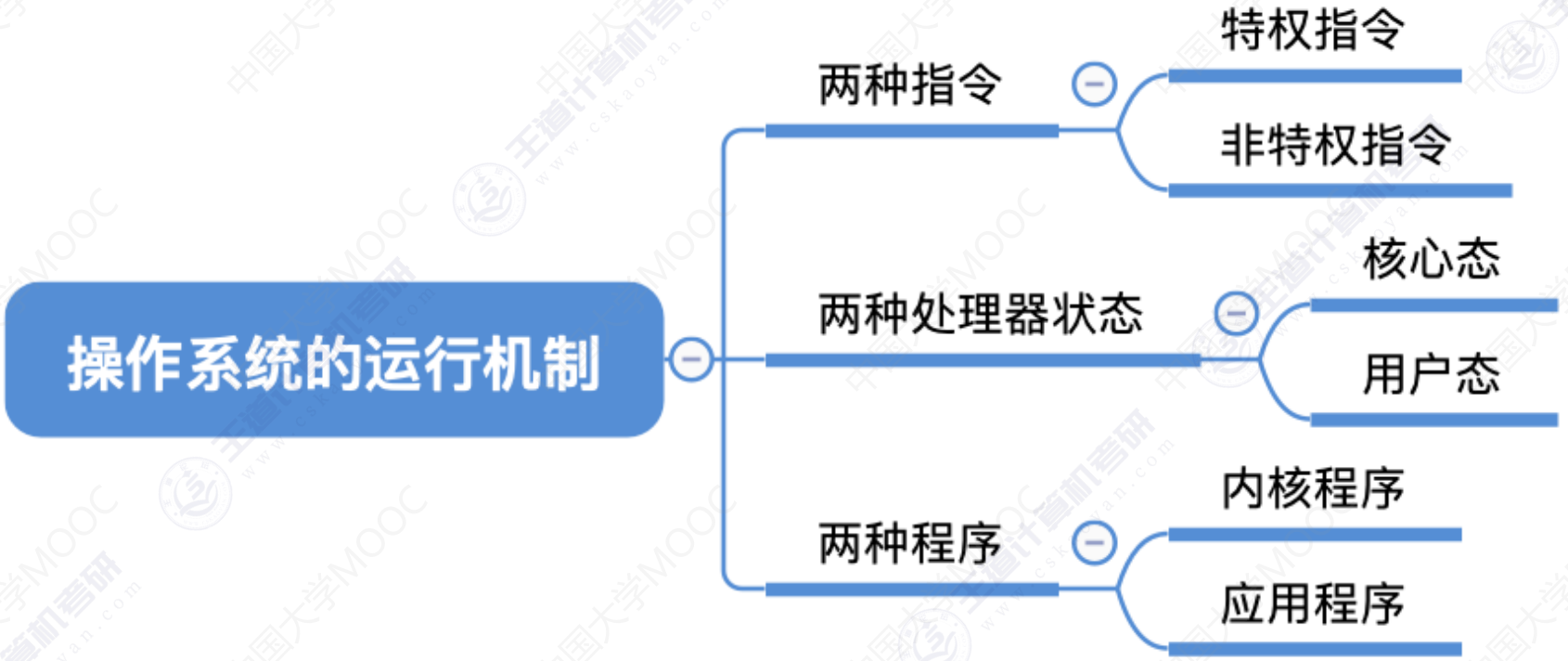
本节内容

操作系统的运行机制

知识总览



预备知识：程序是如何运行的？

C语言代码 → 编译器（“翻译”） → 机器指令（二进制）

一条高级语言的代码翻译过来可能会对应多条机器指令

Int x = 1;
x++;

```
10001010110000110001110000101100000000011010011110010010000000011000000001001011000100111100000011
```

程序运行的过程其实就是CPU执行一条一条的机器指令的过程

```
homlee — -bash — 59×12
[zhanghonglindeMacBook-Air:Documents homlee$ cd 王道OS/
 zhanghonglindeMacBook-Air:王道OS homlee$ ls
 ~$19年操作系统原题题库.docx          参考资料
 ~$正文.doc                          知识点讲解
 习题讲解                            教材修改建议
[冲刺串讲
 zhanghonglindeMacBook-Air:王道OS homlee$ cd~
[-bash: cd~: command not found
 zhanghonglindeMacBook-Air:王道OS homlee$ cd ~
[zhanghonglindeMacBook-Air:~ homlee$ mkdir
 usage: mkdir [-pv] [-m mode] directory ...
 zhanghonglindeMacBook-Air:~ homlee$
```

最基本命令

“小黑框”中使用的命令也

注意与本节的“指令”区别

预备知识：程序是如何运行的？

C语言代码 → 编译器（“翻译”） → 机器指令（二进制）

一条高级语言的代码翻译过来可能会对应多条机器指令

Int x = 1;
x++;

→

```
10001010110000110001110 0
00101100000000110100111 1
10010010000000110000000 1
00101100010011110000001 1
```

程序运行的过程其实就是CPU执行一条一条的机器指令的过程

“指令”就是处理器（CPU）能识别、执行的最基本命令

注：很多人习惯把 Linux、Windows、MacOS 的“小黑框”中使用的命令也称为“指令”，其实这是“交互式命令接口”，注意与本节的“指令”区别开。本节中的“指令”指二进制机器指令

内核程序 v.s. 应用程序

C语言代码 → 编译器（“翻译”） → 机器指令（二进制）

一条高级语言的代码翻译过来可能会对应多条机器指令

Int x = 1;
x++;

```
10001010110000110001110
00101100000000110100111
10010010000000110000001
00101100010011110000011
```

程序运行的过程其实就是CPU执行一条一条的机器指令的过程

我们普通程序员写的程序就是“应用程序”

微软、苹果有一帮人负责实现操作系统，他们写的是“内核程序”
由很多内核程序组成了“操作系统内核”，或简称“内核（Kernel）”
内核是操作系统最重要最核心的部分，也是最接近硬件的部分
甚至可以说，一个操作系统只要有内核就够了（eg：Docker—>仅需Linux内核）
操作系统的功能未必都在内核中，如图形化用户界面 GUI

特权指令 v.s. 非特权指令

Int x = 1;
x++;

```
10001010110000110001110
00101100000000110100111
10010010000000110000001
00101100010011110000011
```

应用程序只能使用“非特权指令”，如：加法指令、减法指令等

程序运行的过程其实就是CPU执行一条一条的机器指令的过程

我们普通程序员写的程序就是“应用程序”

微软、苹果有一帮人负责实现操作系统，他们写的就是“内核程序”

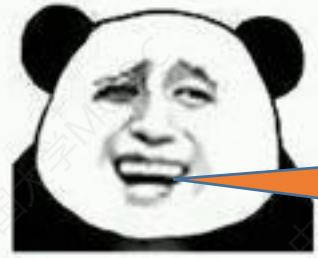
操作系统内核作为“管理者”，有时会让CPU执行一些“特权指令”，如：内存清零指令。这些指令影响重大，只允许“管理者”——即操作系统内核来使用

在CPU设计和生产的时候就划分了特权指令和非特权指令，因此CPU执行一条指令前就能判断出其类型

内核态 v.s. 用户态

Int x = 1;
x++;

10001010110000110001110
00101100000000110100111
10010010000000110000001
00101100010011110000011



那么问题来了...

CPU 能判断出指令类型，但是它怎么区分此时正在运行的是内核程序 or 应用程序？

程序运行的过程其实就是CPU执行一条一条的机器指令的过程

问题：如何实现CPU状态的切换？

CPU 有两种状态，“内核态”和“用户态”
处于内核态时，说明此时正在运行的是内核程序，此时可以执行特权指令
处于用户态时，说明此时正在运行的是应用程序，此时只能执行非特权指令

拓展：CPU 中有一个寄存器叫 程序状态字寄存器（PSW），其中有个二进制位，1表示“内核态”，0表示“用户态”
别名：内核态=核心态=管态；用户态=目态

内核程序

应用程序

处理中断信号的内核程序

00101100000000110100111110010010000000110000000110001011000010001111000000011

内核态

用户态

中断信号

操作系统内核在让出CPU之前，会用一条特权指令把 PSW 的标志位设置为“用户态”

一个

① 为“内核态”，

② 户可以启动某个

③ 操作系统内核程序在合适的时候主动让出CPU，让该应用程序上CPU运行

④ 应用程序运行在“用户态”

⑤ 此时，一位猥琐黑客在应用程序中植入了一条特权指令，企图破坏系统…

⑥ CPU发现接下来要执行的这条指令是特权指令，但是自己又处于“用户态”

⑦ 这个非法事件会引发一个中断信号

CPU检测到中断信号后，会立即变为“核心态”，并停止运行当前的应用程序，转而运行处理中断信号的内核程序

⑧ “中断”使操作系统再次夺回CPU的控制权

⑨ 操作系统会对引发中断的事件进行处理，处理完了再把CPU使用权交给别的应用程序

内核态、用户态 的切换

内核态→用户态：执行一条特权指令——修改PSW的标志位为“用户态”，这个动作意味着操作系统将主动让出CPU使用权

用户态→内核态：由“中断”引发，硬件自动完成变态过程，触发中断信号意味着操作系统将强行夺回CPU的使用权

除了非法使用特权指令之外，还有很多事件会触发中断信号。一个共性是，但凡需要操作系统介入的地方，都会触发中断信号

一个故事：

① 刚开机时，CPU 为“内核态”，操作系统内核程序先上CPU运行

② 开机完成后，用户可以启动某个应用程序

③ 操作系统内核程序在合适的时候主动让出 CPU，让该应用程序上CPU运行

操作系统内核在让出CPU之前，会用一条特权指令把 PSW 的标志位设置为“用户态”

④ 应用程序运行在“用户态”

⑤ 此时，一位猥琐黑客在应用程序中植入了一条特权指令，企图破坏系统…

⑥ CPU发现接下来要执行的这条指令是特权指令，但是自己又处于“用户态”

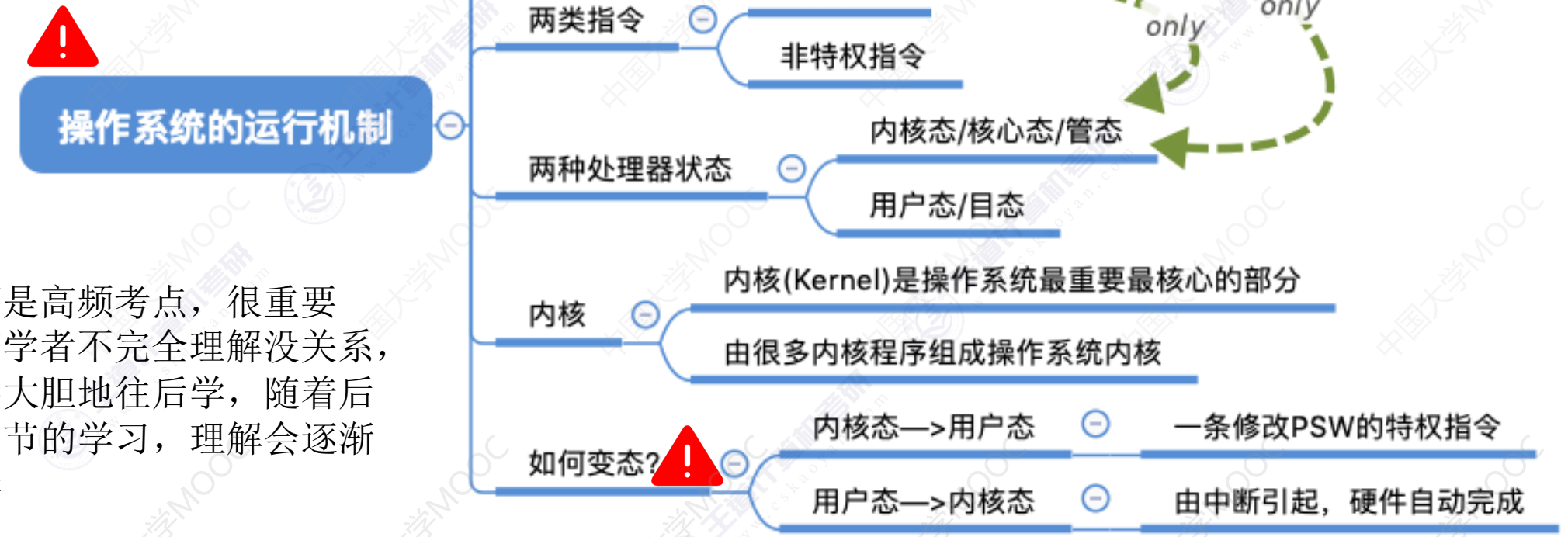
⑦ 这个非法事件会引发一个中断信号

CPU检测到中断信号后，会立即变为“核心态”，并停止运行当前的应用程序，转而运行处理中断信号的内核程序

⑧ “中断”使操作系统再次夺回CPU的控制权

⑨ 操作系统会对引发中断的事件进行处理，处理完了再把CPU使用权交给别的应用程序

知识回顾与重要考点



Tips：
1. 都是高频考点，很重要
2. 初学者不完全理解没关系，放心大胆地往后学，随着后面章节的学习，理解会逐渐加深

公众号：王道在线

b站：王道计算机教育

抖音：王道计算机考研