

# 操作系统

# Operating Systems

## L10 用户级线程

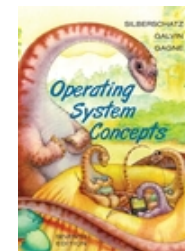
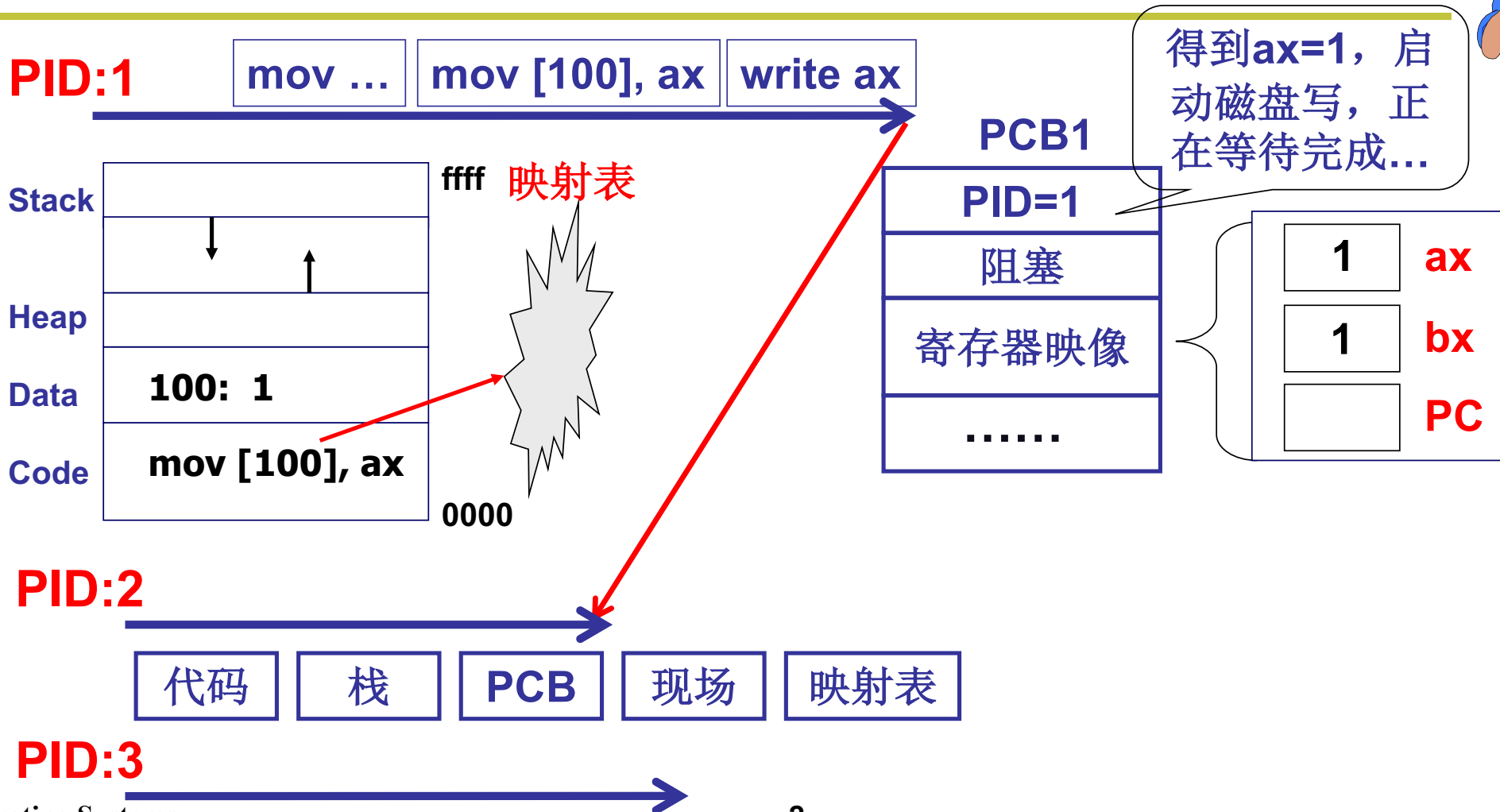
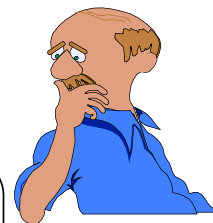
### User Threads

[lizhijun\\_os@hit.edu.cn](mailto:lizhijun_os@hit.edu.cn)

综合楼411室

授课教师：李治军

# 多进程是操作系统的基本图像

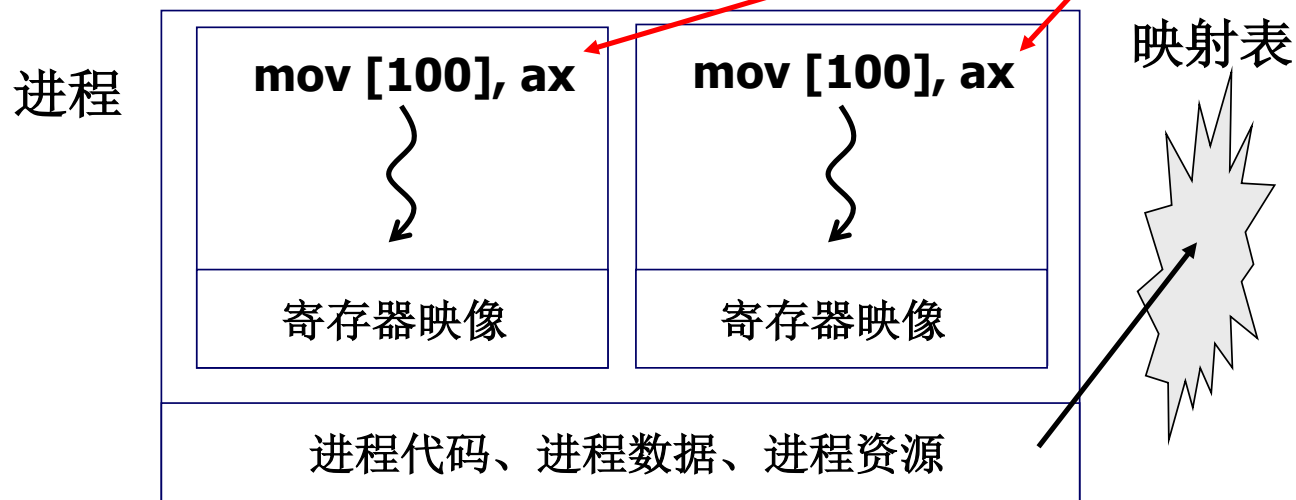


一个进程本身就只对应着主要两个要素：  
CPU各个寄存器的值  
自己的那一部分内存。

## 是否可以资源不动而切换指令序列？

### ■ 进程 = 资源 + 指令执行序列

- 将资源和指令执行分开
- 一个资源 + 多个指令执行序列



- 线程: 保留了并发的优点, 避免了进程切换代价

- 实质就是映射表不变而PC指针变

资源切换相当于将虚拟内存到物理内存的映射关系进行切换。

指令切换就只需要保护PC指针和寄存器即可。

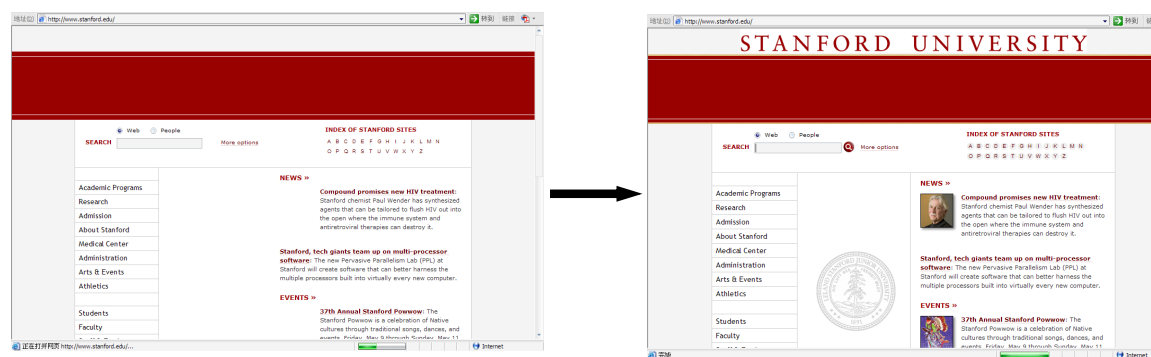
能够在切换的时候  
只进行指令的切换  
而不进行资源的切换?  
这就是线程!



# 多个执行序列+一个地址空间是否实用？

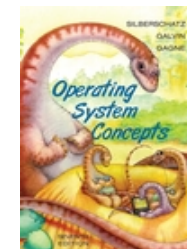
## ■ 一个网页浏览器

- 一个线程用来从服务器接收数据
- 一个线程用来显示文本
- 一个线程用来处理图片(如解压缩)
- 一个线程用来显示图片



## ■ 这些线程要共享资源吗？

- 接收数据放在100处，显示时要读..
- 所有的文本、图片都显示在一个屏幕上

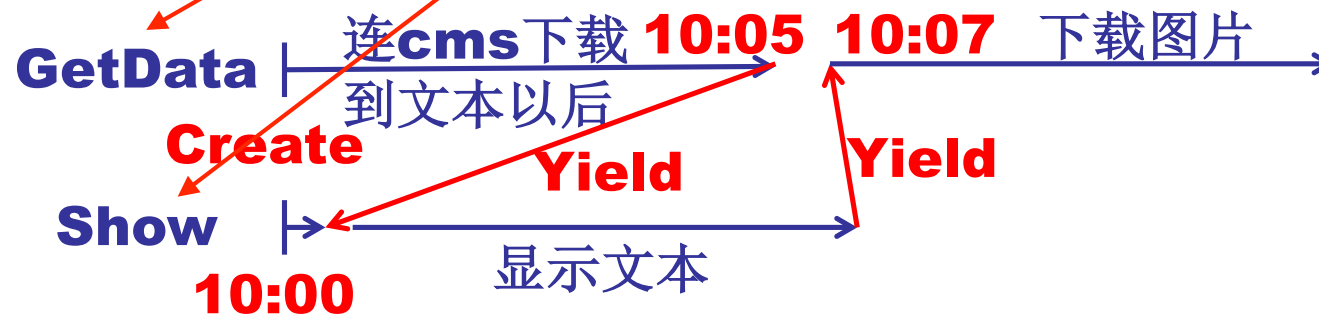


# 开始实现这个浏览器...

```
void WebExplorer()
{
    char URL[] = "http://cms.hit.edu.cn";
    char buffer[1000];
    pthread_create(..., GetData, URL, buffer);
    pthread_create(..., Show, buffer);
}
```

```
void GetData(char *URL, char *p) {...};
void Show(char *p) {...};
```

用户自己可以实现  
线程切换。一段时间之后调用一个函数  
**yield**主动释放，  
切换到另一个线程执行。



# Create? Yield?

## ■ 核心是Yield...

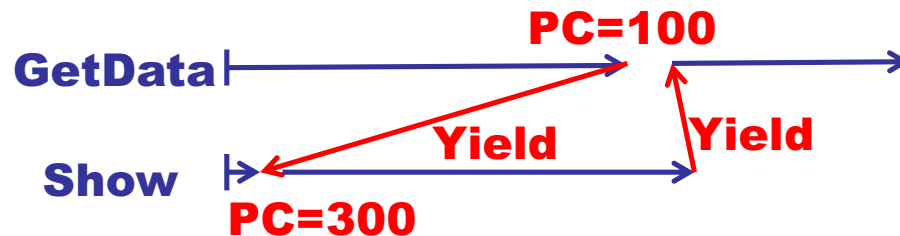
- 能切换了就知道切换时需要是个什么样子
- Create就是要制造出第一次切换时应该的样子

样子弄明白了，剩下的就是写程序实现这个样子了...

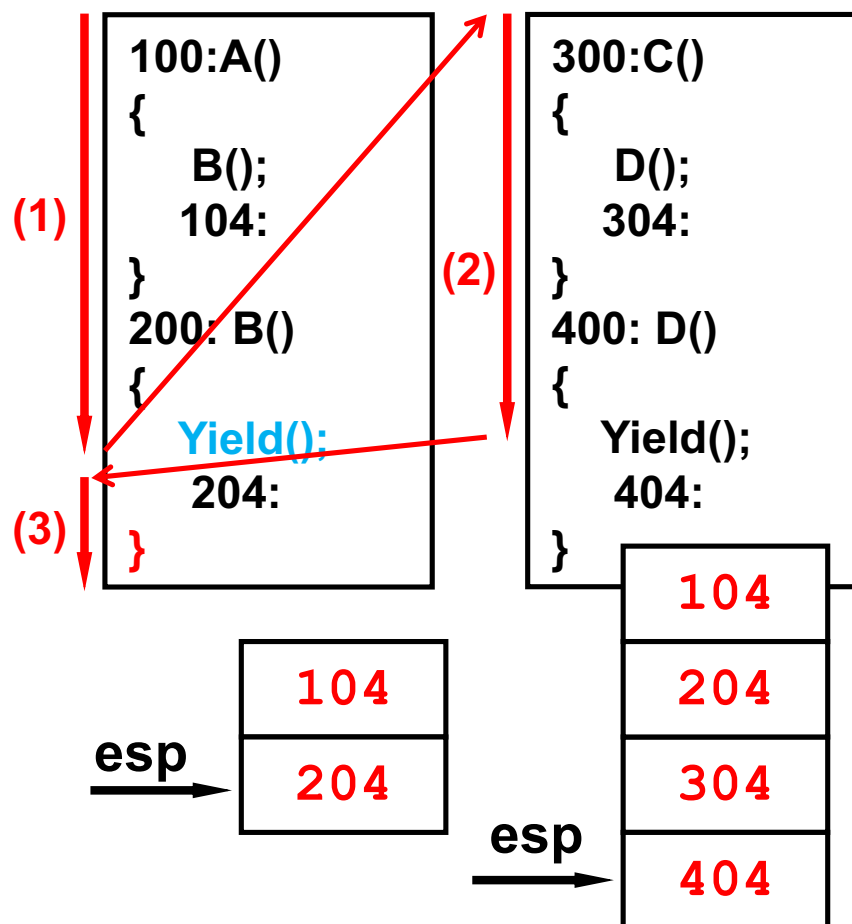
```
100:A()
{
    B();
    104:
}
200: B()
{
    Yield();
    204:
}
```

```
300:C()
{
    D();
    304:
}
400: D()
{
    Yield();
    404:
}
```

## ■ 仔细看Yield，就是100跳到300



## 两个执行序列与一个栈...



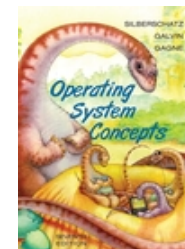
```
void Yield()    void Yield()
{
    找到300;    找到 ?;
    jmp 300;    jmp ?;
}
```

■ (3)再往下执行会怎么样?

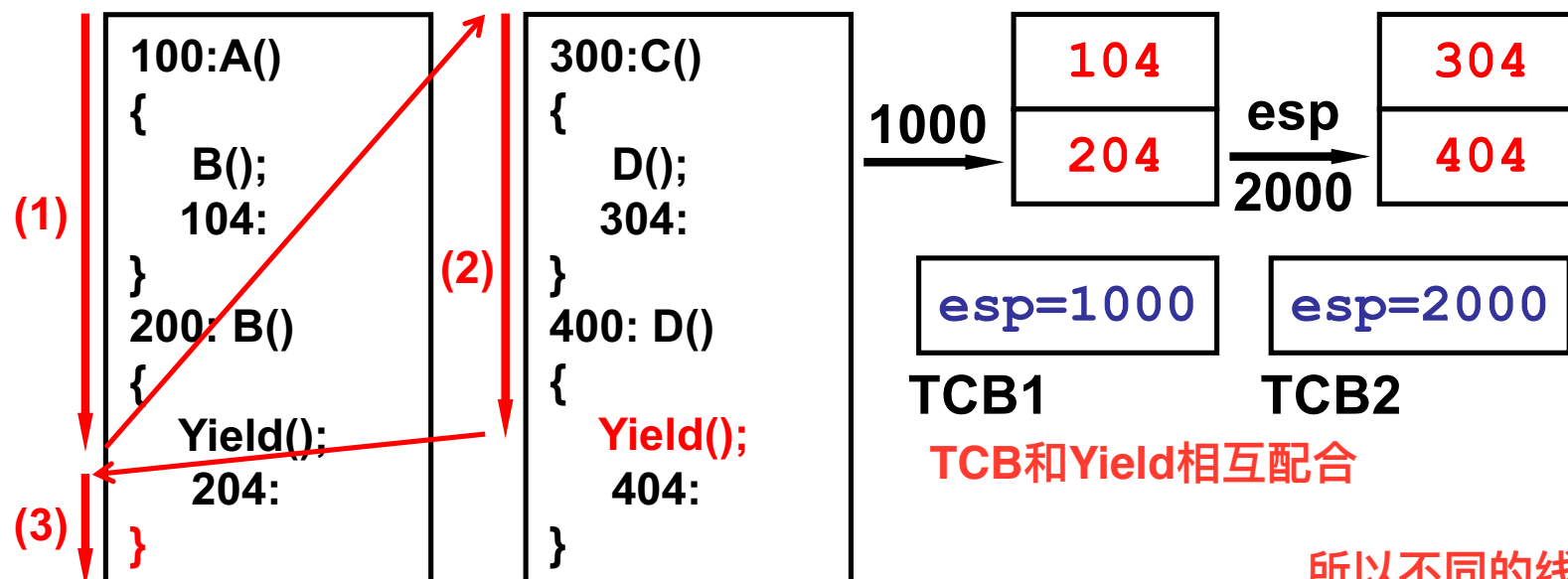
■ 问题怎么解决?

为什么?

两个线程共用了一个栈  
会导致返回地址等发生冲突



# 从一个栈到两个栈...



- Yield切换要先切换栈，然后...

```
void Yield() {  
    TCB2.esp=esp;  
    esp=TCB1.esp;  
    jmp 204; 应该去掉  
}
```

不需要切换PC，只需要换栈即可。

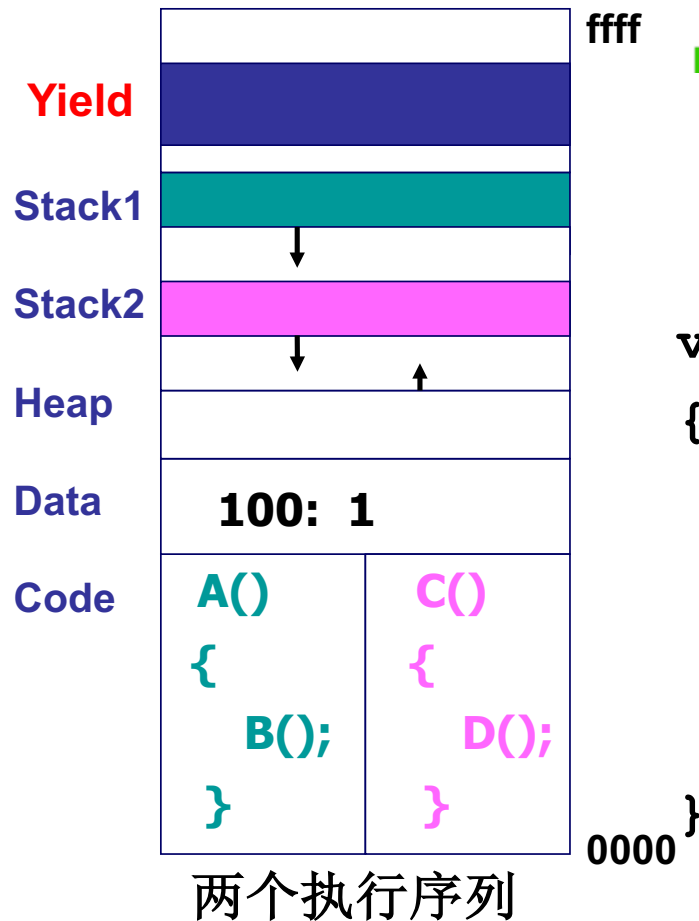
所以不同的线程需要有自己的栈

- (3)再往下执行会怎么样?
- 204是调用Yield()才压栈的...





# 两个线程的样子：两个TCB、两个栈、切换的PC在栈中

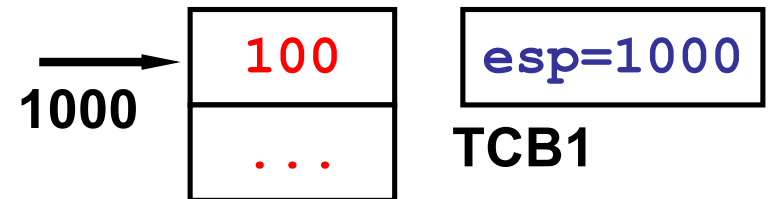


■ ThreadCreate的核心就是用程序做出这三样东西

```
void ThreadCreate(A)
{
```

```

    TCB *tcb=malloc();
    *stack=malloc();
    *stack = A;//100
    tcb.esp=stack;
}
```



# 将所有东西组合在一起.....

```
void WebExplorer() //main()
{ ThreadCreate(GetData, URL, buffer); ...
  while(1) Yield(); }
```

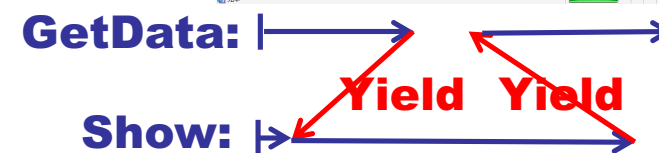
```
void GetData(char *URL, char *p) {
  连接URL; 下载; Yield(); ... }
```

```
void ThreadCreate(func, arg1) {
  申请栈; 申请TCB; func等入栈; 关联TCB与栈; ... }
```

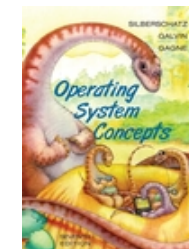
```
void Yield() { 压入现场; esp放在当前TCB
  中; Next(); 从下个TCB取出esp; 弹栈切换线程; }
```

调度函数，对系统影响很大，如可优先调度show!

■ **gcc -o explorer get.c yield.c ... 或 gcc get.c.. -lthread**



■ **GetData**下载到文本时会调用**Yield()**...



当用户线程进行IO时，进入到内核态，  
发生阻塞，但是操作系统看不到它是个线程，所以会切换到其他进程而不是  
这个进程的其他线程，所以浏览器还是黑，而不是切换到show线程

## 为什么说用户级线程——Yield是用户程序

■ 如果进程的某个线程进入内核并阻塞，则...

### GetData

连接URL发起请求；

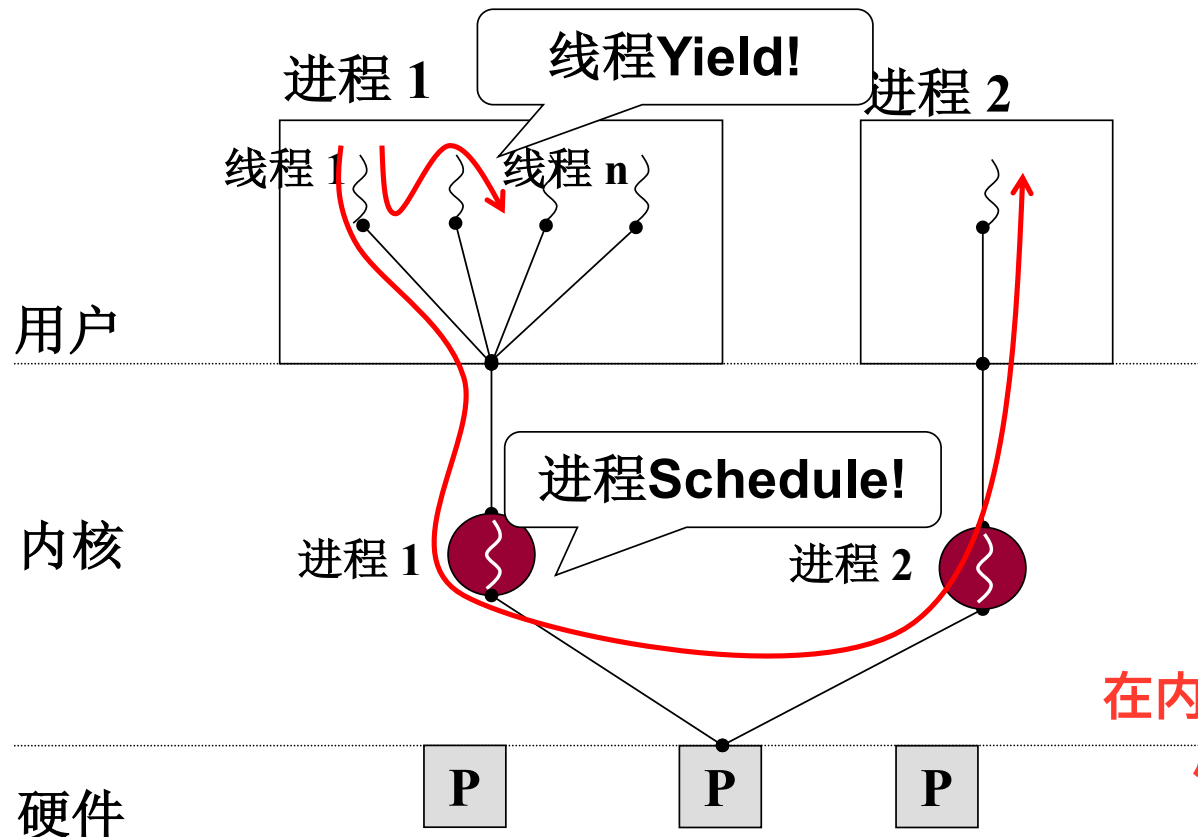
等待网卡IO...

进程阻塞

### Show

显示文本和链接；

...;



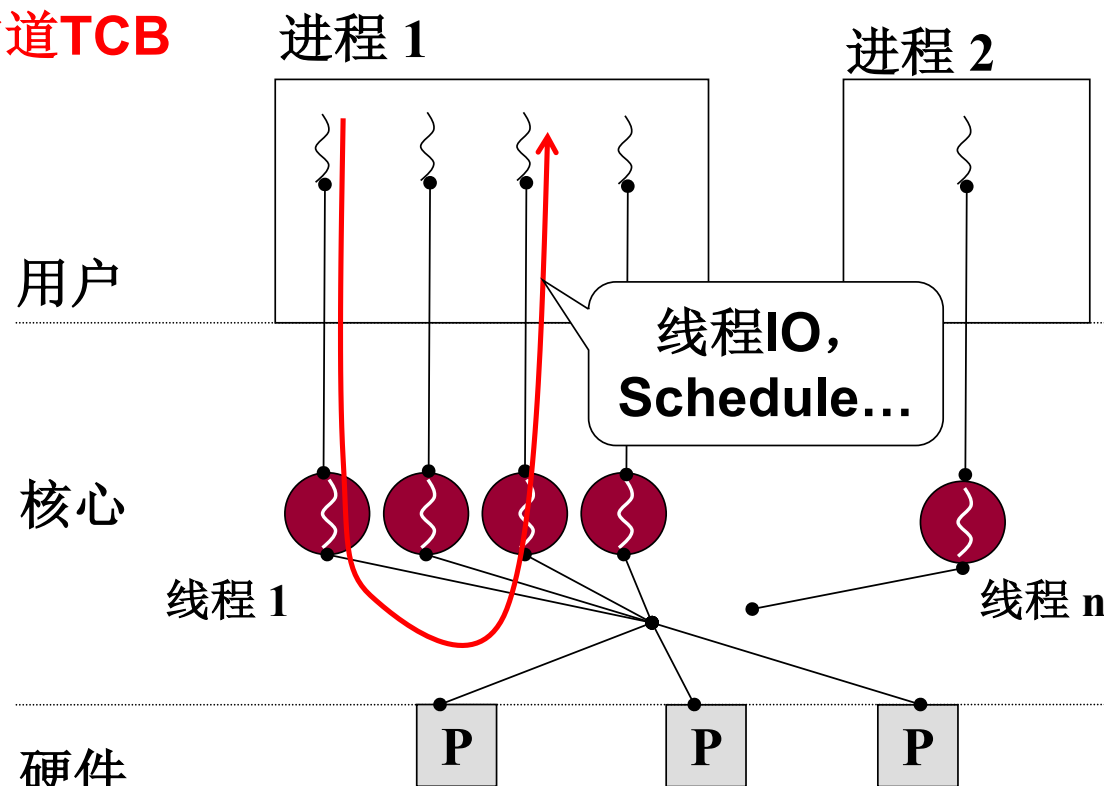
用户级线程的切换  
只是在用户态之间切换，  
操作系统完全感知不到它的存在  
在内核中阻塞了，就会被切换到其他进程，  
尽管这个时候show线程可以执行。



核心级线程是在内核态中做的，操作系统可见

## 核心级线程      核心级线程和用户级线程区别，哪个快？

- **ThreadCreate**是系统调用，会进入内核，内核知道**TCB**



- `gcc -o explorer explorer.c yield.c ...`

- 内核级线程`gcc -o explorer explorer.c...`; **ThreadCreate**是系统调用；**Yield()**用户不可见，调度点由系统决定

用户级线程可以由用户实现  
各个线程之间的调度，  
内核级线程完全有操作系统进行调度，  
用户无法控制

