

操作系统

Operating Systems

L11 内核级线程

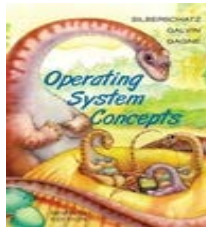
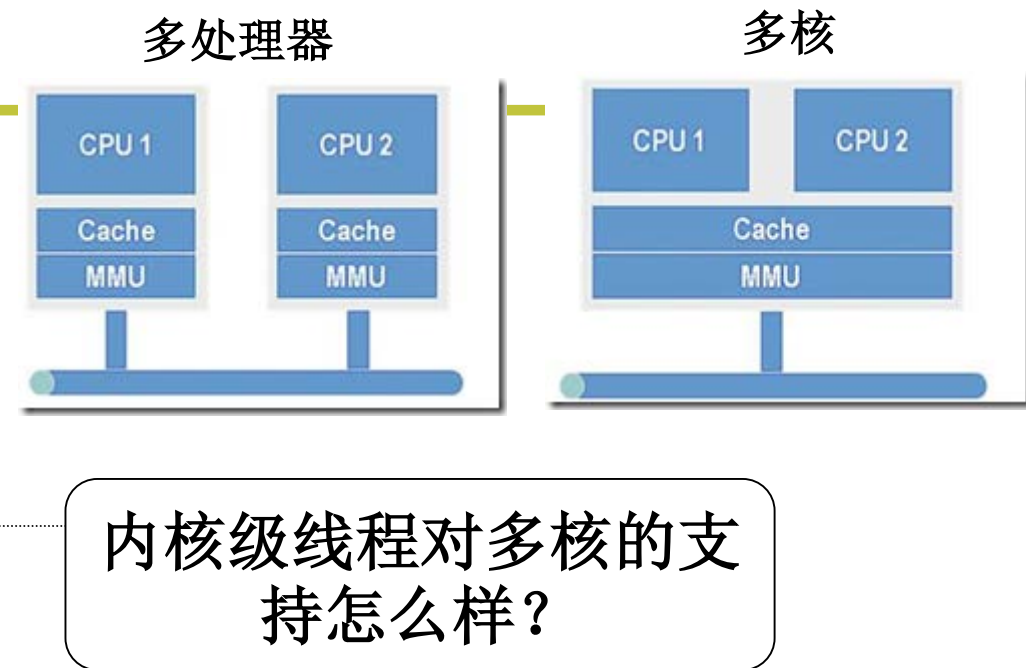
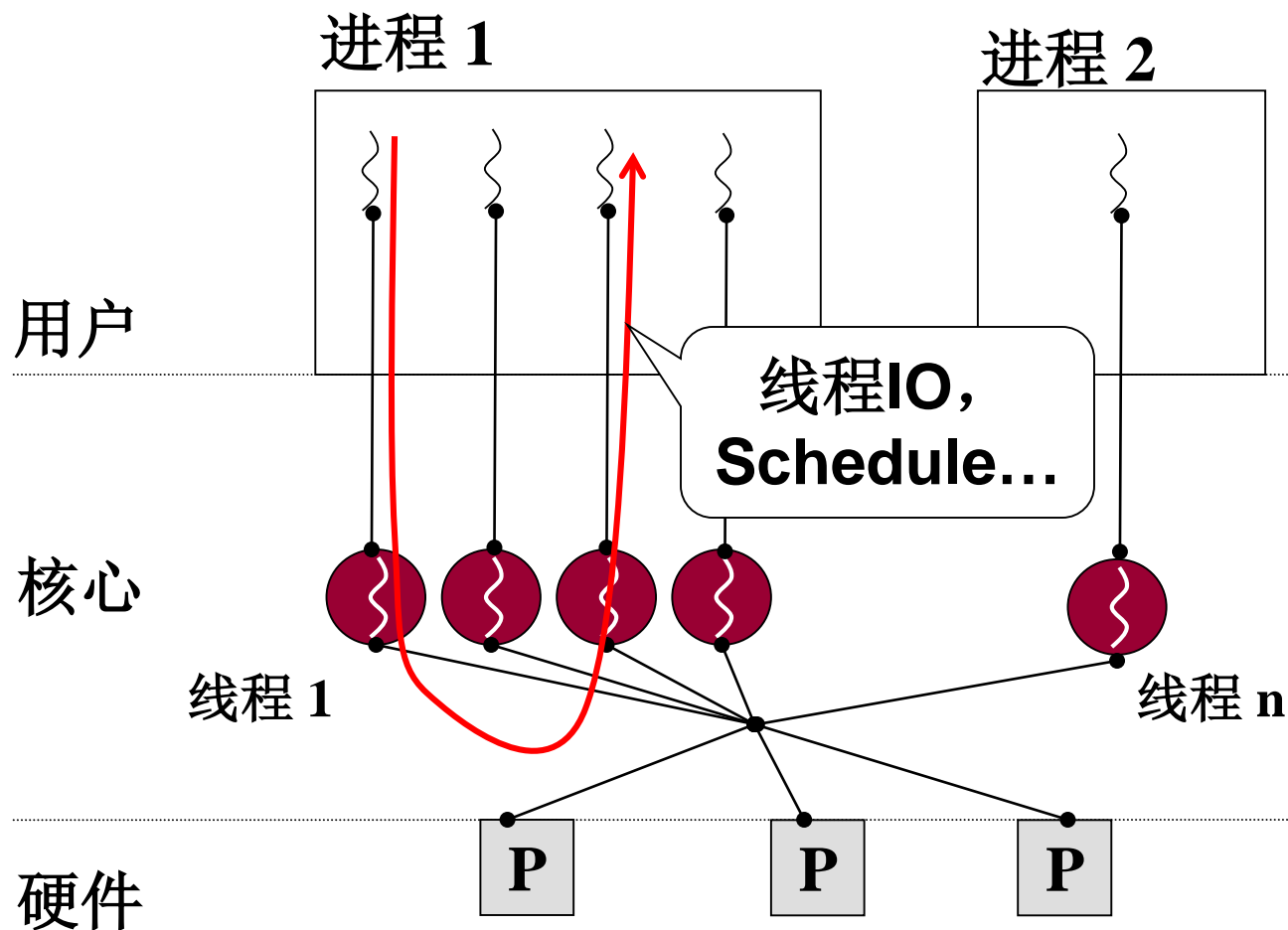
Kernel Threads

授课教师：李治军

lizhijun_os@hit.edu.cn

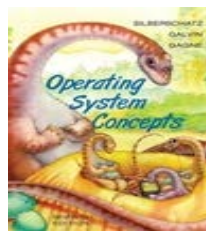
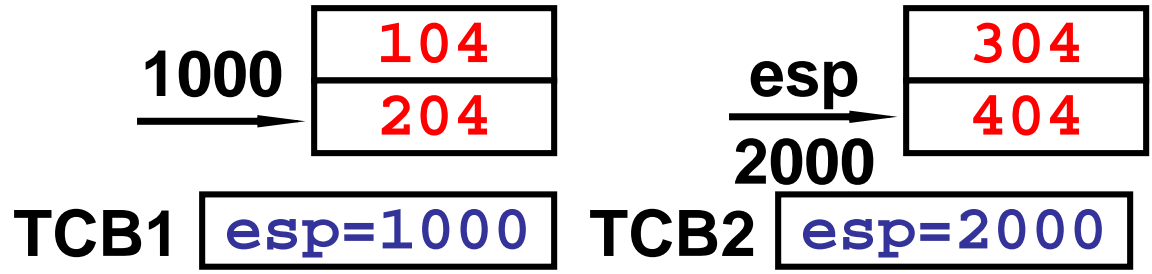
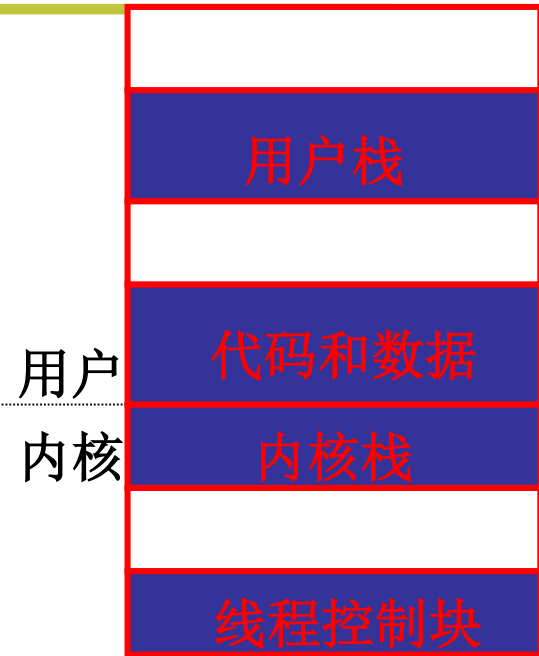
综合楼411室

开始核心级线程

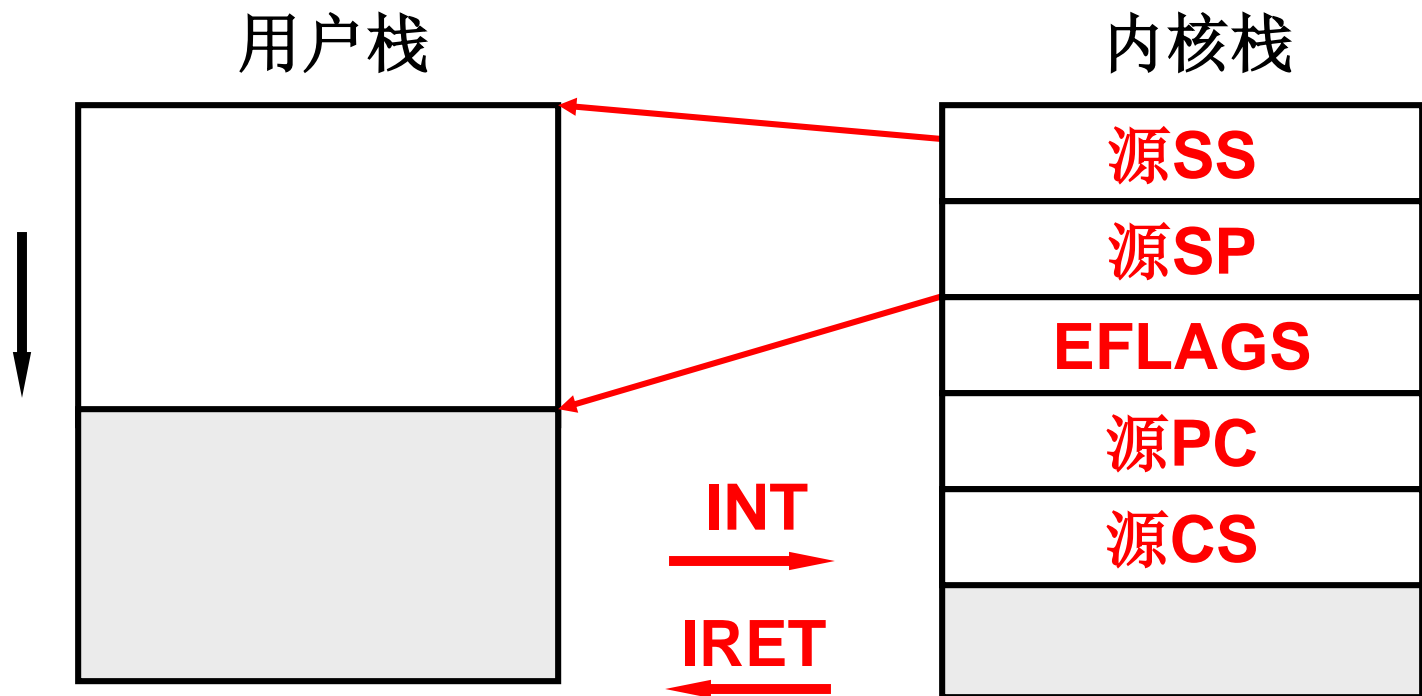


和用户级相比，核心级线程有什么不同？

- ThreadCreate是系统调用，内核管理TCB，内核负责切换线程
- 如何让切换成型？ – 内核栈，TCB
 - 用户栈是否还要用？执行的代码仍然在用户态，还要进行函数调用
 - 一个栈到一套栈；两个栈到两套栈
 - TCB关联内核栈，那用户栈怎么办？



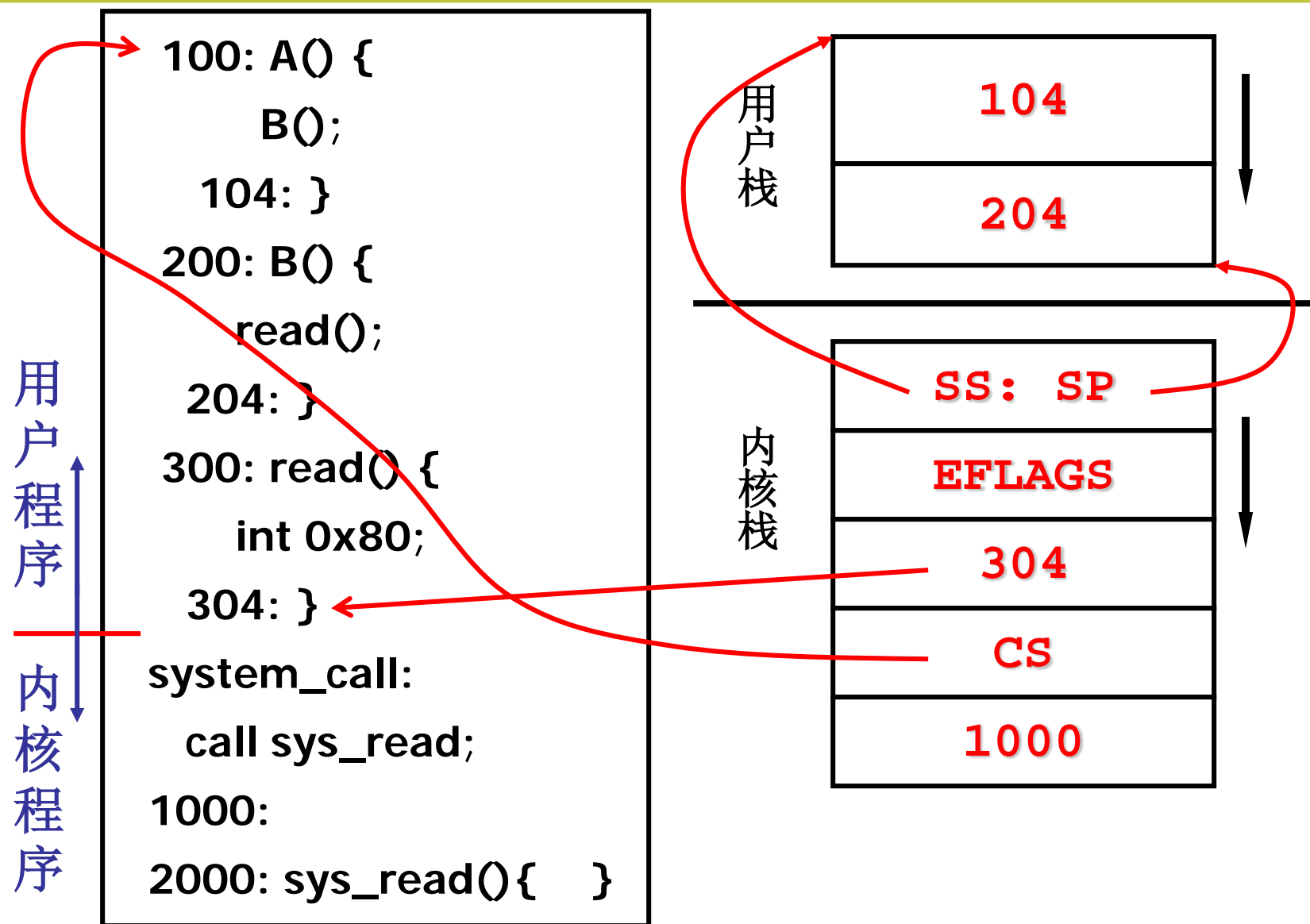
用户栈和内核栈之间的关联



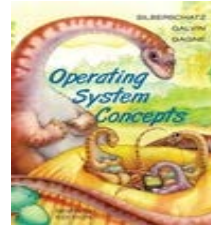
- 所有中断(时钟、外设、INT 指令)都引起上述切换
- 中断(硬件)又一次帮助了操作系统...



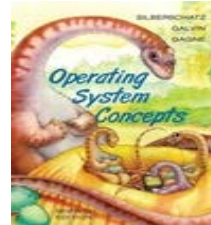
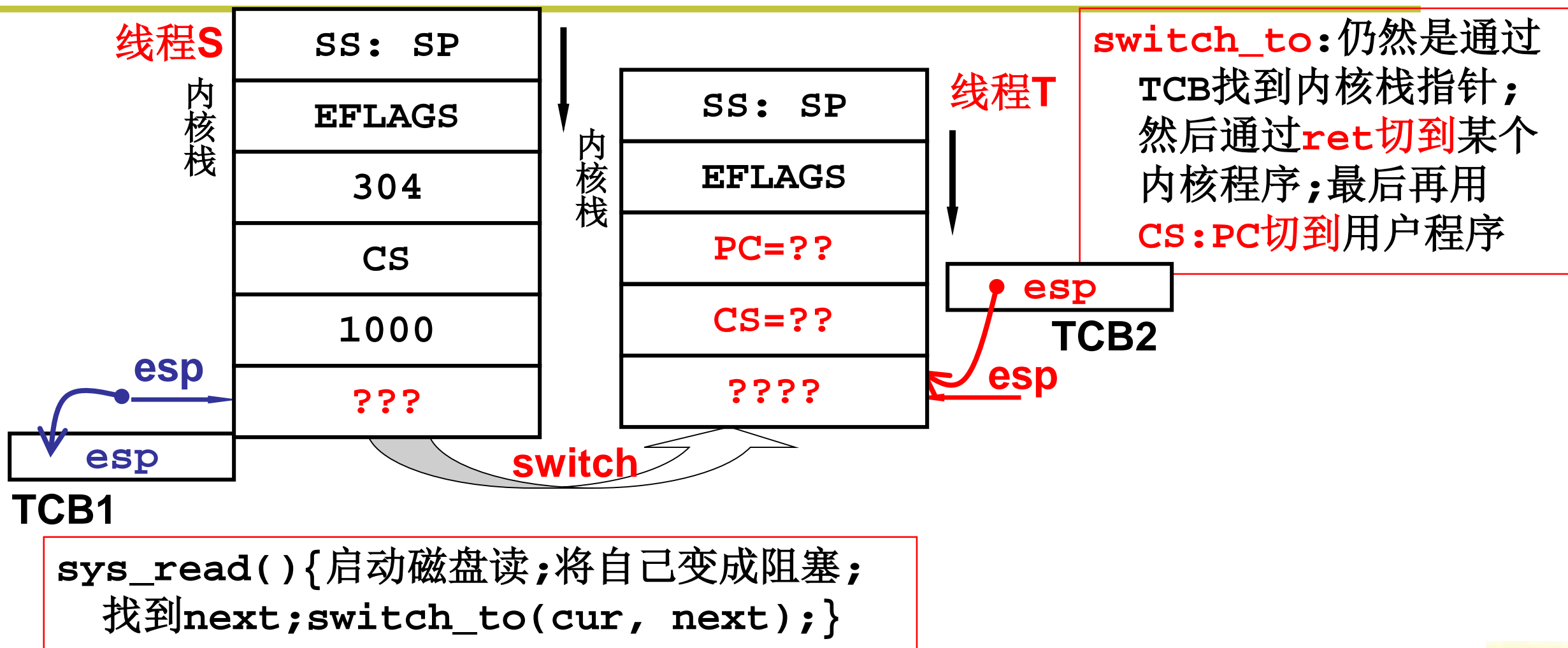
仍然是那个A(), B(), C(), D()...



■ 认真体会从内核返回
(中断返回)时的样子...



开始内核中的切换: `switch_to`



回答上面的问号??, ???, ?????...

```
100: A() { 线程S代码
```

```
...
```

```
int 0x80;
```

```
...
```

```
2000: sys_read(){ }
```

```
500: C() { 线程T代码
```

```
...
```

```
interrupt:
```

```
call sys_xxx;
```

```
3000:
```

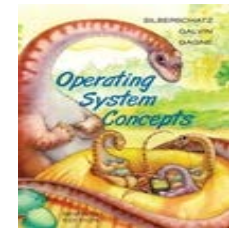
```
4000: sys_xxx(){ }
```

- ??? : sys_read函数的某个地方
- ?? : interrupt之前的某个地方
- ??? : sys_xxx函数中的某个地方

■ 最关键的地方来了: T创建时如何填写??, ?????

- ?? 500, 函数C()的开始地址
- ?????
一段能完成第二级返回的代码, 一段包含iret的代码...

SS: SP
EFLAGS
PC=??
CS=??
????



内核线程switch_to的五段论

第一级切换

中断入口:(进入切换)

```
push ds;... pusha;  
mov ds, 内核段号; ...  
call 中断处理
```

??:

中断处理:(引发切换)

```
启动磁盘读或时钟中断;  
schedule();  
} //ret
```

```
schedule: next=..;  
call switch_to;  
} //ret
```

switch_to:(内核栈切换)

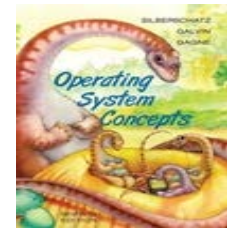
```
TCB[cur].esp=%esp;  
%esp=TCB[next].esp;  
ret
```

中断出口:(第二级切换)

```
popa;...; pop ds;  
iret
```

S、T非同一进程:(地址切换)

```
要首先切换地址映射表;  
TCB[cur].ldtr=%ldtr  
%ldtr=TCB[next].ldtr  
//内存管理
```

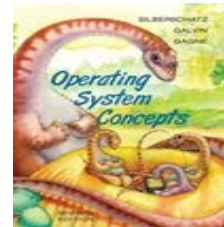
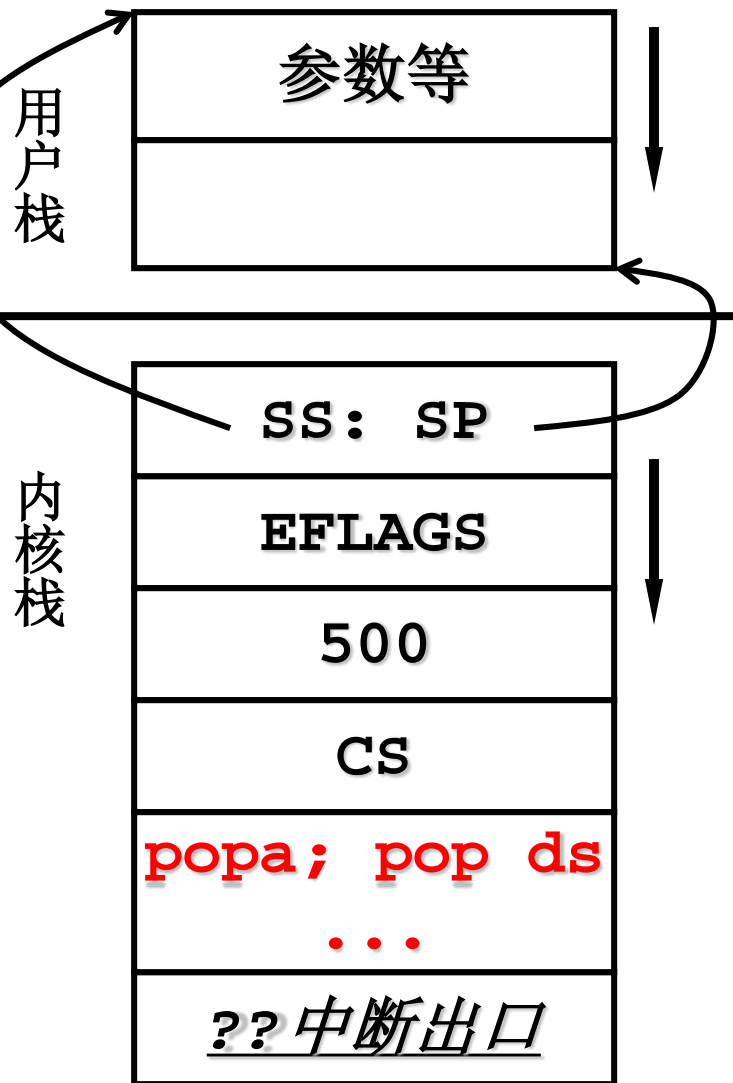


ThreadCreate! 做成那个样子...

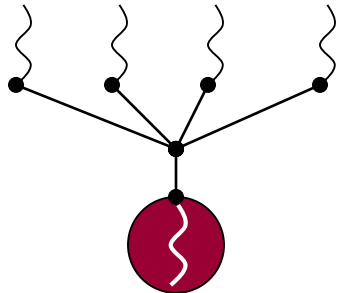
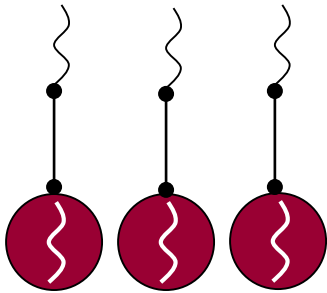
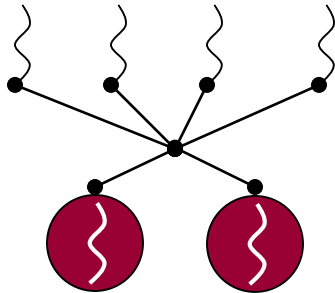
```
500: C() { 线程T代码
```

```
...
```

```
void ThreadCreate(...)  
{  
    TCB tcb=get_free_page();  
    *krlstack = ...;  
    *userstack传入;  
    填写两个stack;  
    tcb.esp=krlstack;  
    tcb.状态=就绪;  
    tcb入队;  
}
```



用户级线程、核心级线程的对比

	用户级线程	核心级线程	用户+核心级
实现模型			
利用多核	差	好	好
并发度	低	高	高
代价	小	大	中
内核改动	无	大	大
用户灵活性	大	小	大

