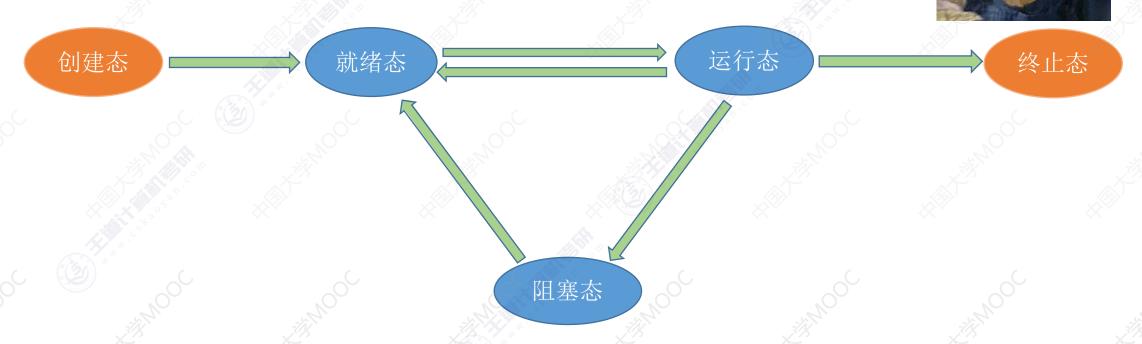


什么是进程控制?

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24考研交流群: 769832062

王道考研/CSKAOYAN.COM

知识总览



如何实现进程控制?

用户

应用程序(软件)

非内核功能(如GUI)

进程管理、存储器管理、设备管理等功能

时钟管理

中断处理

原语(设备驱动、CPU切换等)

裸机 (纯硬件)

计算机系统的层次结构

内核

原语是一种特殊的程序, 它的执行具有原子性。 也就是说,这段程序的 运行必须一气呵成,不 可中断

王道24考研交流群: 769832062

操作系统一

用"原语"实现

如何实现进程控制?

原语的执行具有"原子性",一气呵成

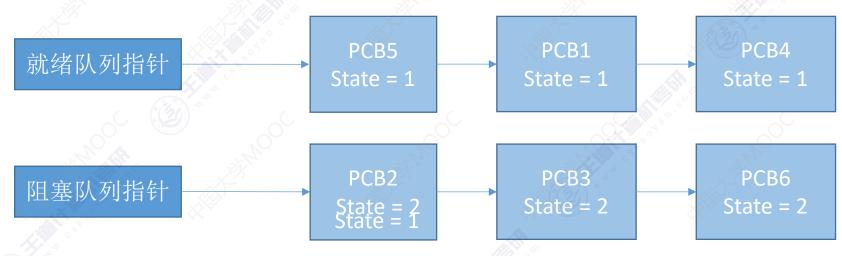
思考:为何进程控制(状态转换)的过程要"一气呵成"?

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



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...

可以用"原语" 来实现"一气 呵成"啊汪!



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

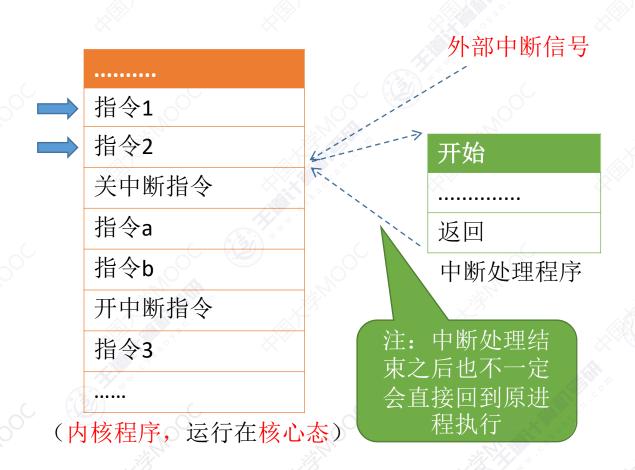
完成了第一步后收到中断信号,那么PCB2的state=1,但是它却被放在阻塞队列里

王道24考研交流群: 769832062

土迫考研/CSKAOYAN.COM

如何实现原语的"原子性"?

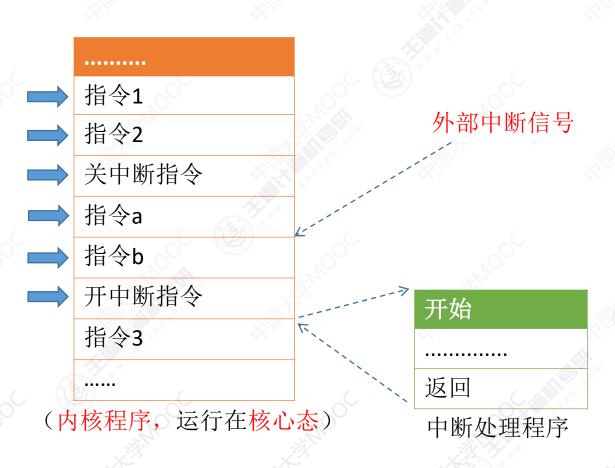
原语的执行具有原子性,即执行过程只能一气呵成,期间不允许被中断。 可以用"关中断指令"和"开中断指令"这两个特权指令实现原子性



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"?

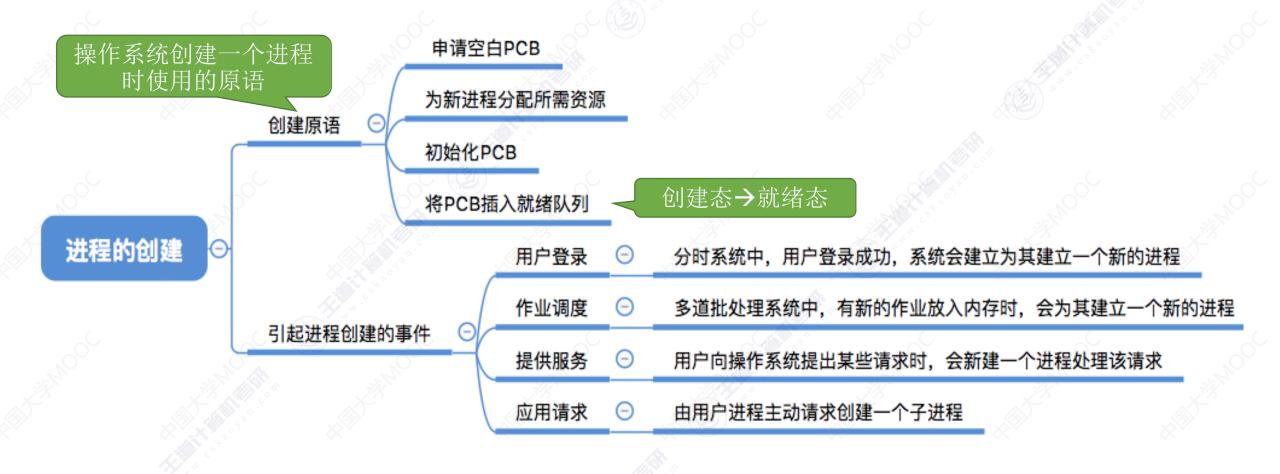
原语的执行具有原子性,即执行过程只能一气呵成,期间不允许被中断。 可以用"关中断指令"和"开中断指令"这两个特权指令实现原子性



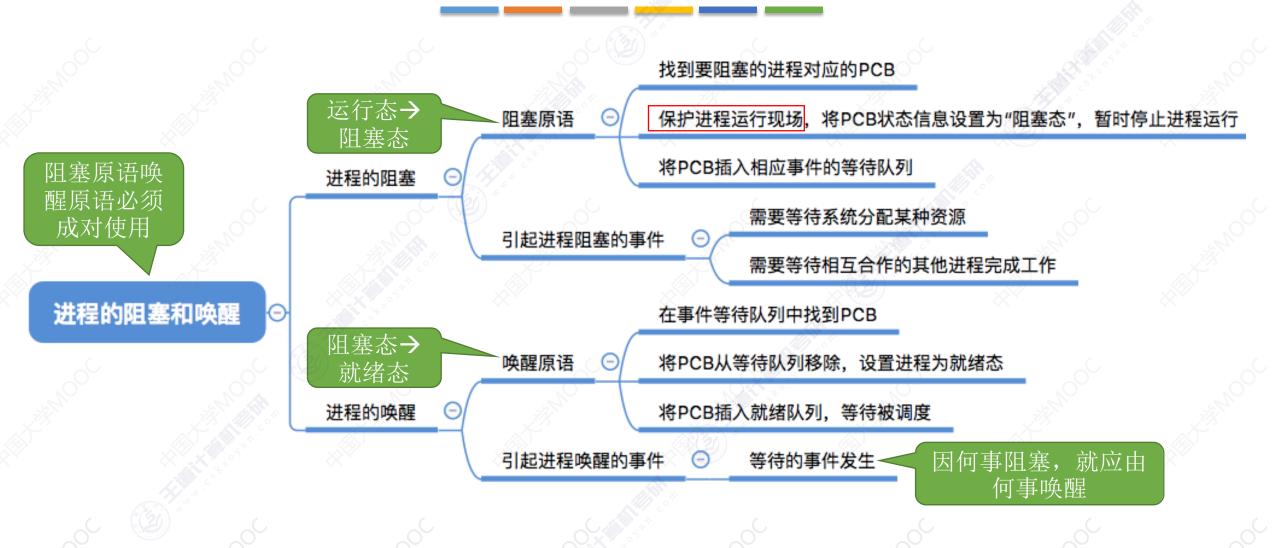
CPU执行了关中断指令之后,就不再例行 检查中断信号,直到执行开中断指令之后 才会恢复检查。

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"

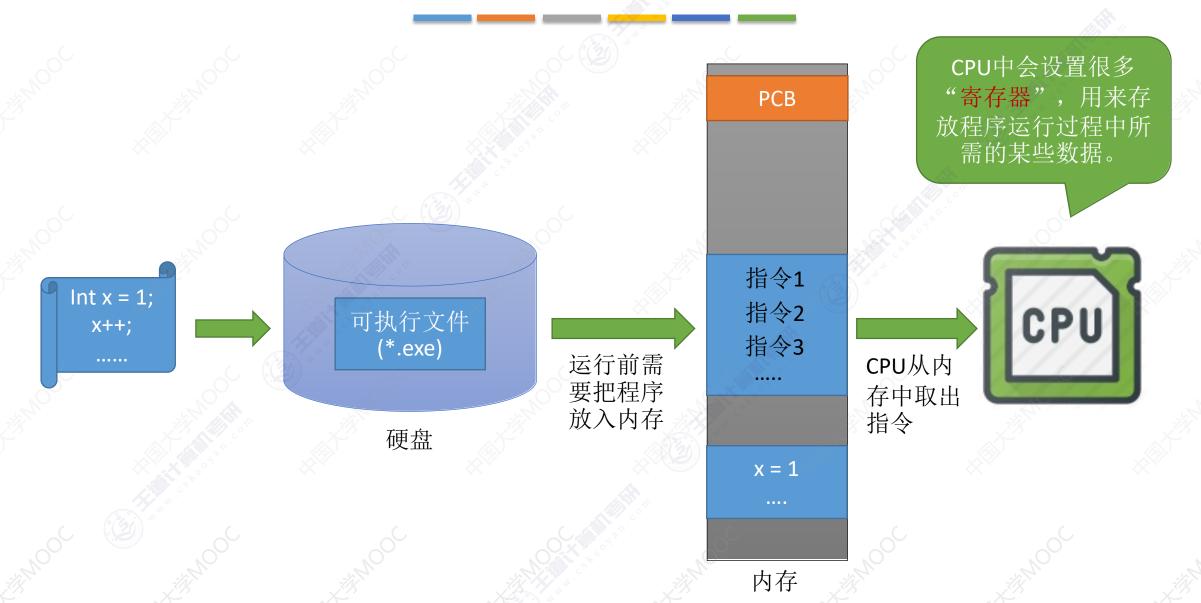
思考:如果这两个特权指令允许用户程序使用的话,会发生什么情况?







进程上下文(Context) 将运行环境信息存入PCB 运行态→就绪态 就绪态→运行态 PCB移入相应队列 切换原语 选择另一个进程执行,并更新其PCB 根据PCB恢复新进程所需的运行环境 进程的切换 当前进程时间片到 有更高优先级的进程到达 引起进程切换的事件 当前进程主动阻塞 当前进程终止



int x = 1; x++;

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 x=1

内存

PSW:程序状态字寄存器

PC: 指令2的地址

IR: 指令1...

通用寄存器:

•••••

器,存放下一条指令的地址

程序计数

指令寄存器, 存放当前正 在执行的指 令



int x = 1;

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 x=1

内存

PSW: 程序状态 字寄存器

PC: 指令3的地址

IR: 指令2...

通用寄存器: 1

程序计数 器,存放 下一条指 令的地址

指令寄存器, 存放当前正 在执行的指

其他一些必 要信息



X++;

指令1:往内存 指令2:把变量 指令3:寄存器 指令4:把寄存

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 x=1

内存

PSW:程序状态字寄存器

PC: 指令4的地址

IR: 指令3...

通用寄存器: 12

•••••

程序计数器,存放下一条指令的地址

指令寄存器, 存放当前正 在执行的指 令



这些指令顺序执行 的过程中,很多中 间结果是放在各种 寄存器中的

> int x = 1; x++;

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 xx=12

内存

PSW:程序状态 字寄存器

PC: 指令5的地址

IR: 指令4...

通用寄存器:

.....

程序计数器,存放下一条指令的地址

指令寄存器, 存放当前正 在执行的指 令

其他一些必 __要信息



int x = 1; x++;

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 x=1

内存

PSW:程序状态字寄存器

PC: 指令2的地址

IR: 指令1...

通用寄存器:

•••••

器,存放下一条指令的地址

程序计数

指令寄存器, 存放当前正 在执行的指 令



int x = 1;

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 x=1

内存

PSW: 程序状态 字寄存器

PC: 指令3的地址

IR: 指令2...

通用寄存器: 1

程序计数 器,存放 下一条指 令的地址

指令寄存器, 存放当前正 在执行的指

其他一些必 要信息

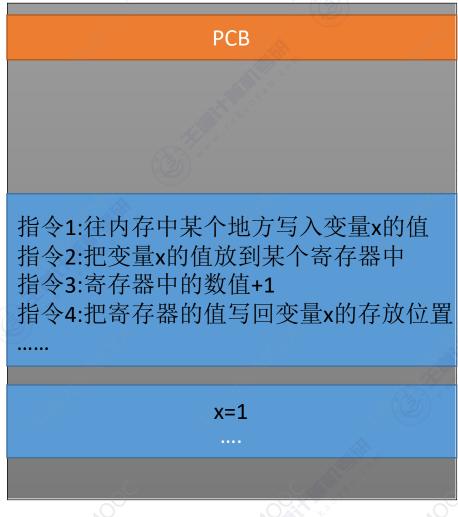


X++;

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

PSW:程序状态 字寄存器

PC: 指令4的地址

IR: 指令3...

通用寄存器: 12

.....

程序计数器,存放下一条指令的地址

指令寄存器, 存放当前正 在执行的指 今

其他一些必 __要信息

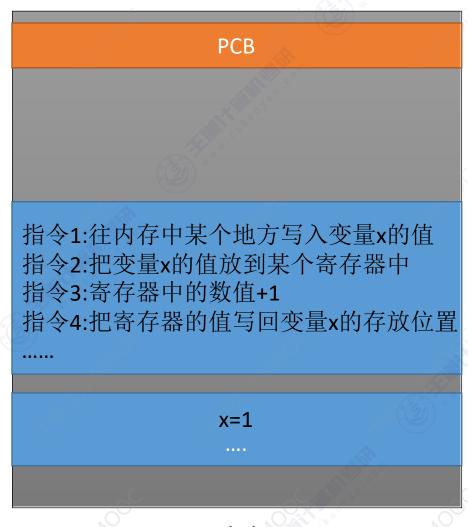


思考: 执行完指令3后, 另一个进程开始上CPU 运行。

注意:另一个进程在运行过程中也会使用各个寄存器



灵魂拷问:之后还怎么 切换回之前的进 程????



内存

PSW:程序状态 字寄存器

PC: 指令y的地址

IR: 指令x...

通用寄存器: 250

.

程序计数器,存放下一条指令的地址

指令寄存器, 存放当前正 在执行的指 令



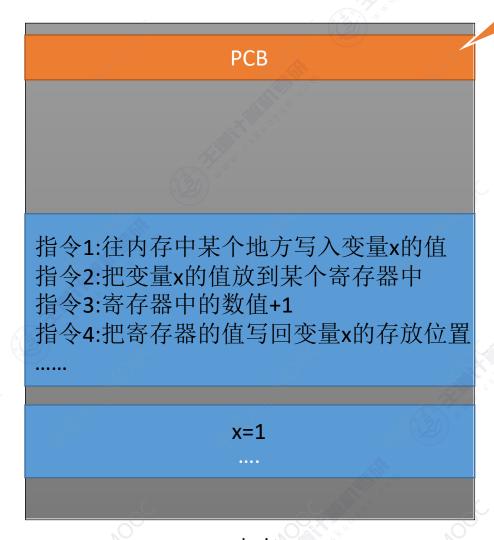
知识滚雪球:程序是如何运产的 PSW: XXXXX

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:程序状态 字寄存器

PC: 指令4的地址

IR: 指令3...

PC: 指令4的地址

通用寄存器: 2

通用寄存器: 2

.....

程序计数器,存放下一条指令的地址

指令寄存器, 存放当前正 在执行的指 今



知识滚雪球:程序是如何运产的 PSW: XXXXX

PC: 指令4的地址

通用寄存器: 2

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)

PCB 指令1:往内存中某个地方写入变量x的值 指令2:把变量x的值放到某个寄存器中 指令3:寄存器中的数值+1 指令4:把寄存器的值写回变量x的存放位置 x=1

内存

PSW:程序状态 字寄存器

PC: 指令y的地址

IR: 指令x...

通用寄存器: 250

程序计数 器,存放 下一条指 令的地址

指令寄存器, 存放当前正 在执行的指



知识滚雪球:程序是如何运产数 PSW: XXXXX

PC: 指令4的地址

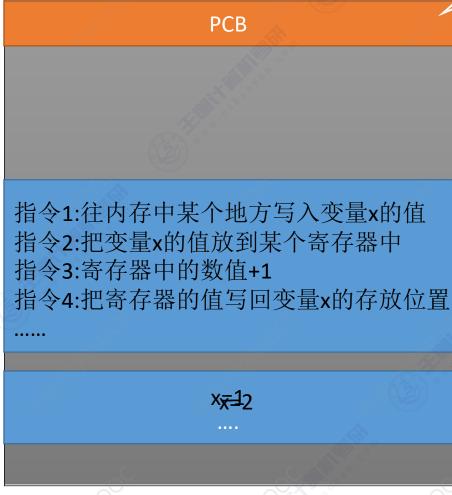
通用寄存器: 2

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



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:程序状态 字寄存器

PC: 指令4的地址

IR: 指令4...

通用寄存器: 2

程序计数 器,存放 下一条指 令的地址

指令寄存器, 存放当前正 在执行的指



进程上下文(Context)

运行态**→**阻塞态/就绪态 就绪态**→**运行态

切换原语

引起进程切换的事件

将运行环境信息存入PCB

PCB移入相应队列

选择另一个进程执行,并更新其PCB

根据PCB恢复新进程所需的运行环境

进程的切换

当前进程时间片到

有更高优先级的进程到达

当前进程主动阻塞

当前进程终止

知识回顾与重要考点



分配/回收资源 主道24考研交流群:

769832062

王道考研/CSKAOYAN.COM

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? 创建态→就绪态 创建进程:需要 需修改PCB内容 初始化PCB、分 配系统资源 和相应队列 就绪队列 创建完成,提交 调度、切换 **CPU** 时间片耗尽/CPU被抢占 事件1阻塞队列 事件发生 等待事件1 事件2阻塞队列 阻塞态→就绪态 等待事件2 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 事件n 阻塞队列 配系统资源 等待事件3

769832062

王道24考研交流群:

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

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

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

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



△ 公众号: 王道在线



b站: 王道计算机教育



计 抖音: 王道计算机考研