中断开始任务做完后 进入到 system\_call 阶段，这个阶段依旧是保存用户态的各种寄存器信息

也就是 call sys\_fork 之前的各种push



**第二步**

设置内核栈的tcb原理跟前面用户态相似。

**第三步 内核栈之间的相互切换。（这里的问题关键是 在内核中我们一开始没有这两个线程，我们只有一个进入到内核中的进程，这个也就解释我们在iret的时候为什么使用的是通（））**

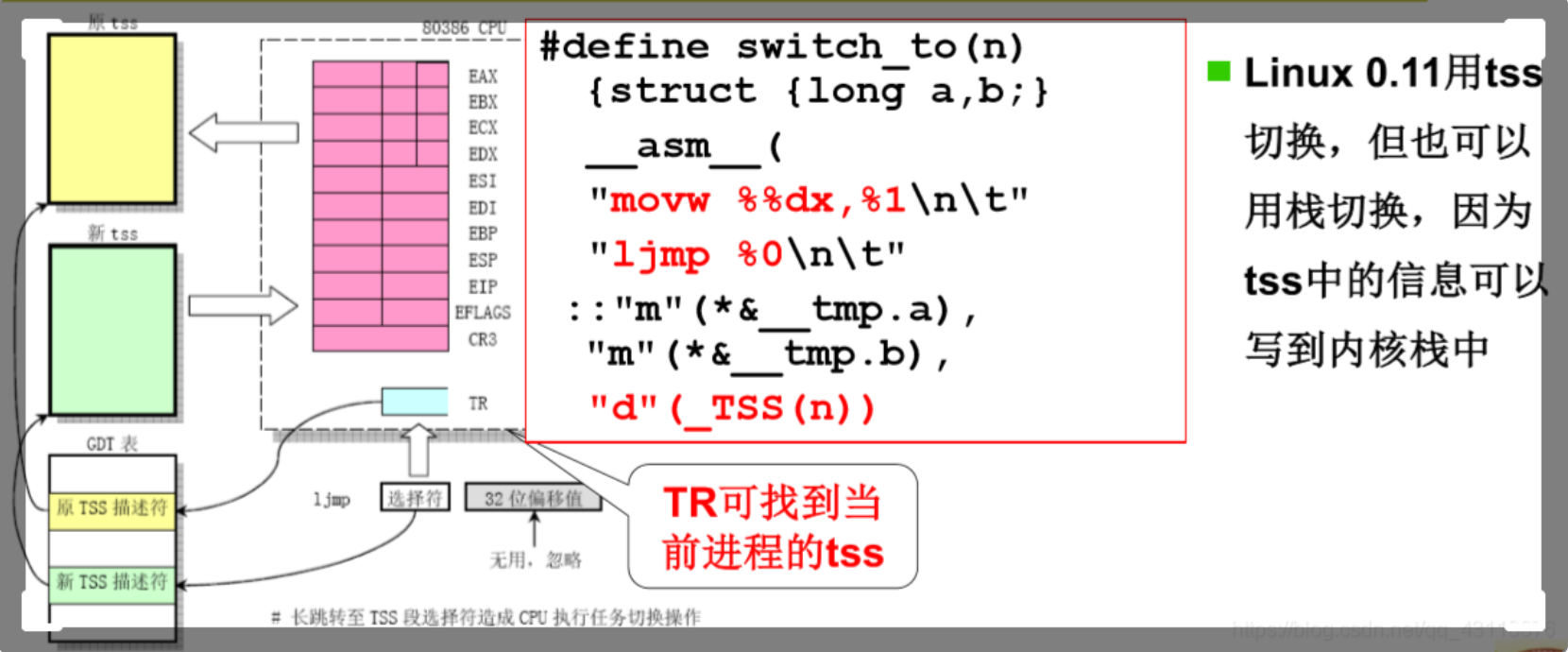
我们原来的思路是用户栈1和用户栈2通过交换es 和sp 完成栈之间的切换，同时通过}符号（ret） 把初始化在用户栈2的cs 和ip地址弹出，让我们定位到线程2的代码位置。

这里老师思路是基于liunx 0.11

linux 0.11 中基于TSS(Task Struct Segement) 切换，实验中将其改为基于内核栈的切换，TSS切换速度较慢，现代操作系统中大多为内核栈切换。

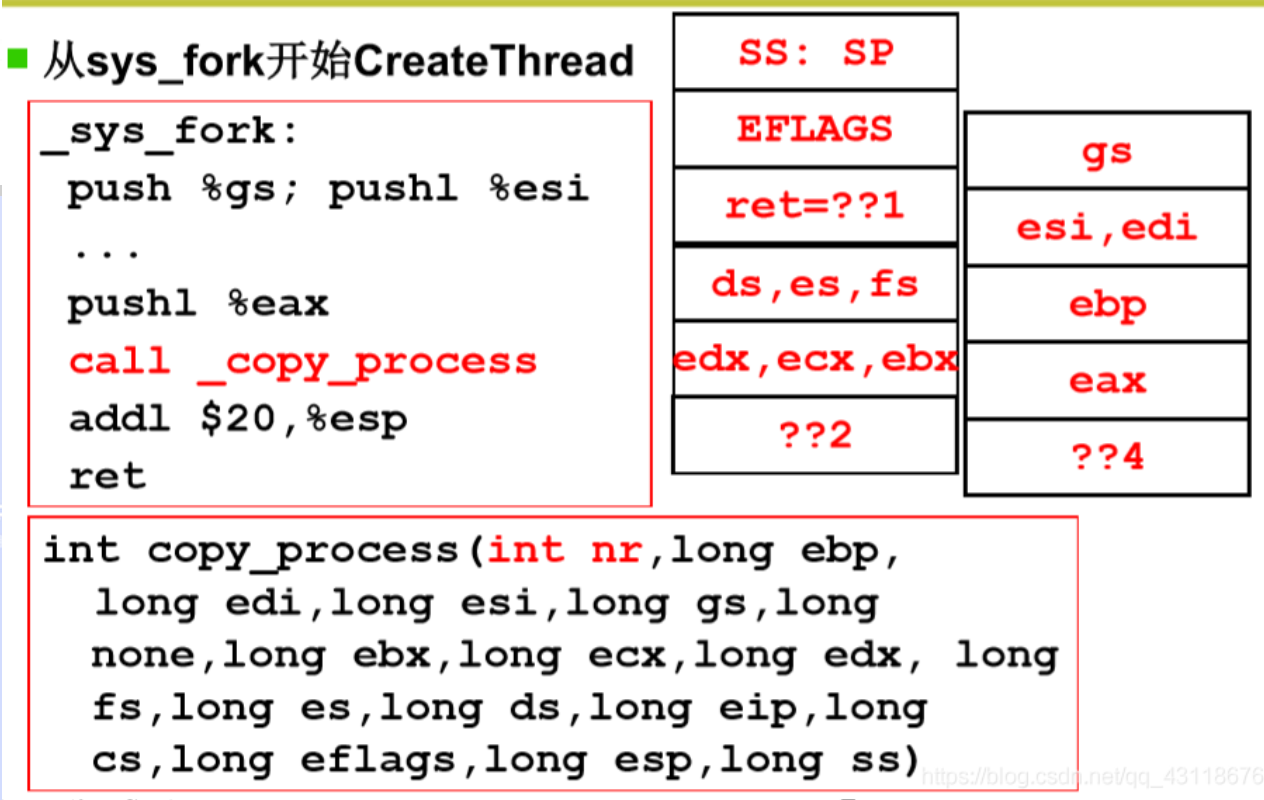
TSS 类似于内存快照，保存各种寄存器信息，经过tr我们通过gdt表转换出我们实际需要的tss位置。新的tss（需要切换的用户栈2 姑且这么理解）加载到cpu中，父线程的信息保存到旧tss中，请注意所有信息都保存在了tss表中，包括我们切实需要的ss sp 和cs ip，完成了前面用户态的栈切换和tcb切换。

至于下面图片代码不清楚（即使看完了汇编，真的想哭，咋那么多奇怪的写法）



所以tss表的初始化十分重要，他关系到我们内核栈的切换

sys\_fork 完成这项任务（谈到fork 可以说这个是本课程的核心，他对于内核栈的切换做了一个充分诠释）看看书上第六章一个大折痕



我们为什么需要在内核中创建父子进程，实际上是为了调用不同的代码，调用堆栈2（也就是父进程复制过来的子进程）对应的实际处理任务代码。子进程很多包括堆栈都是复制父进程的，但是他们的eax不一样，我们就利用if语句对eax值的判断来实现父子进程，不通代码切换。

call \_copy\_process 用于实现创建子进程

copy\_process

创建子进程完成以后，我们还没有执行子进程的处理任务代码啊。在sys-fork中的cmpl第二判断就是在判断是不是子进程，如果是我们就调用子进程，第一个cmpl我们用来判断是不是父进程，根据eax来区分

