

操作系统

Operating Systems

L13 操作系统的那棵“树”

L13 The Tree of OS

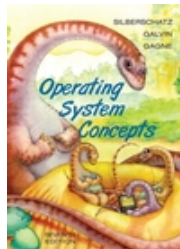
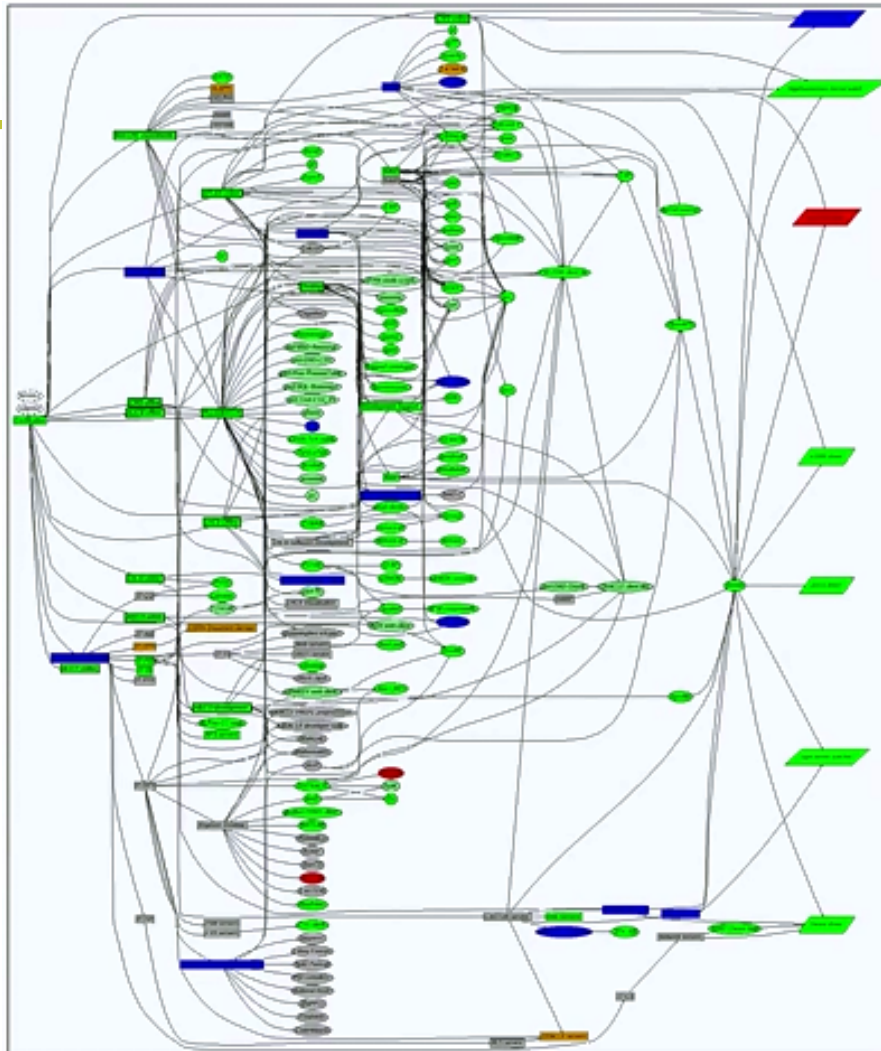
lizhijun_os@hit.edu.cn

授课教师：李治军

综合楼411室

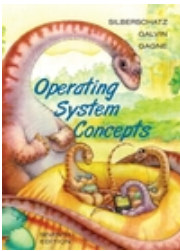
Linux Kernel Source Tree

Linus Torvalds

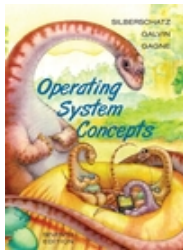


**The mind is not a vessel that
needs filling, but wood that
needs igniting!**

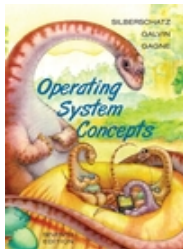
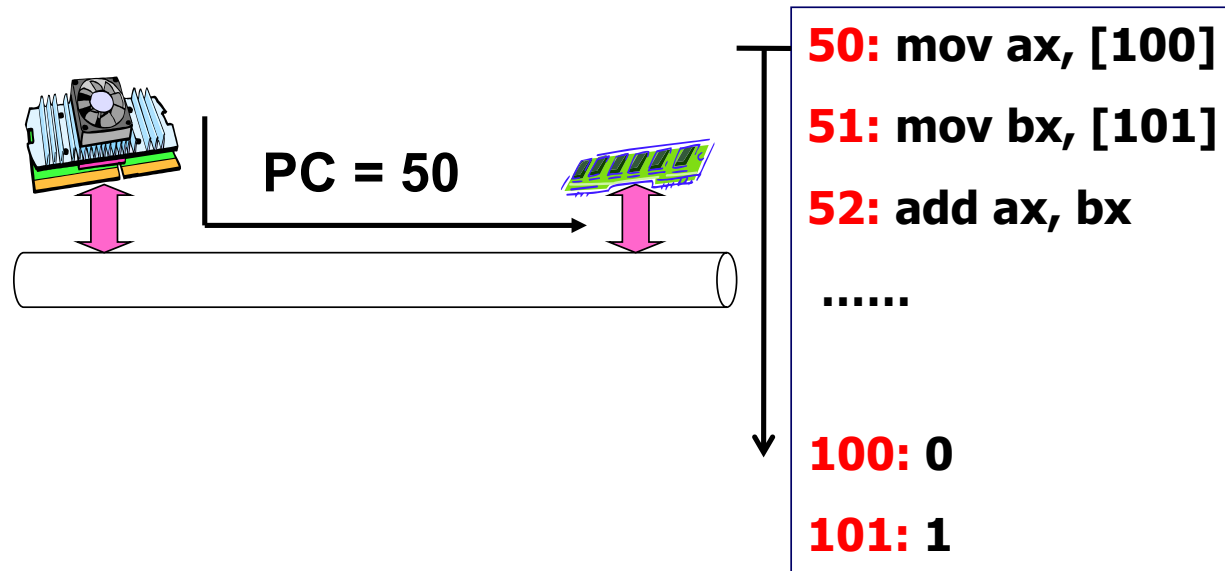
Plutarch quote



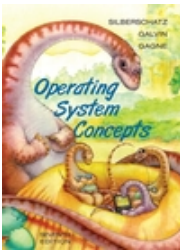
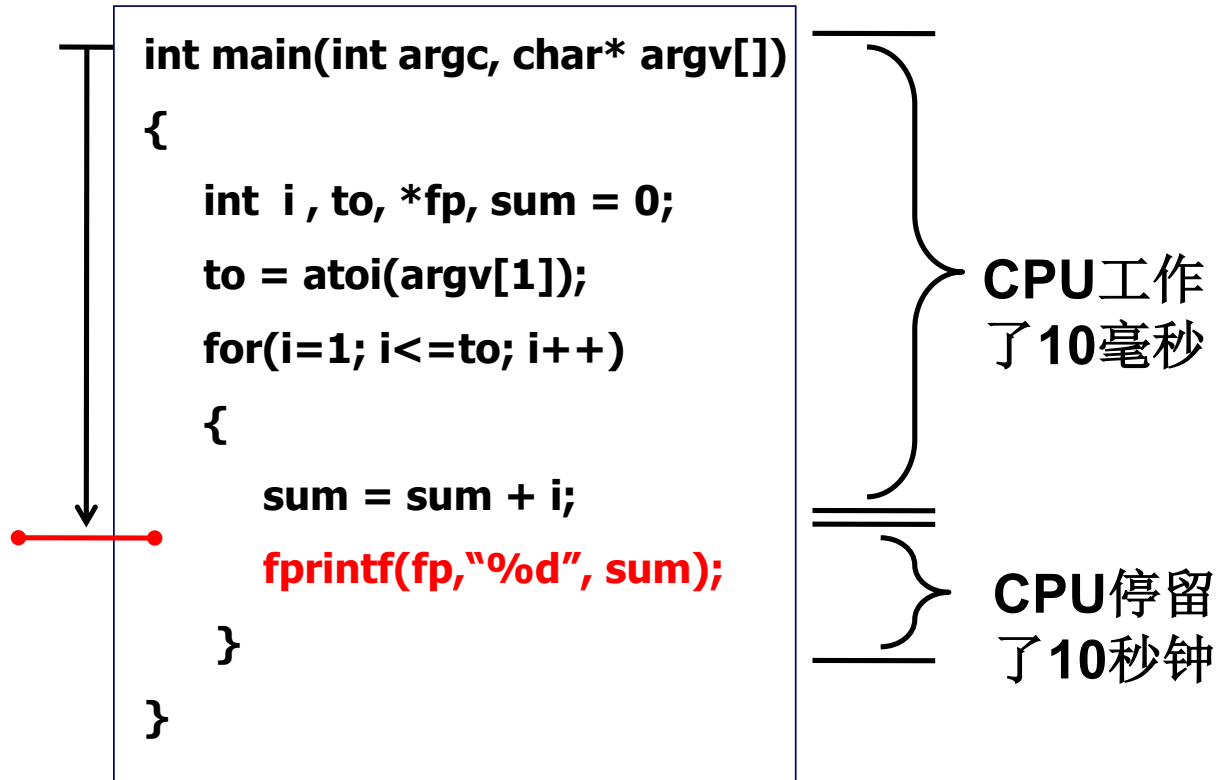
IDEA



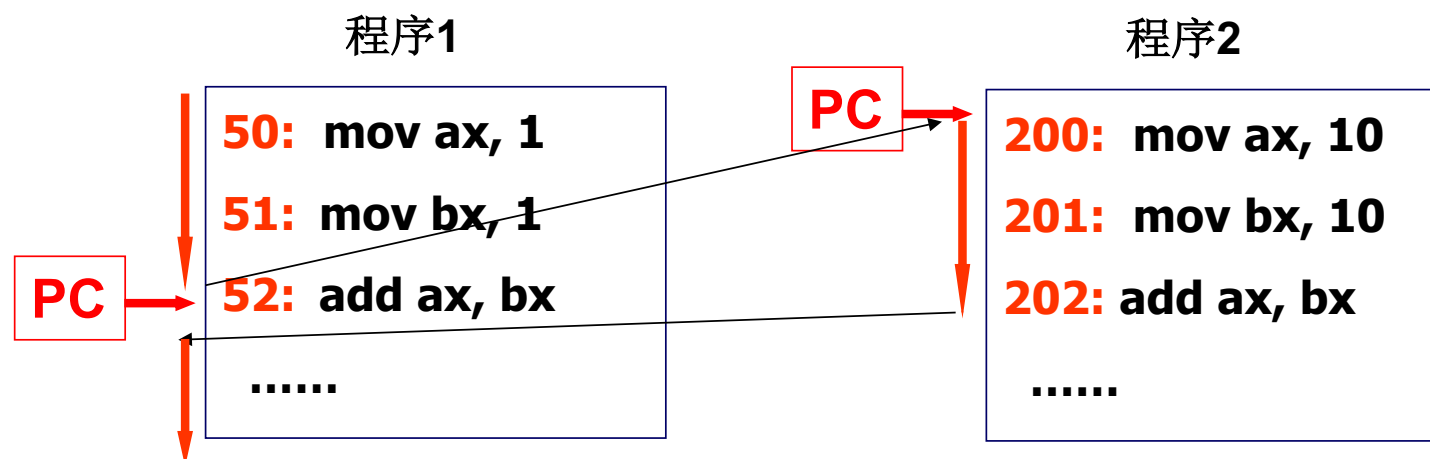
运转CPU



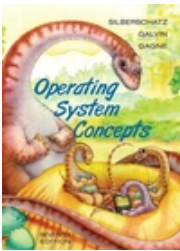
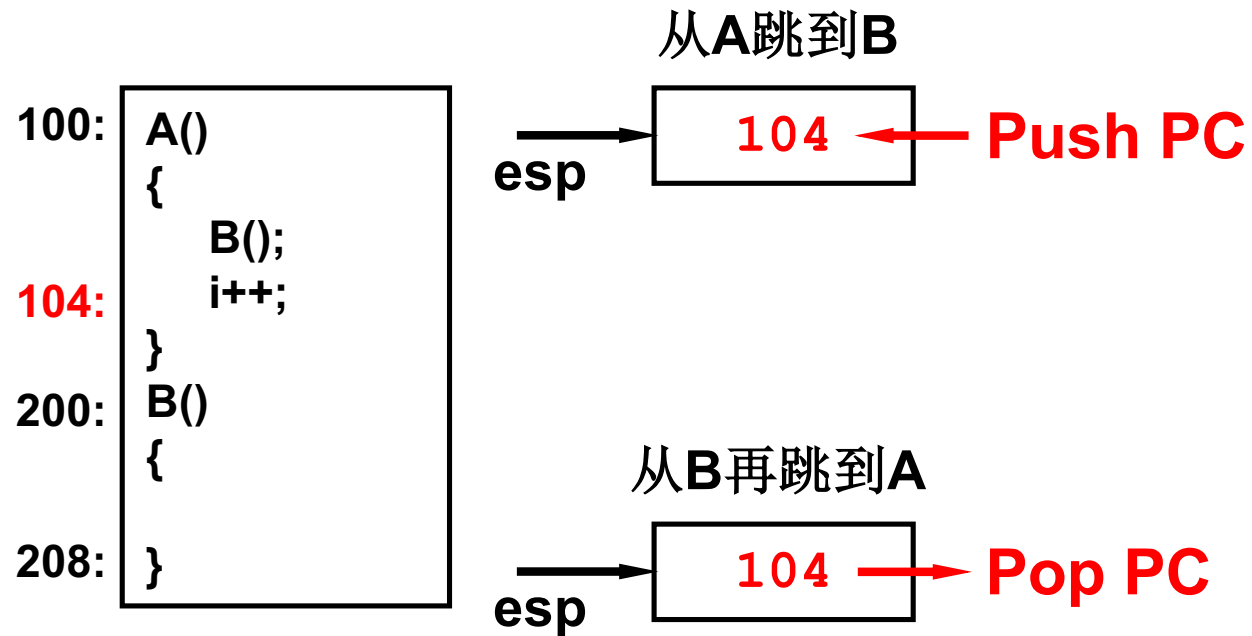
CPU没有好好运转



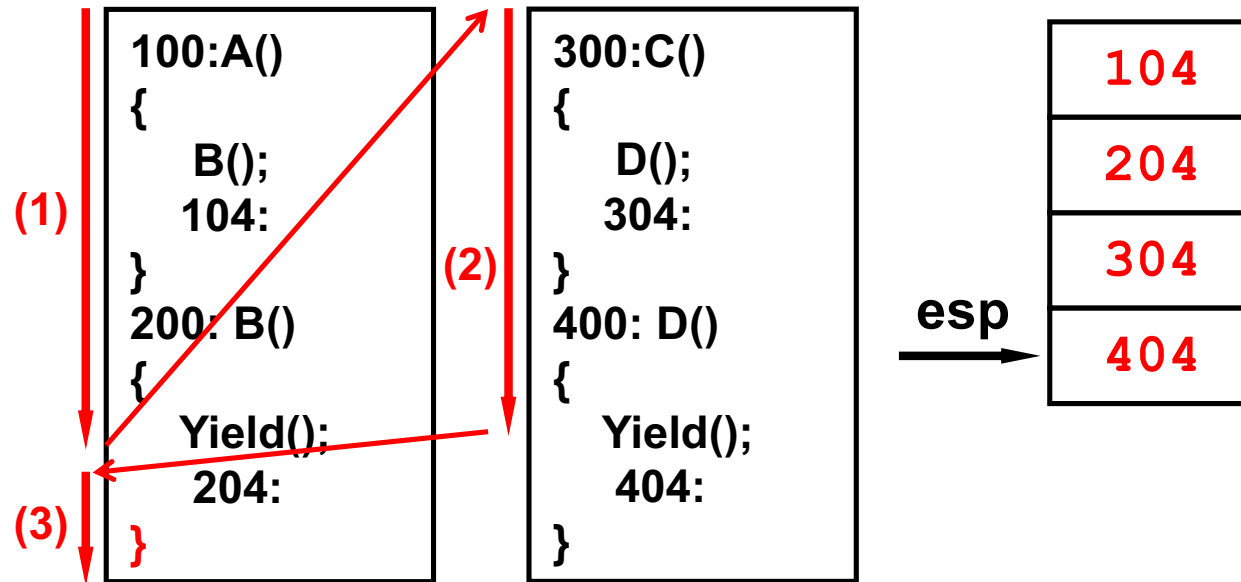
得让CPU好好运转



从A跳到B我们并不陌生



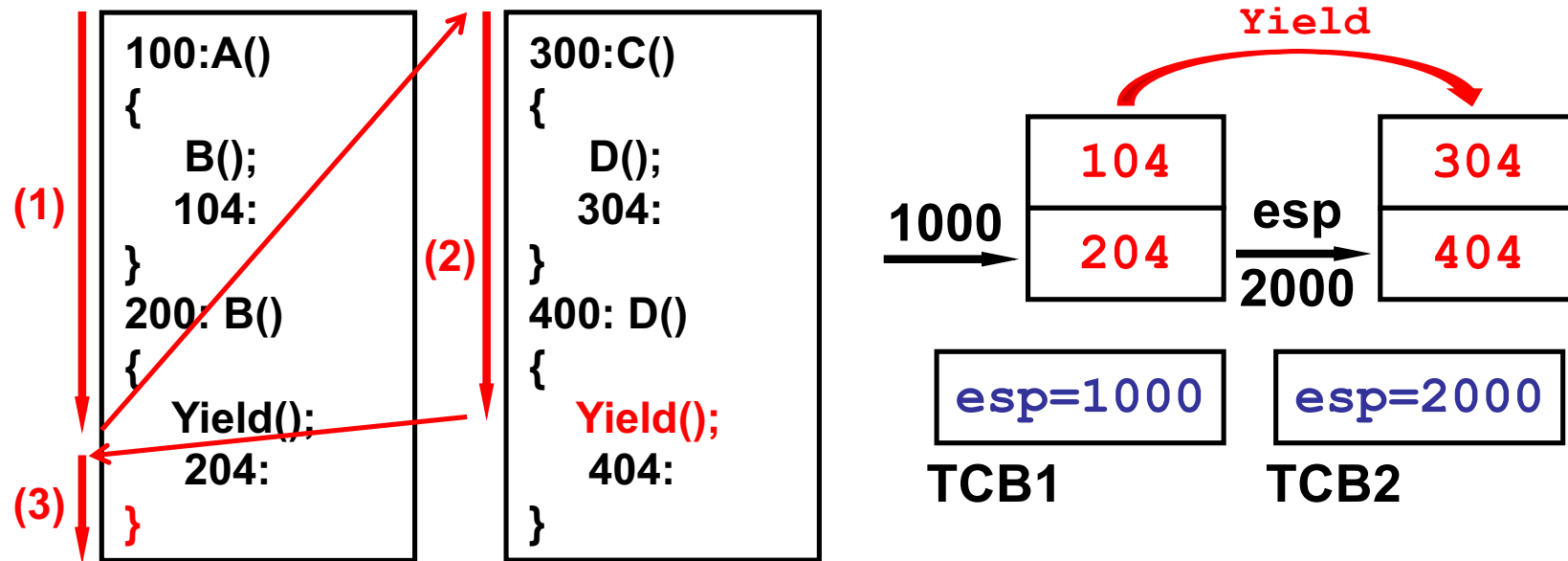
一个栈+Yield造成的混乱



面对这样的栈你怎么可能从**B**顺利的回**A**?



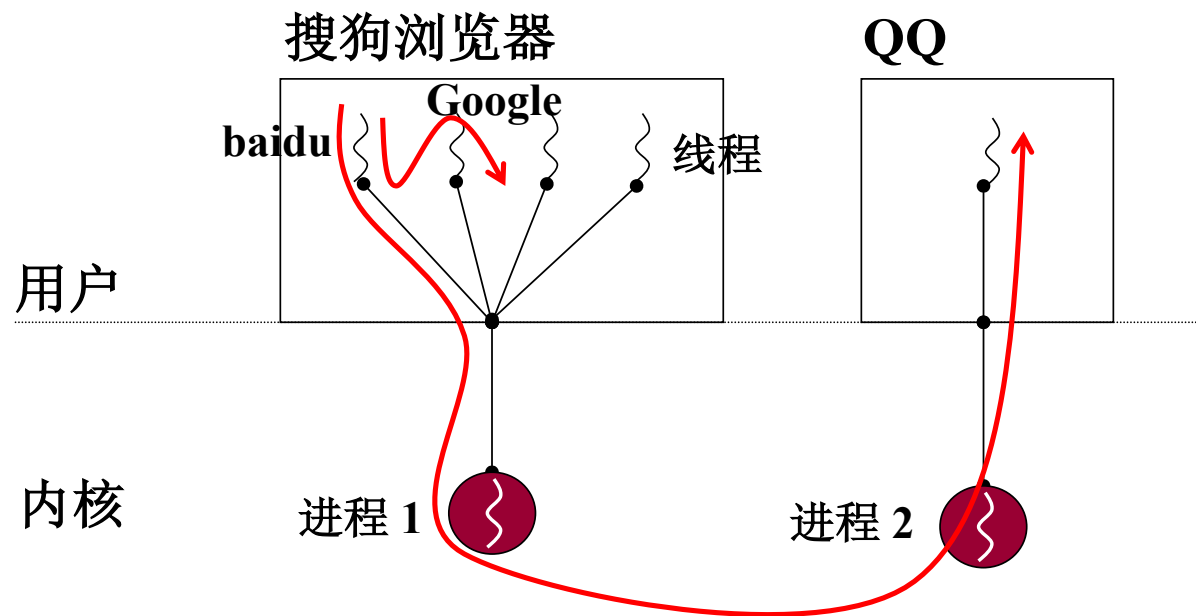
两个栈+两个用户TCB



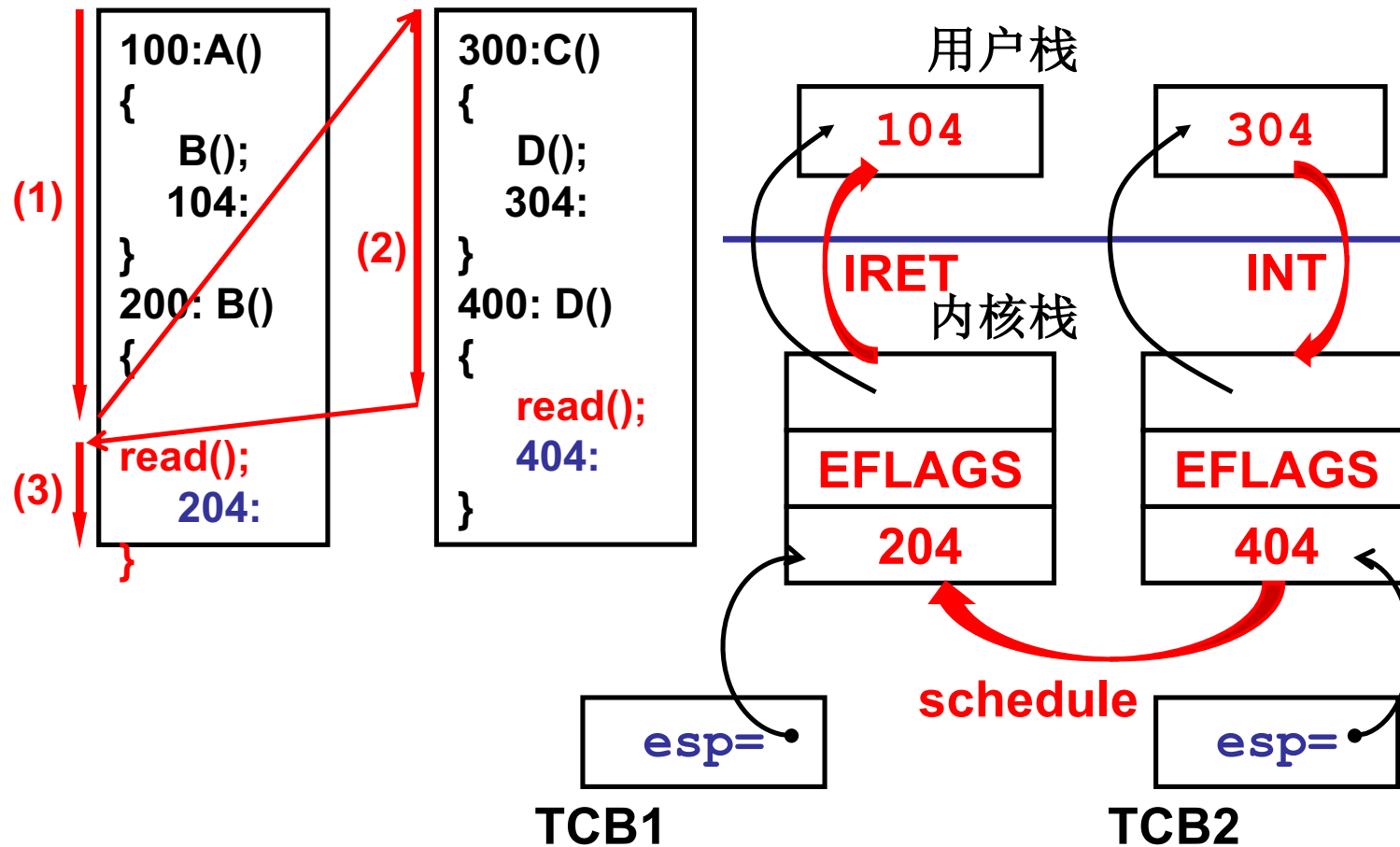
`Yield()` 找到下一个TCB→找到新的栈
→切到新的栈



一直在用户态那怎么行？

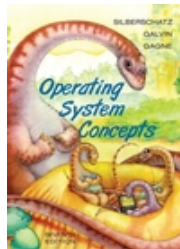
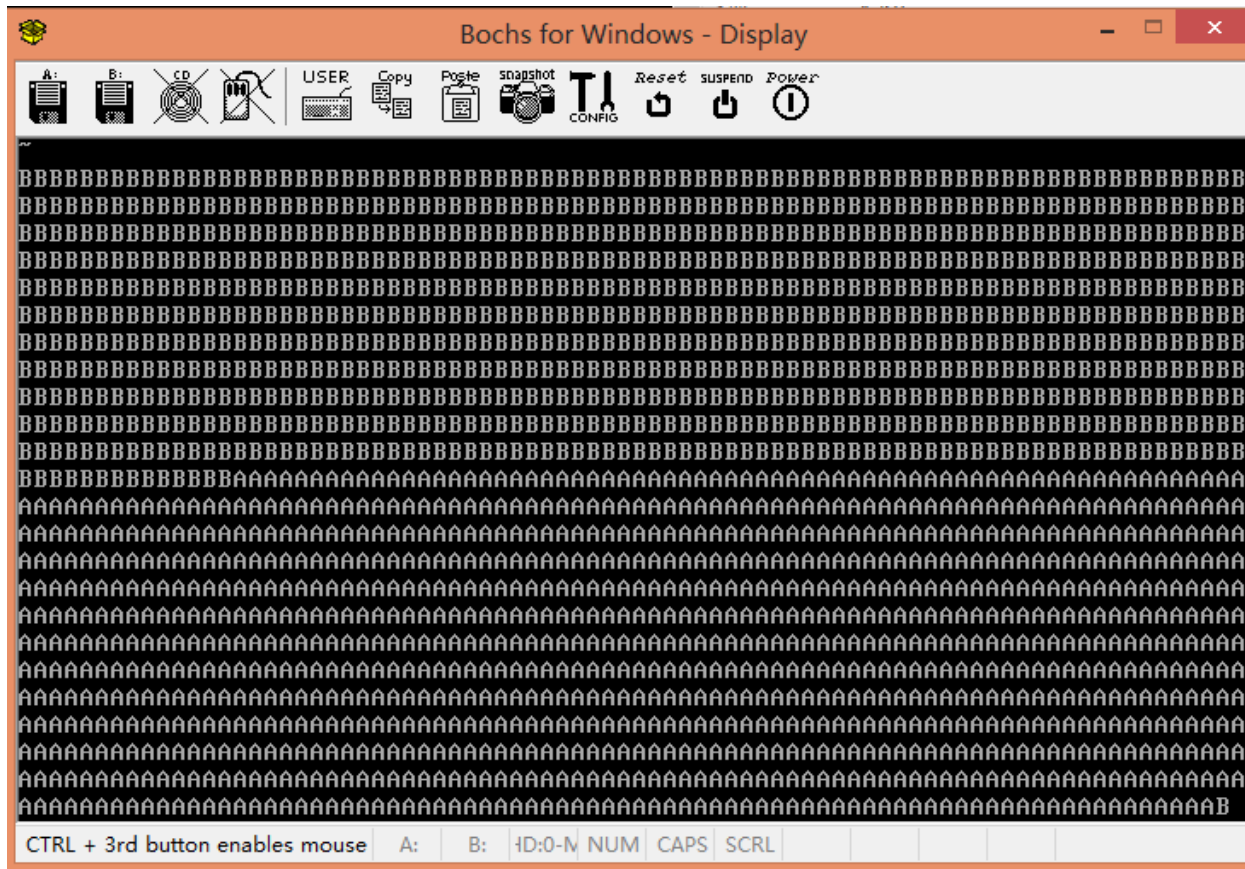


引入内核栈的切换



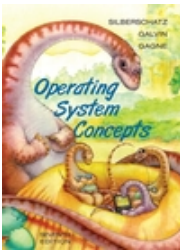
到实现idea的时候了

从一个简单、清晰、明确的目标开始



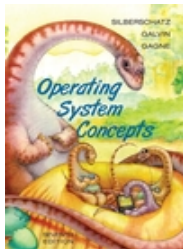
从用户代码开始

```
main()                AB.c
{
    if(!fork()) {while(1)printf("A");}
    if(!fork()) {while(1)printf("B");}
    wait();
}
```



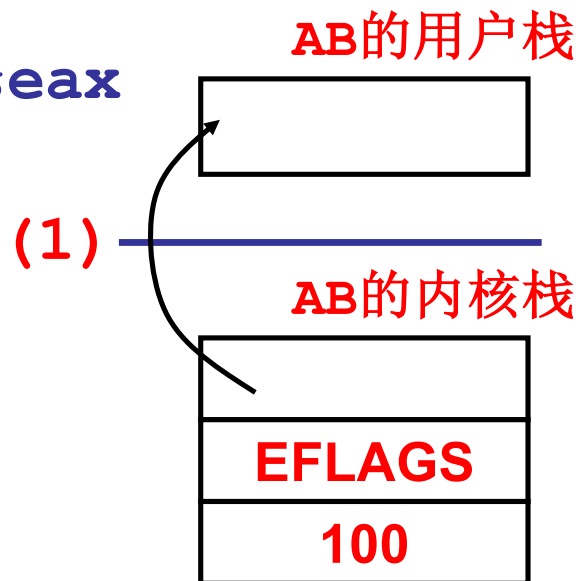
程序是什么?就是人的思维的C表达

```
main() {                                把自己变成计算机跑一遍
    mov __NR_fork, %eax
    int 0x80                             AB.c
100: mov %eax, res
    cmpl res, 0
    jne 208
200: printf("A")
    jmp 200
208: ...
304: wait()
}
```



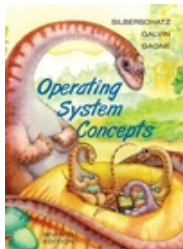
INT进入内核

```
main() {  
    mov __NR_fork, %eax  
    int 0x80  
100: mov %eax, res  
    cml res, 0
```



```
(2) set_system_gate(0x80, &system_  
call);
```

```
(3) system_call:  
    call sys_call_table(, %eax, 4)
```



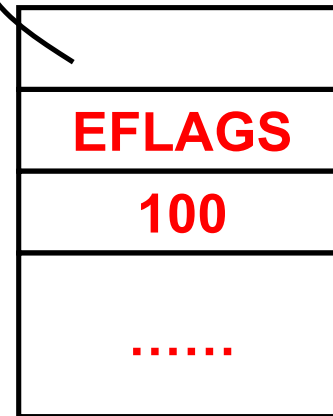
开始sys_fork

```
sys_fork:  
    pushl ...  
    call copy_process  
    ret
```

AB的用户栈

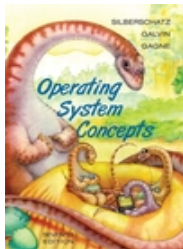
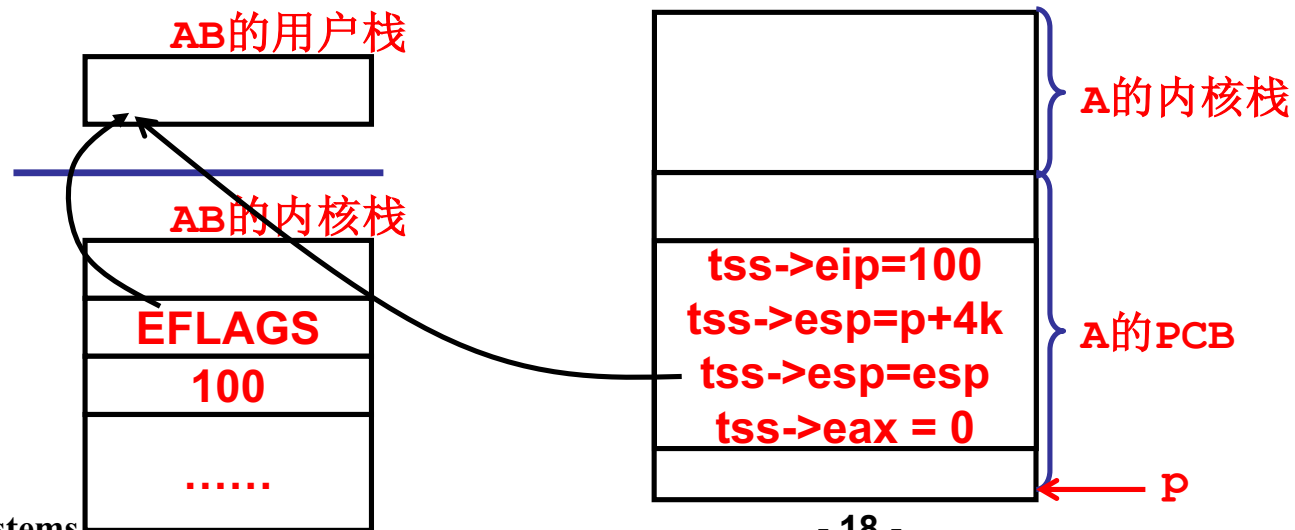


AB的内核栈



开始copy_process

```
copy_process(...long eip,=100 ...) {  
    p = (PCB *)get_free_page();  
    p -> tss.esp0 = p+4k;  
    p -> tss.esp = esp;  
    p -> tss.eax=0; p -> tss.eip = eip;...}
```



开始返回...

copy_process() { ... } //ret到哪里?

sys_fork:

... call copy_process ...

ret //到哪里?

system_call:

...call sys_call_table(,%eax,4)

cmpl \$0,state(current)

jne reschedule

iret //到哪里?

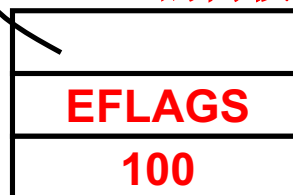
main: int 0x80

100: mov %eax, res cmpl res,0

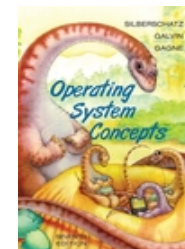
AB的用户栈



AB的内核栈



```
main()
{
    if(!fork()){while(1)printf("A");}
    if(!fork()){while(1)printf("B");}
    wait();
}
```

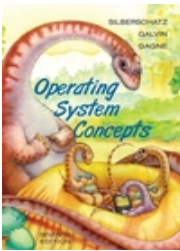
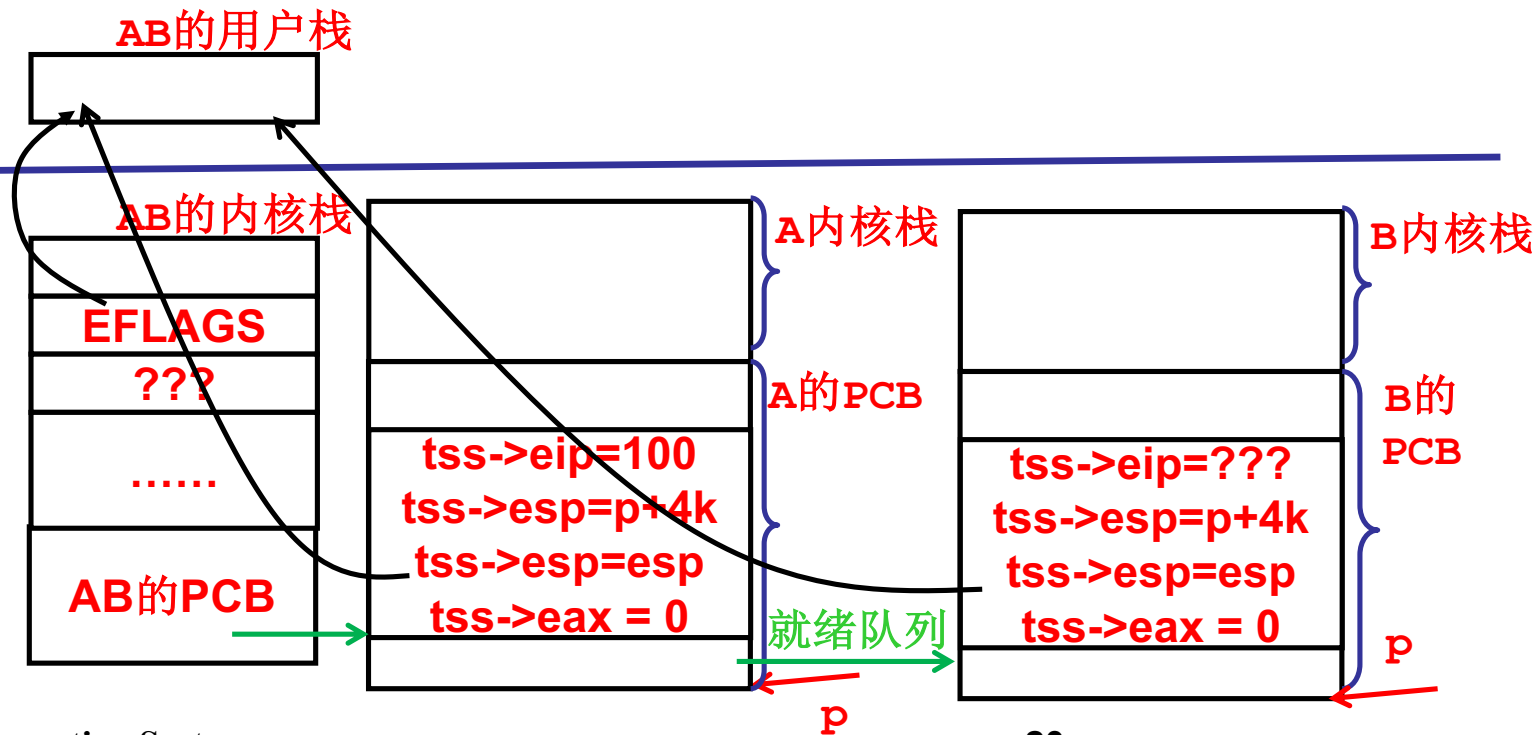


main继续执行，现在 we 有了什么？

```
main() {
```

又一个fork;再一次int 0x80;

在内核中再产生一个PCB和内核栈...



main继续，到了哪里？

```
main() {  
    ... wait(); //又是mov __NR_wait  
                int 0x80  
  
system_call:  
    call sys_waitpid  
  
sys_waitpid() //exit.c中  
    current->state=TASK_INTERRUPTIBLE;  
    schedule();  
}
```



schedule

```
schedule() {  
    if ((*p)->state == TASK_RUNNING &&  
        (*p)->counter > c)  
        c = (*p)->counter, next = i;  
    ...  
    switch_to(next); }
```



switch_to切换

AB的用户栈

AB的内核栈

A内核栈

B内核栈

A的PCB

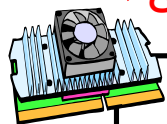
B的PCB

AB的PCB

current

next

就绪队列



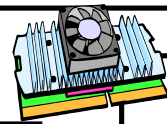
eax
ebx
esp
esp0
eip

AB的PCB中
的tss

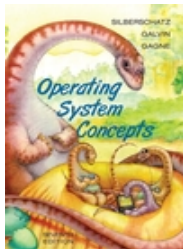
current

A的PCB中
的tss

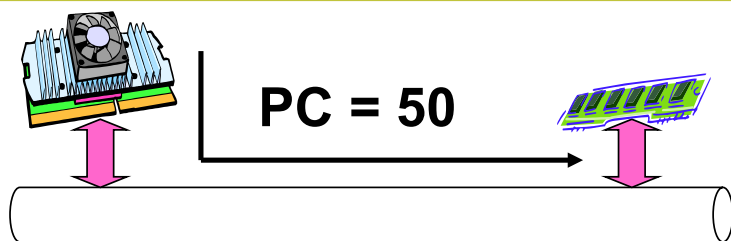
next



eax
ebx
esp
esp0
eip



接下来会怎么样？



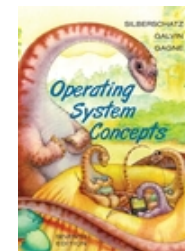
eip=100 eax=0 esp=...

这就是**线程**，代码始终不曾变过，一直在内存中...

fork的代码也不曾变过，所以实验8可以通过**fork** “**伪造**”

进程A开始不断的打**A**...

```
main() {  
    mov __NR_fork, %eax  
    int 0x80  
100: mov %eax, res  
    cmpl res, 0  
    jne 208  
200: printf("A")  
    jmp 200  
208: ...  
304: wait()  
}
```



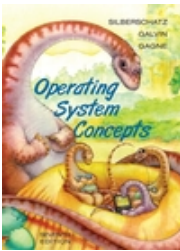
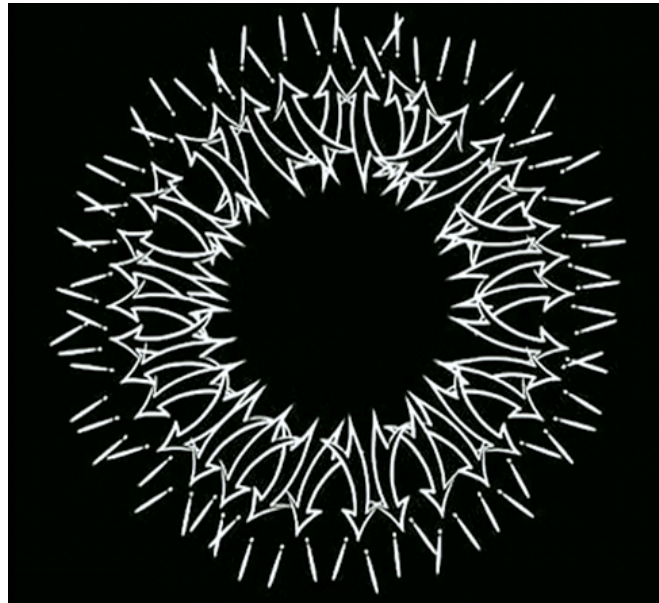
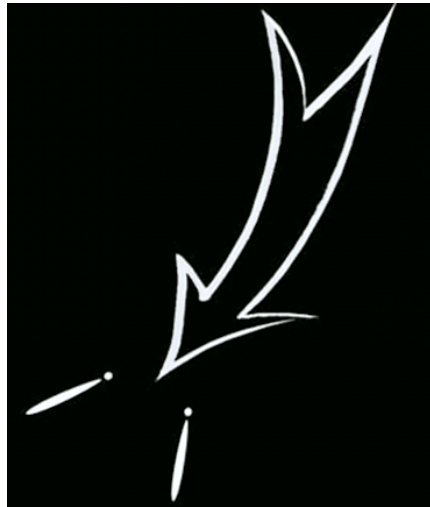
我们的目标是什么？

交替的打出A和B...

怎么可能打出B?//让B进程执行

接下来会发生什么?把自己变成计算机想一想...

中断...什么中断?



时钟中断

```
void sched_init(void)  //在sched.c中
{ set_intr_gate(0x20,&timer_interrupt);
```

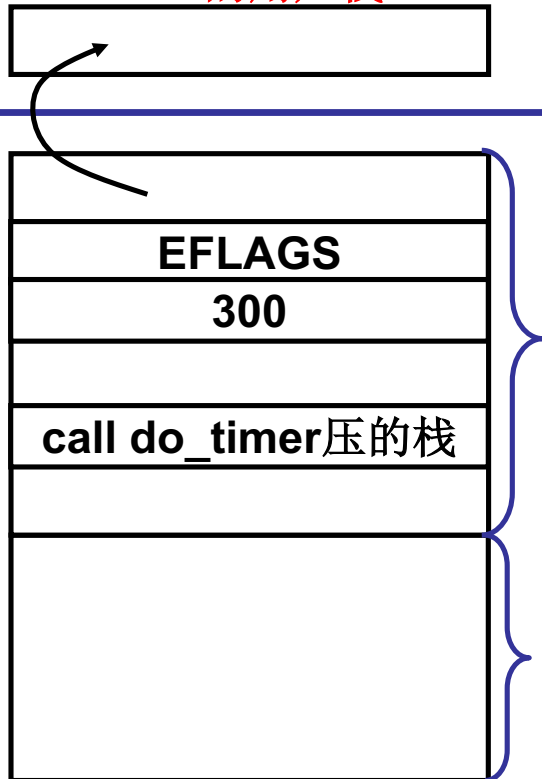
```
void _timer_interrupt:
    ...
    call do_timer
```

```
void do_timer(...)
{  if((--current->counter>0) return;
    current->counter=0;
    schedule(); }
```



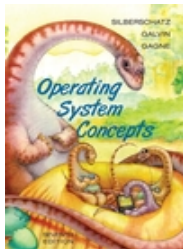
有那么一次时钟中断

B的用户栈

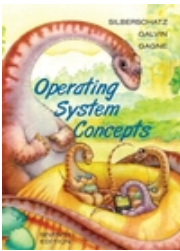
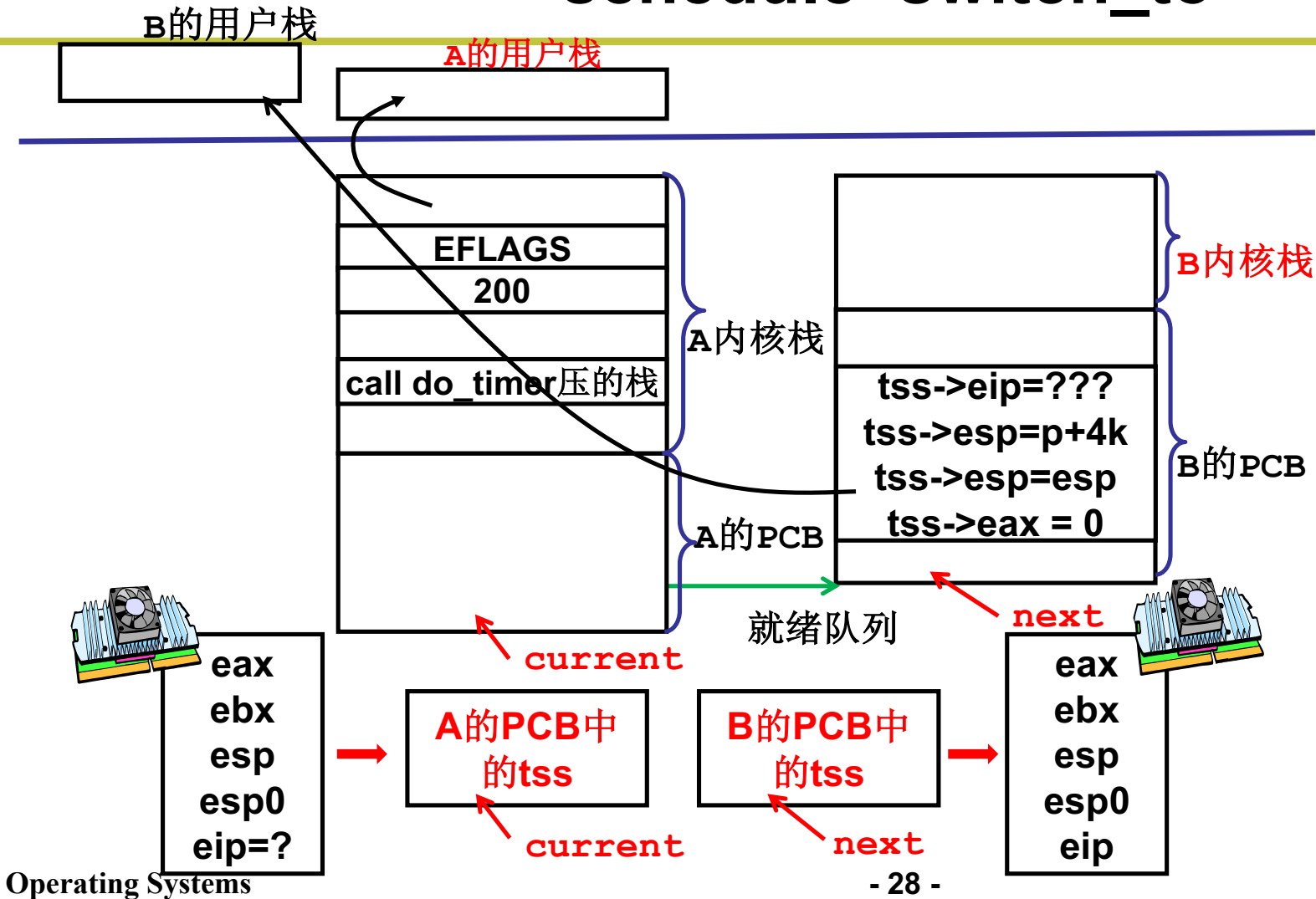


```
main() {  
    mov __NR_fork, %eax  
    int 0x80  
100: mov %eax, res  
    cml res, 0  
    jne 208  
200: printf("A")  
    jmp 200  
208: ... 300: printf("B")  
        jmp 300  
308: wait()  
}
```

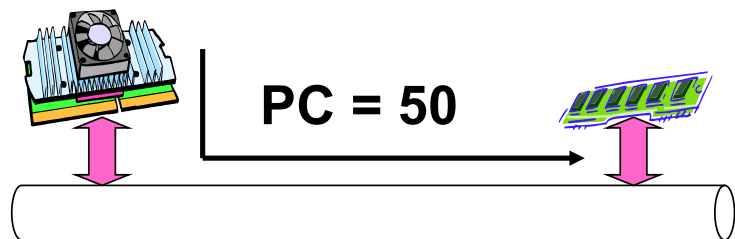
```
do_timer(...) {  
    current->counter=0; schedule(); }  
}
```



schedule+switch_to



接下来会怎么样？

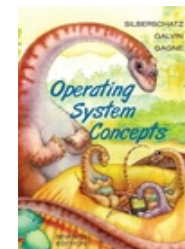


`eip=??? eax=0 esp=...`

代码始终不曾变过，
如同对操作系统的热爱...

进程B开始不断的打B...

```
main() {  
    mov __NR_fork, %eax  
    int 0x80  
100: mov %eax, res  
    cmpl res, 0  
    jne 304  
200: printf("A")  
    jmp 200  
208: ...300: printf("B")  
    jmp 300  
304: wait()  
}
```



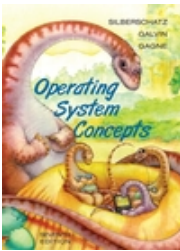
我们的目标达到了吗？

交替的打出**A**和**B**...

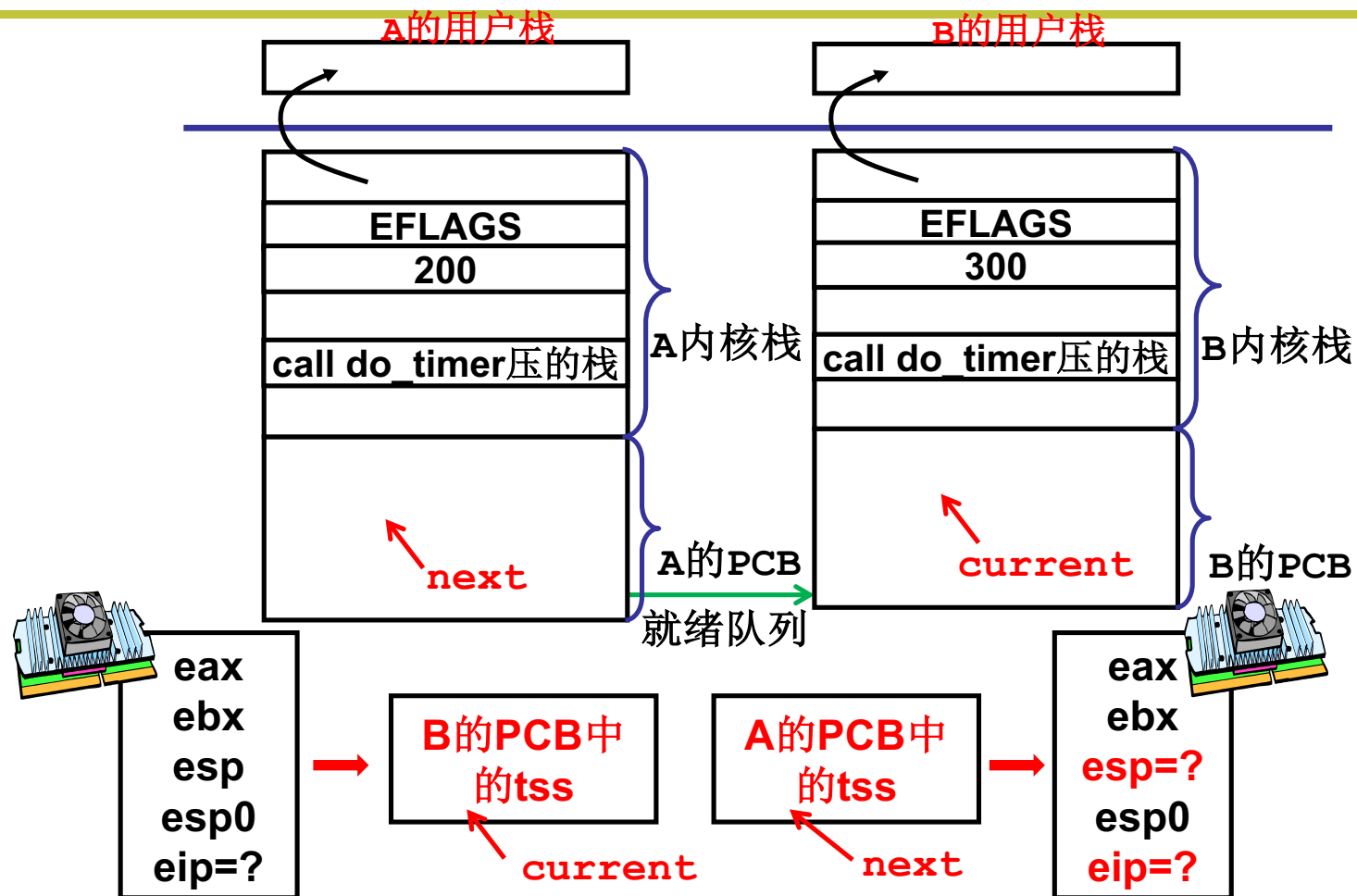
已经打出了**B**，完事了吗？ 何为交替？

接下来会发生什么？把自己变成计算机想一想...

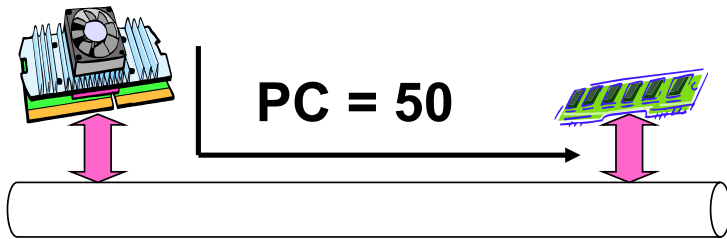
中断, 仍然是中断... 什么中断？



又有那么一次时钟中断, 再一次schedule+switch_to



接下来会怎么样？



```
void do_timer(...)  
{  
    ...  
    schedule();  
}
```

```
void _timer_interrupt:  
    ...  
    call do_timer  
    ...这后面是什么？
```

