本节内容

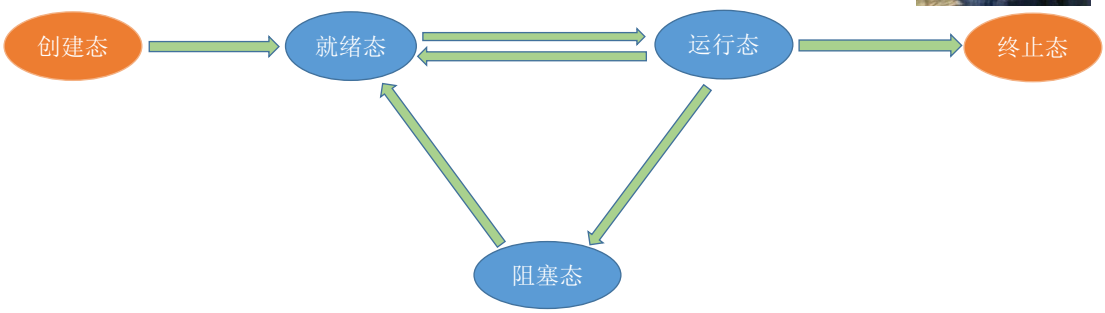
进程控制

王道考研/CSKAOYAN.COM

什么是进程控制？

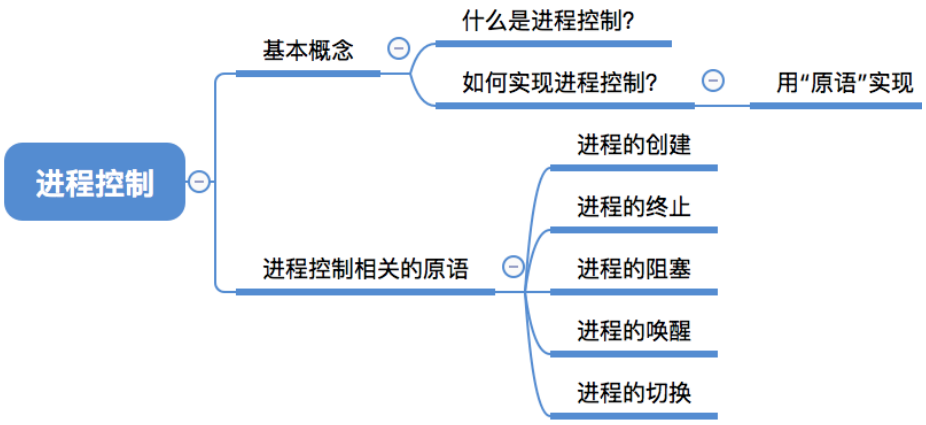
进程控制的主要功能是对系统中的所有进程实施有效的管理，它具有创建新进程、撤销已有进程、实现进程状态转换等功能。

简化理解：反正进程控制就是要实现进程状态转换



王道考研/CSKAOYAN.COM

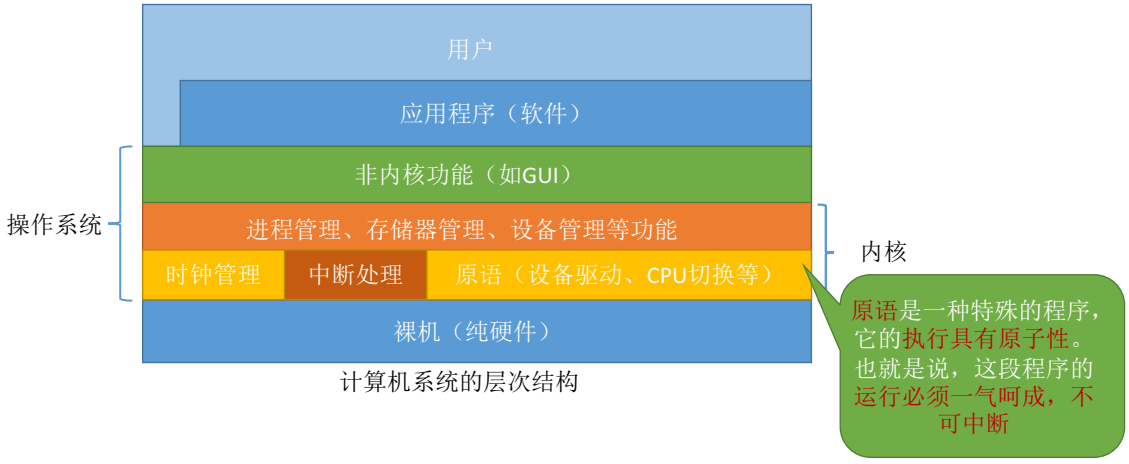
知识总览



王道考研/CSKAOYAN.COM

用“原语”实现

如何实现进程控制？



计算机系统的层次结构

王道考研/CSKAOYAN.COM

如何实现进程控制？

用“原语”实现

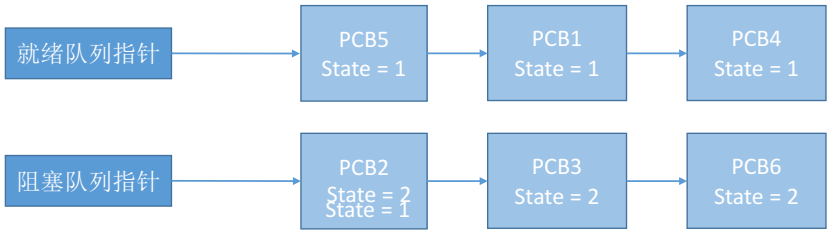
原语的执行具有“原子性”，一气呵成

思考：为何进程控制（状态转换）的过程要“一气呵成”？

如果不能“一气呵成”，就有可能导致操作系统中的某些关键数据结构信息不统一的情况，这会影响操作系统进行别的管理工作

可以用“原语”来实现“一气呵成”啊汪！

Eg：假设PCB中的变量 state 表示进程当前所处状态，1表示就绪态，2表示阻塞态...

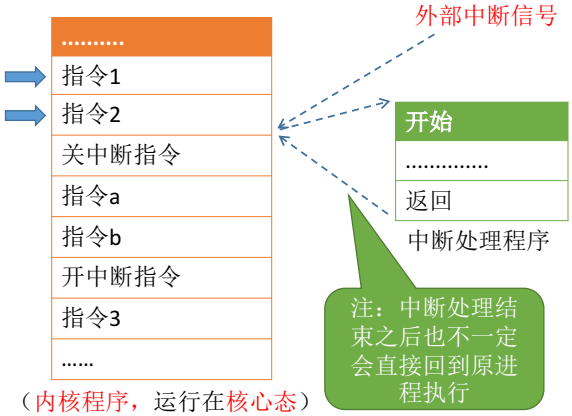


假设此时进程2等待的事件发生，则操作系统中，负责进程控制的内核程序至少需要做这样两件事：
①将PCB2的 state 设为 1
②将PCB2从阻塞队列放到就绪队列

完成了第一步后收到中断信号，那么PCB2的state=1，但是它却被放在阻塞队列里

如何实现原语的“原子性”？

原语的执行具有原子性，即执行过程只能一气呵成，期间不允许被中断。
可以用 “关中断指令”和“开中断指令”这两个特权指令实现原子性



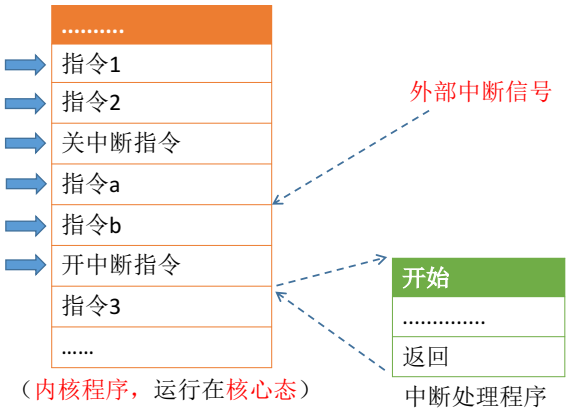
（内核程序，运行在核心态）

注：中断处理结束之后也不一定会直接回到原进程执行

正常情况：CPU每执行完一条指令都会例行检查是否有中断信号需要处理，如果有，则暂停运行当前这段程序，转而执行相应的中断处理程序。

如何实现原语的“原子性”？

原语的执行具有原子性，即执行过程只能一气呵成，期间不允许被中断。
可以用 “关中断指令”和“开中断指令”这两个特权指令实现原子性



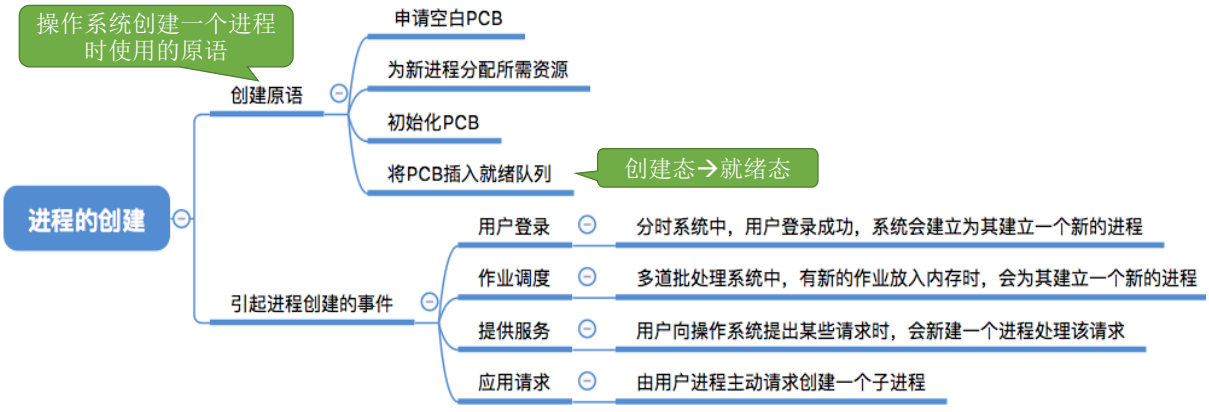
（内核程序，运行在核心态）

CPU执行了关中断指令之后，就不再例行检查中断信号，直到执行开中断指令之后才会恢复检查。

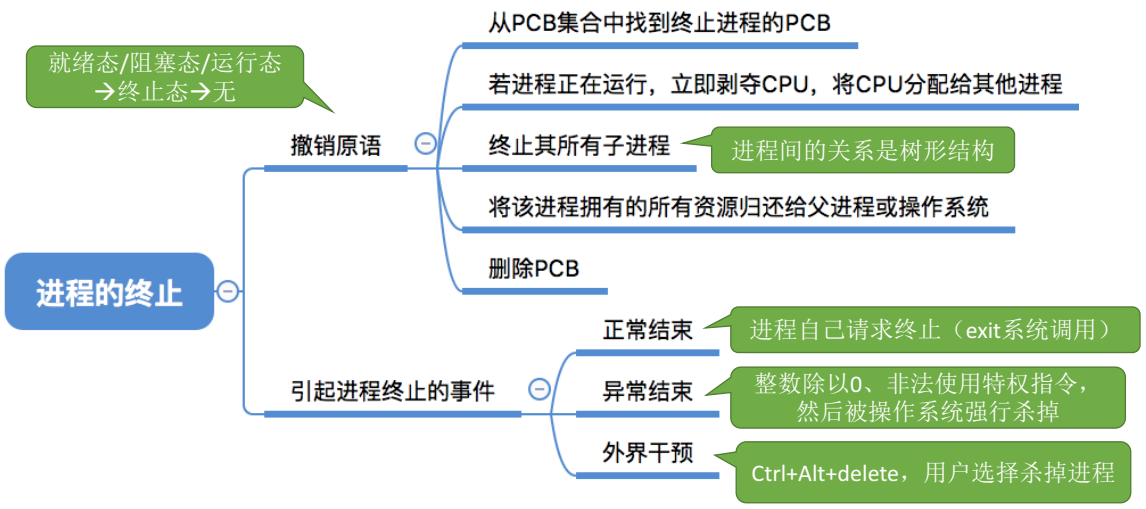
这样，关中断、开中断 之间的这些指令序列就是不可被中断的，这就实现了“原子性”

思考：如果这两个特权指令允许用户程序使用的话，会发生什么情况？

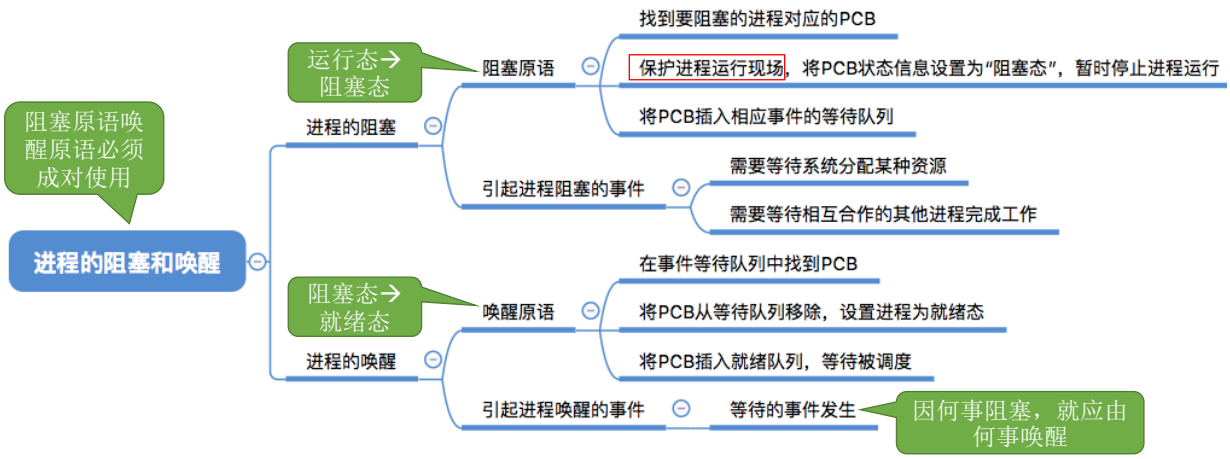
进程控制相关的原语



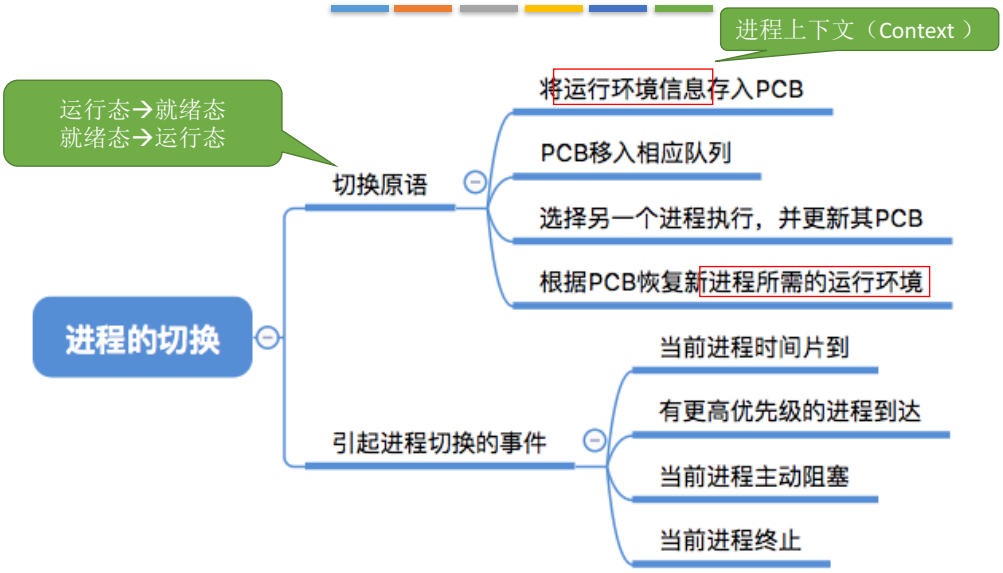
进程控制相关的原语



进程控制相关的原语

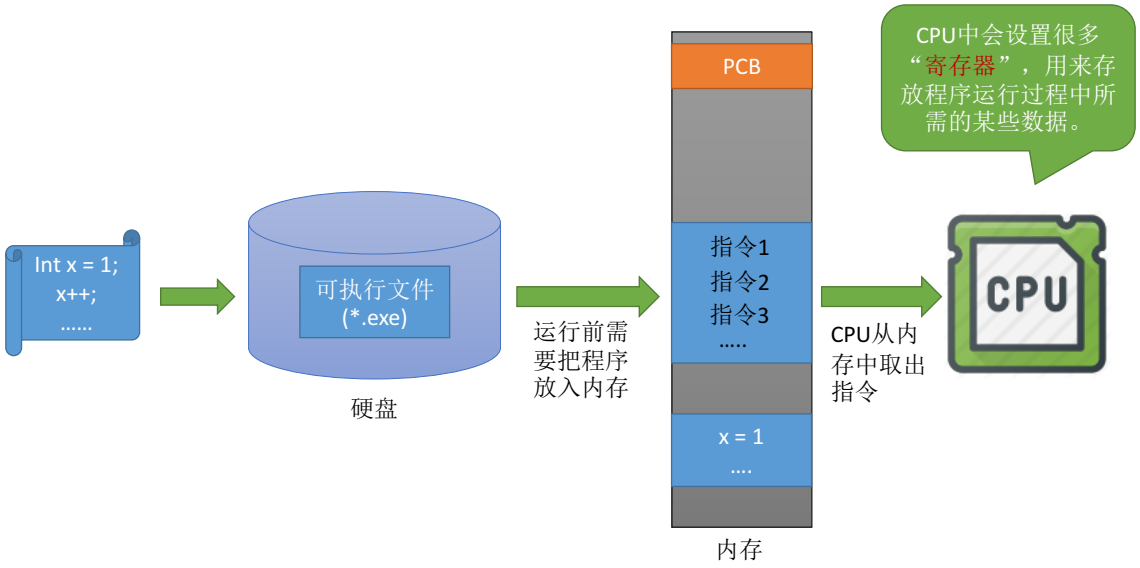


进程控制相关的原语



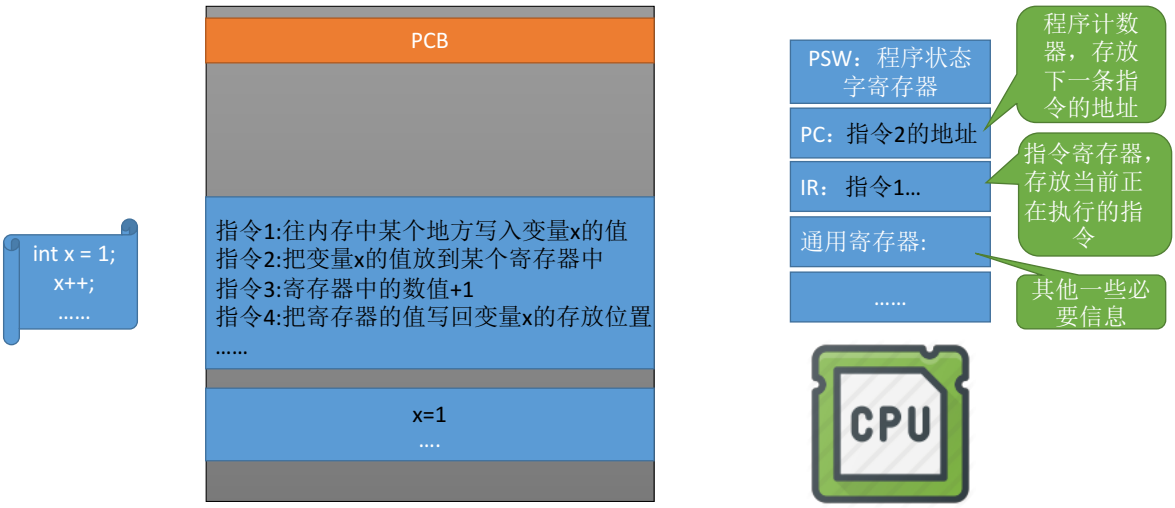
王道考研/CSKAOYAN.COM

知识滚雪球：程序是如何运行的？

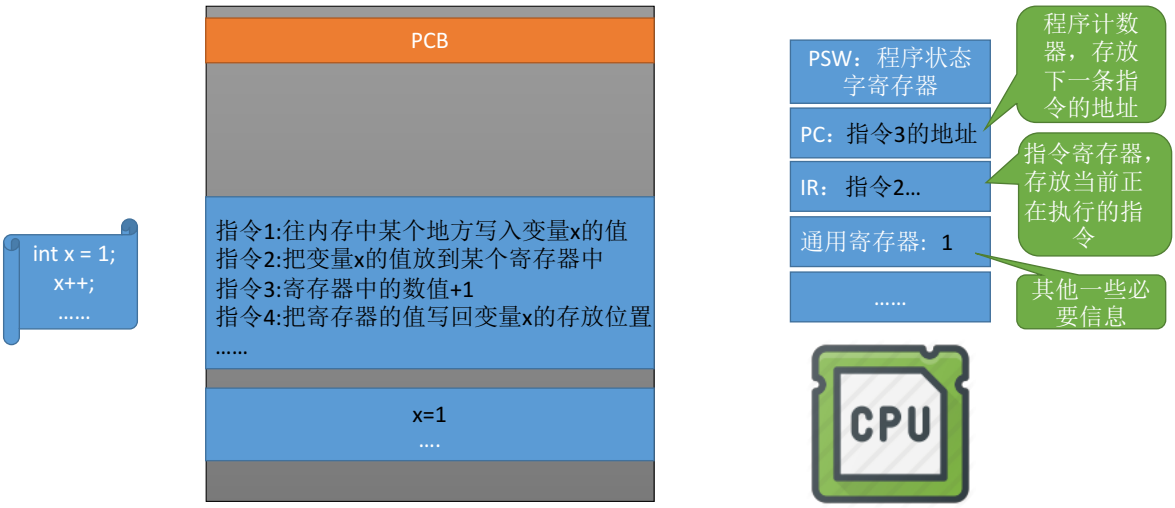


王道考研/CSKAOYAN.COM

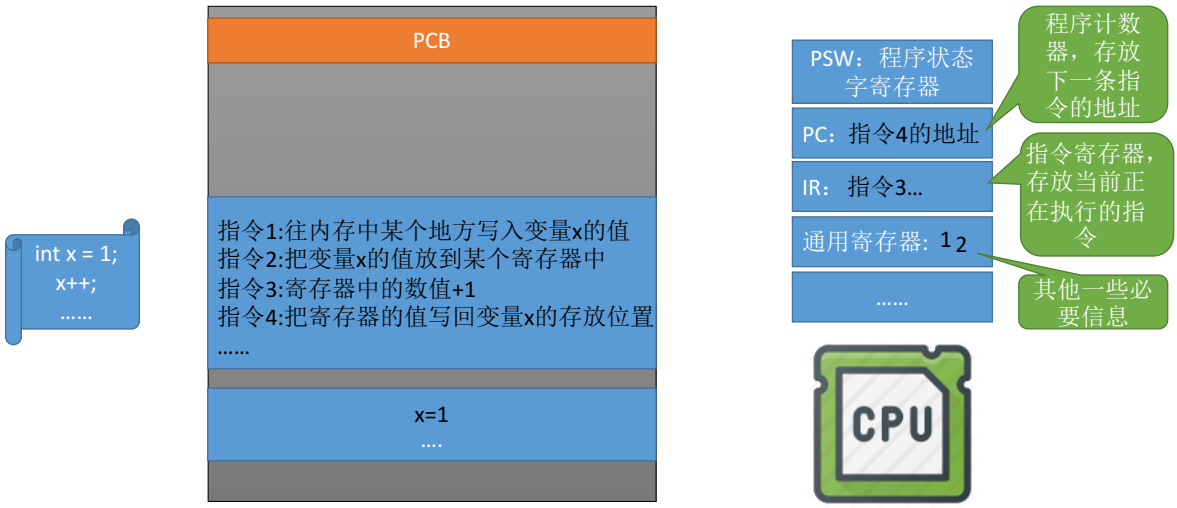
知识滚雪球：程序是如何运行的？



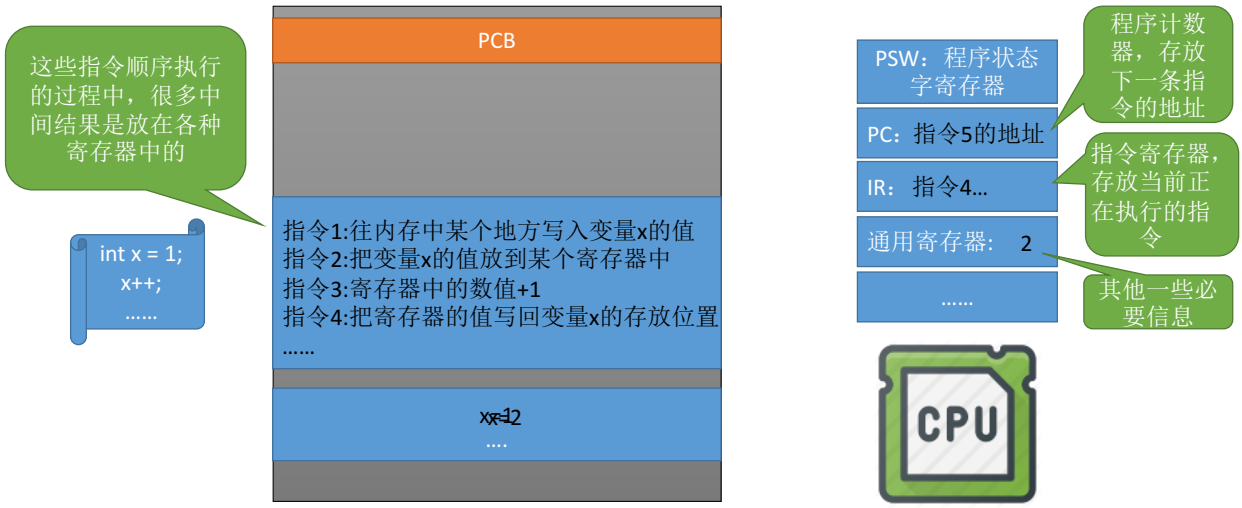
知识滚雪球：程序是如何运行的？



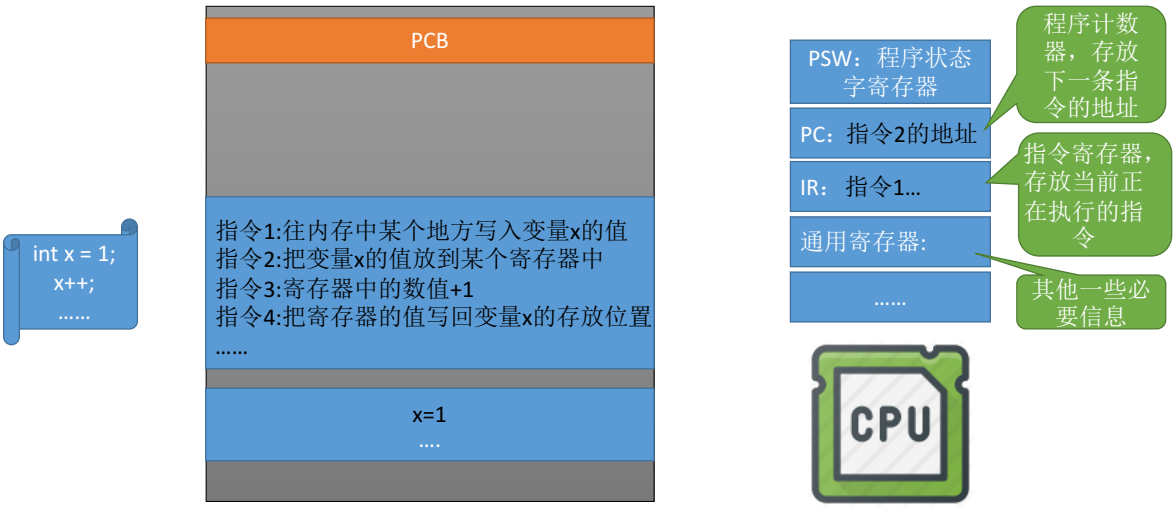
知识滚雪球：程序是如何运行的？



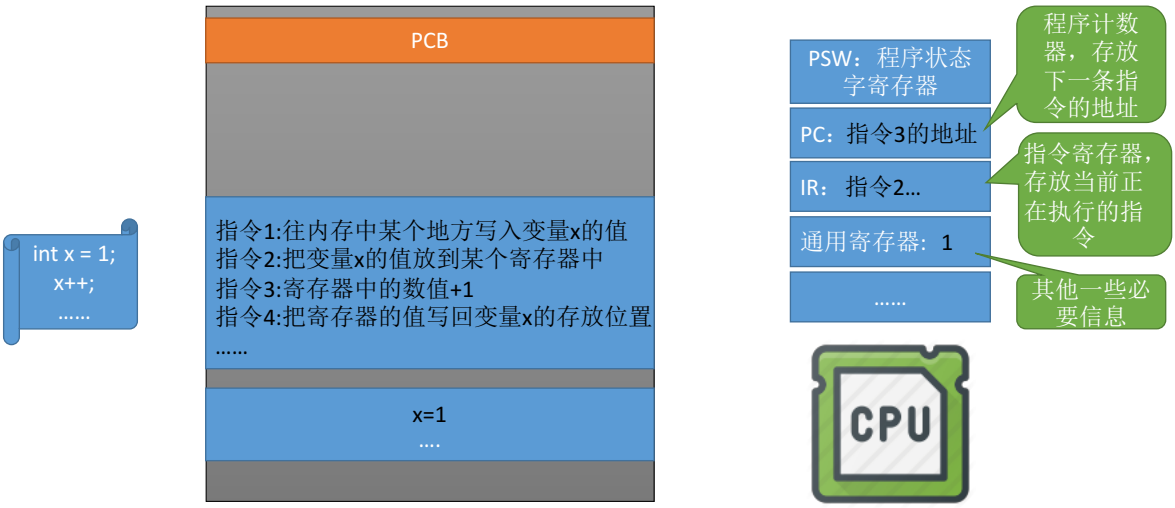
知识滚雪球：程序是如何运行的？



知识滚雪球：程序是如何运行的？



知识滚雪球：程序是如何运行的？

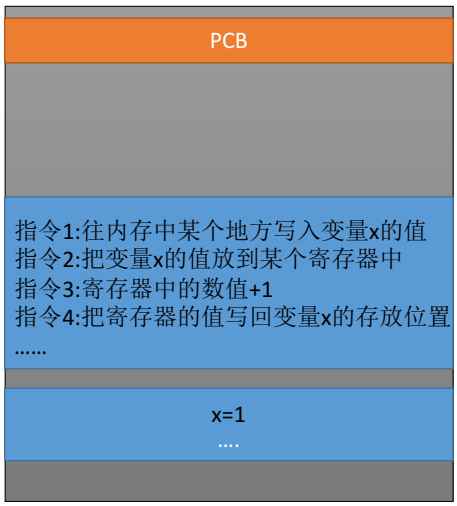


知识滚雪球：程序是如何运行的？

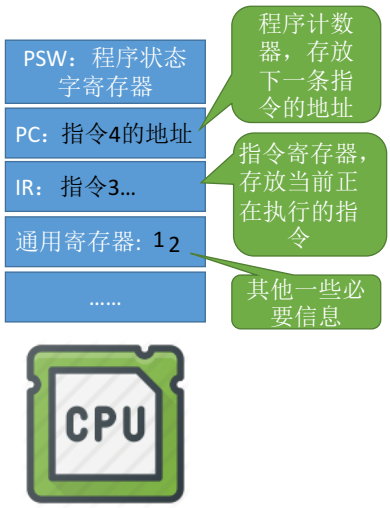
思考：执行完指令3后，另一个进程开始上CPU运行。

注意：另一个进程在运行过程中也会使用各个寄存器

int x = 1;
x++;
……



内存



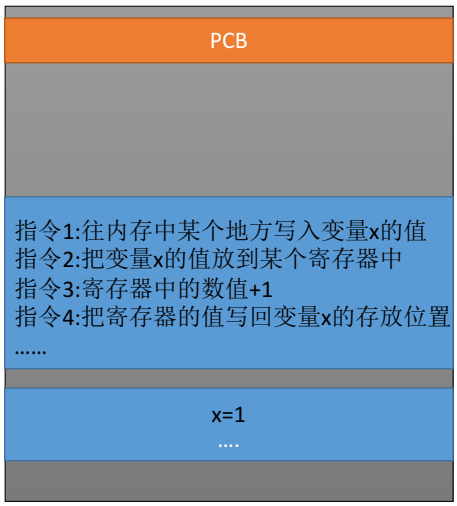
知识滚雪球：程序是如何运行的？

思考：执行完指令3后，另一个进程开始上CPU运行。

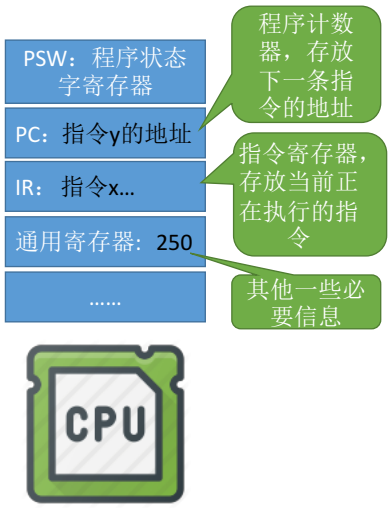
注意：另一个进程在运行过程中也会使用各个寄存器

int x = 1;
x++;
……

灵魂拷问：之后还怎么切换回之前的进程？？？？



内存



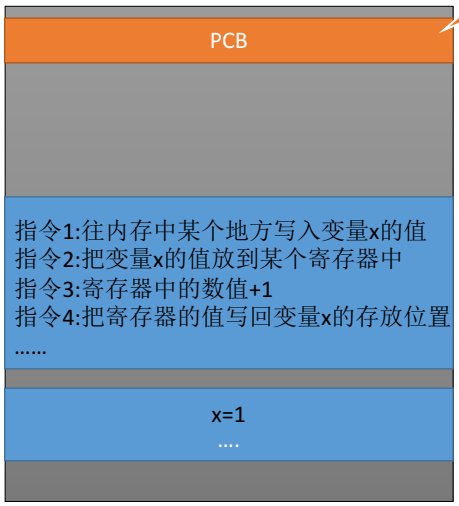
知识滚雪球：程序是如何运行的？

思考：执行完指令3后，另一个进程开始上CPU运行。

注意：另一个进程在运行过程中也会使用各个寄存器

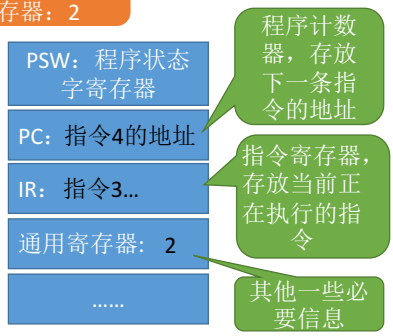
```
int x = 1;
x++;
.....
```

解决办法：在进程切换时先在PCB中保存这个进程的运行环境（保存一些必要的寄存器信息）



内存

PSW：xxxxx
PC：指令4的地址
通用寄存器：2



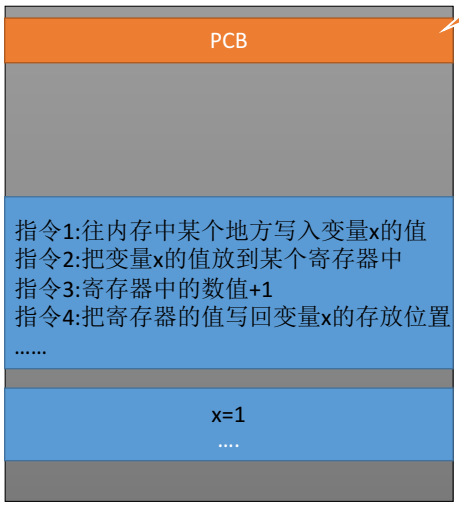
知识滚雪球：程序是如何运行的？

思考：执行完指令3后，另一个进程开始上CPU运行。

注意：另一个进程在运行过程中也会使用各个寄存器

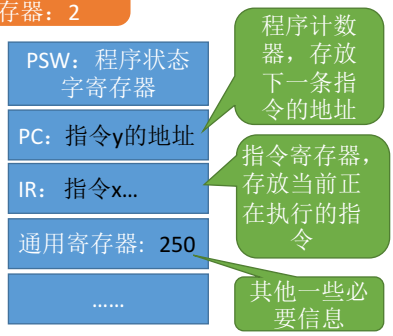
```
int x = 1;
x++;
.....
```

解决办法：在进程切换时先在PCB中保存这个进程的运行环境（保存一些必要的寄存器信息）



内存

PSW：xxxxx
PC：指令4的地址
通用寄存器：2



王道考研/CSKAOYAN.COM

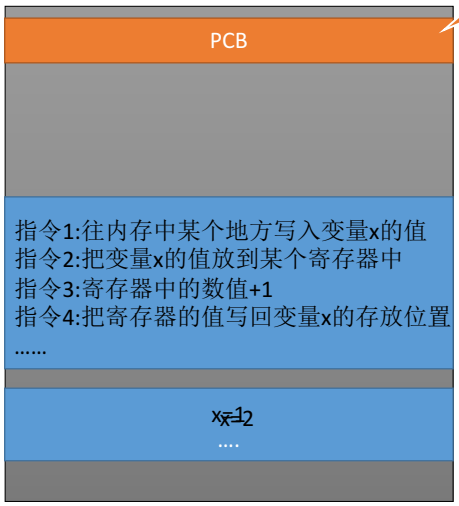
知识滚雪球：程序是如何运行的？

思考：执行完指令3后，另一个进程开始上CPU运行。

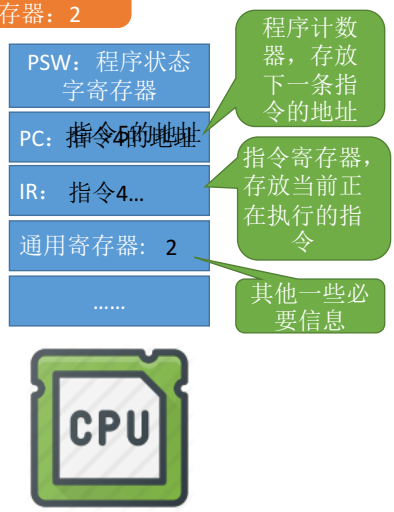
注意：另一个进程在运行过程中也会使用各个寄存器

int x = 1;
x++;
.....

解决办法：在进程切换时先在PCB中保存这个进程的运行环境（保存一些必要的寄存器信息）



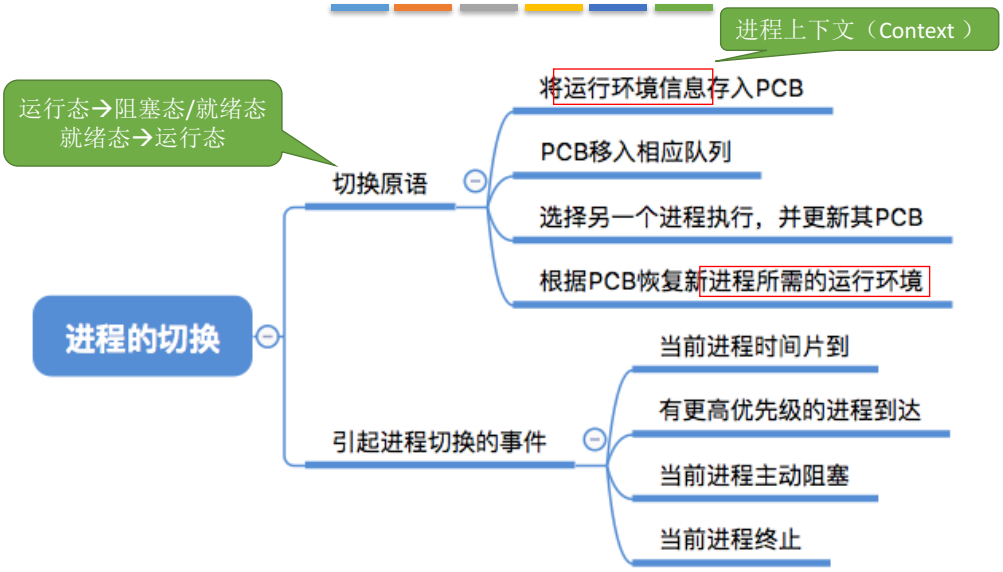
内存



当原来的进程再次投入运行时，可以通过PCB恢复它的运行环境

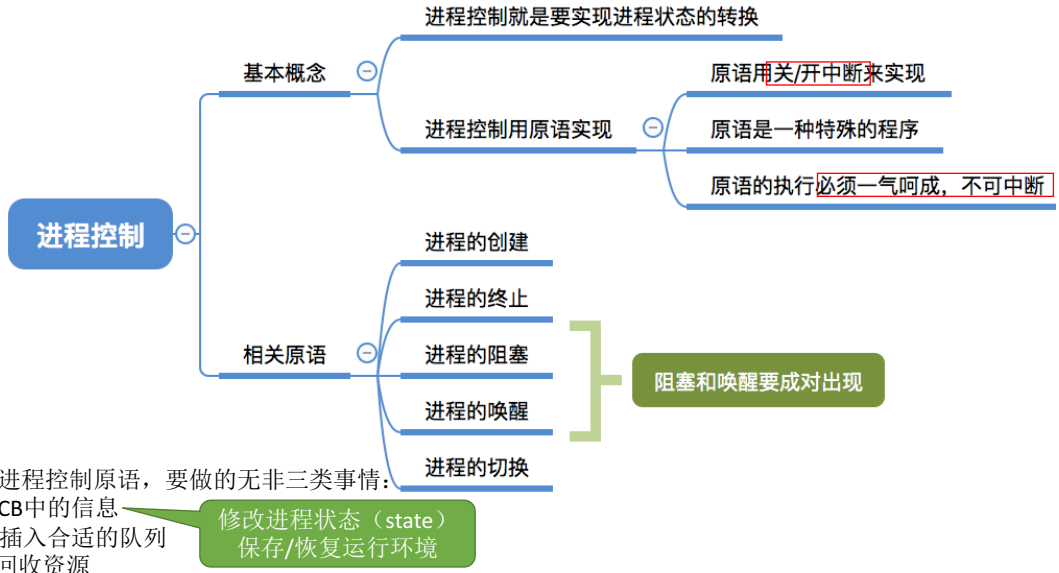
王道考研/CSKAOYAN.COM

进程控制相关的原语



王道考研/CSKAOYAN.COM

知识回顾与重要考点



无论哪个进程控制原语，要做的无非三类事情：

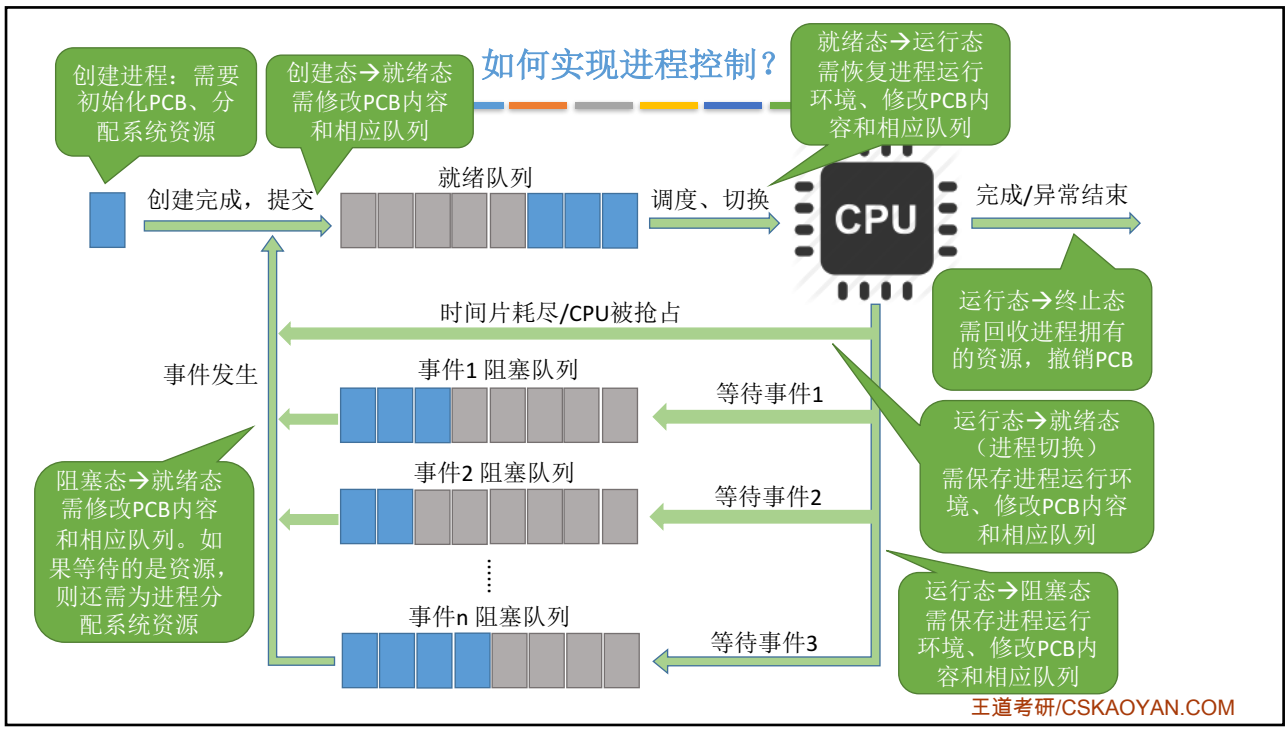
1. 更新PCB中的信息
2. 将PCB插入合适的队列
3. 分配/回收资源

修改进程状态（state）
保存/恢复运行环境

进程控制相关的原语

学习技巧：进程控制会导致进程状态的转换。无论哪个进程控制原语，要做的无非三类事情：

1. 更新PCB中的信息
   a. 所有的进程控制原语一定都会修改进程状态标志
   b. 剥夺当前运行进程的CPU使用权必然需要保存其运行环境
   c. 某进程开始运行前必然要恢复期运行环境
2. 将PCB插入合适的队列
3. 分配/回收资源


如何实现进程控制？
创建进程：需要初始化PCB、分配系统资源
创建态→就绪态需修改PCB内容和相应队列
就绪态→运行态需恢复进程运行环境、修改PCB内容和相应队列
创建完成，提交
就绪队列
调度、切换
CPU
完成/异常结束
运行态→终止态需回收进程拥有的资源，撤销PCB
时间片耗尽/CPU被抢占
事件发生
事件1 阻塞队列
等待事件1
运行态→就绪态（进程切换）需保存进程运行环境、修改PCB内容和相应队列
事件2 阻塞队列
等待事件2
阻塞态→就绪态需修改PCB内容和相应队列。如果等待的是资源，则还需为进程分配系统资源
事件n 阻塞队列
等待事件3
运行态→阻塞态需保存进程运行环境、修改PCB内容和相应队列
王道考研/CSKAOYAN.COM