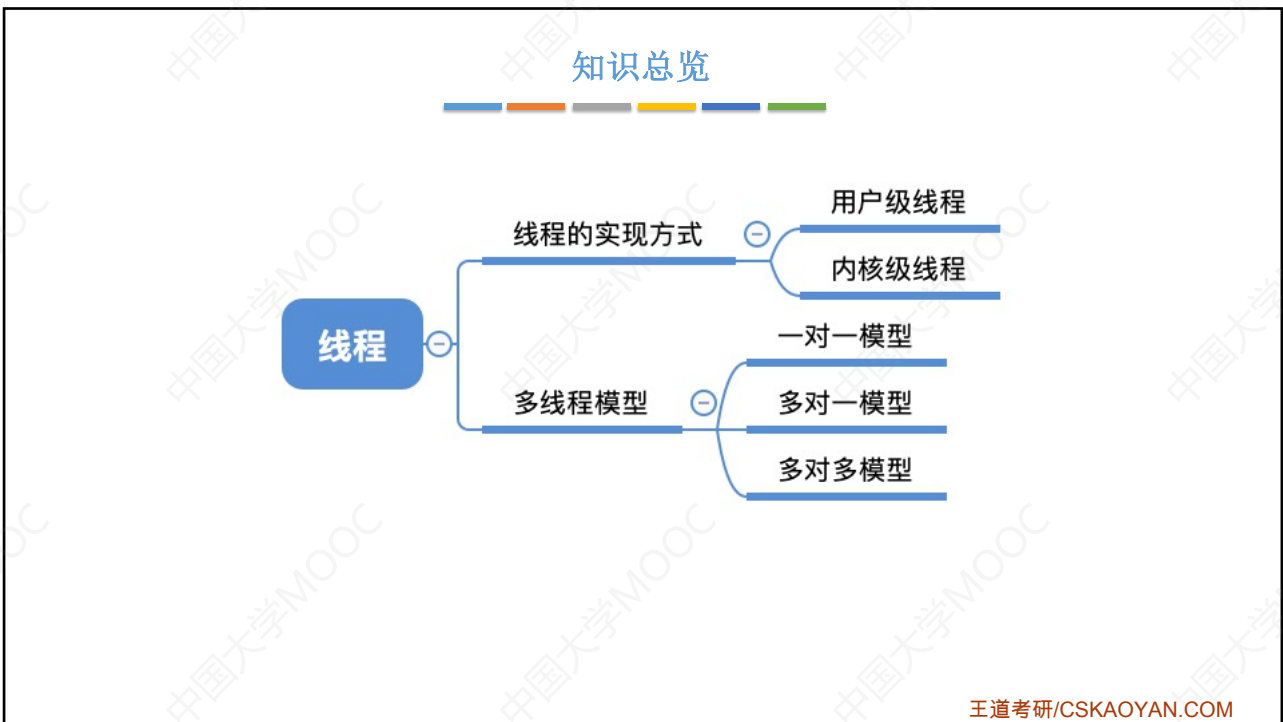


本节内容

线程的实现方式  
多线程模型

王道考研/CSKAOYAN.COM

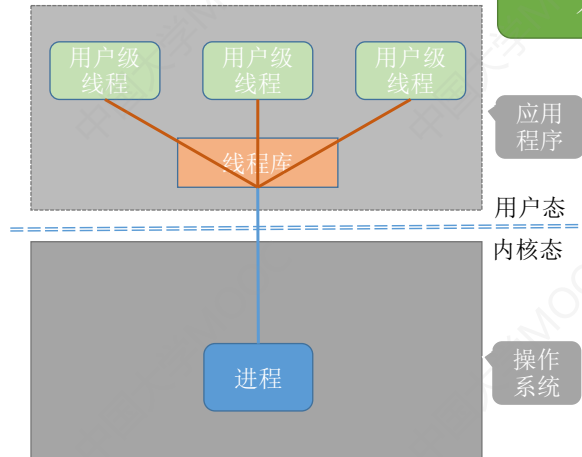
1



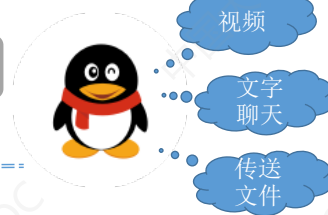
2

## 线程的实现方式

用户级线程 (User-Level Thread, ULT)



历史背景：早期的操作系统（如：早期Unix）只支持进程，不支持线程。当时的“线程”是由线程库实现的



```
int main() {
    while (true) {
        处理视频聊天的代码;
    }
}
```

进程1

```
int main() {
    while (true) {
        处理文字聊天的代码;
    }
}
```

进程2

```
int main() {
    while (true) {
        处理文件传输的代码;
    }
}
```

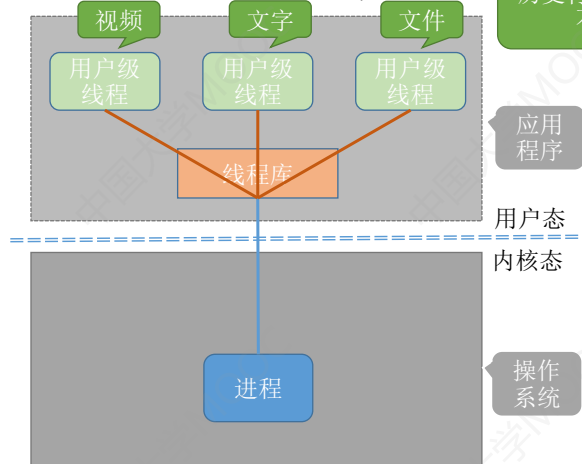
进程3

王道考研/CSKAOYAN.COM

3

## 线程的实现方式

用户级线程 (User-Level Thread, ULT)



历史背景：早期的操作系统（如：早期Unix）只支持进程，不支持线程。当时的“线程”是由线程库实现的

```
int main() {
    int i = 0;
    while (true) {
        if (i==0){处理视频聊天的代码;}
        if (i==1){处理文字聊天的代码;}
        if (i==2){处理文件传输的代码;}
        i = (i+1)%3; //i的值为 0,1,2,0,1,2...
    }
}
```

QQ进程

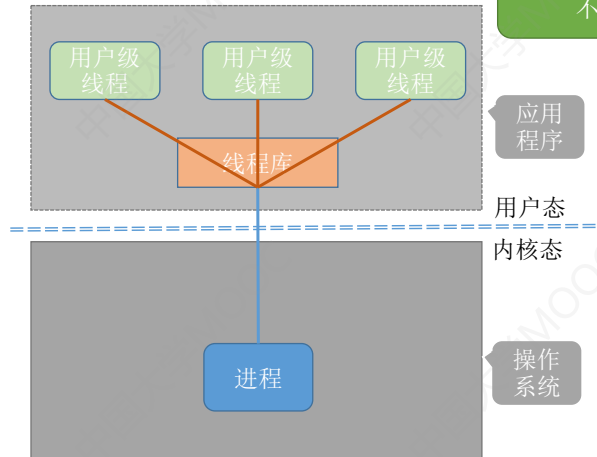
从代码的角度看，线程其实就是一段代码逻辑。上述三段代码逻辑上可以看作三个“线程”。while 循环就是一个最弱智的“线程库”，线程库完成了对线程的管理工作（如调度）。

王道考研/CSKAOYAN.COM

4

## 线程的实现方式

用户级线程 (User-Level Thread, ULT)



历史背景：早期的操作系统（如：早期Unix）只支持进程，不支持线程。当时的“线程”是由线程库实现的

很多编程语言提供了强大的线程库，可以实现线程的创建、销毁、调度等功能。



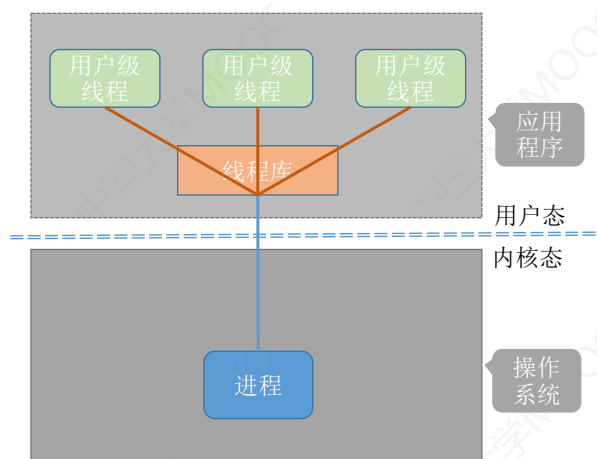
1. 线程的管理工作由谁来完成？
2. 线程切换是否需要CPU变态？
3. 操作系统是否能意识到用户级线程的存在？
4. 这种线程的实现方式有什么优点和缺点？

王道考研/CSKAOYAN.COM

5

## 线程的实现方式

用户级线程 (User-Level Thread, ULT)



1. 用户级线程由应用程序通过线程库实现，所有的**线程管理工作**都由**应用程序负责**（包括线程切换）

2. 用户级线程中，**线程切换可以在用户态下即可完成**，无需操作系统干预。

3. 在用户看来，是有多个线程。但是在操作系统内核看来，并意识不到线程的存在。**“用户级线程”就是“从用户视角看能看到的线程”**

4. 优缺点

优点：用户级线程的切换在用户空间即可完成，不需要切换到核心态，线程管理的系统开销小，效率高

缺点：当一个用户级线程被阻塞后，整个进程都会被阻塞，并发度不高。多个线程不可在多核处理机上并行运行。

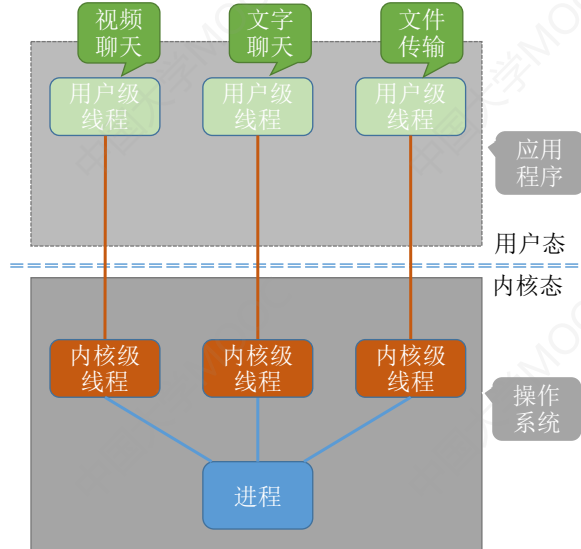
王道考研/CSKAOYAN.COM

6

## 线程的实现方式

内核级线程（Kernel-Level Thread, KLT, 又称“内核支持的线程”）

由操作系统支持的线程



大多数现代操作系统都实现了内核级线程，如 Windows、Linux



1. 线程的管理工作由谁来完成？
2. 线程切换是否需要CPU变态？
3. 操作系统是否能意识到内核级线程的存在？
4. 这种线程的实现方式有什么优点和缺点？

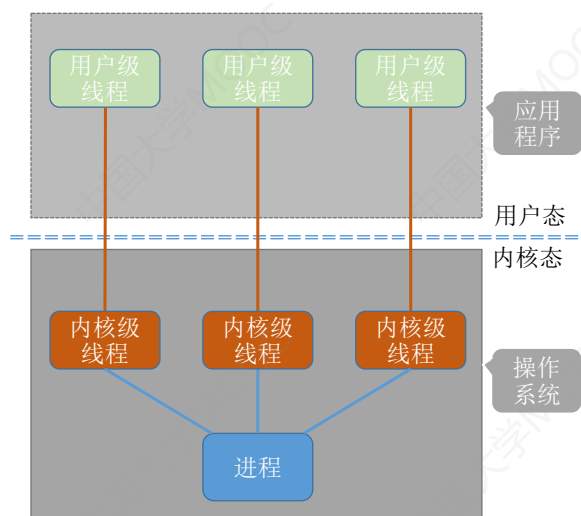
王道考研/CSKAOYAN.COM

7

## 线程的实现方式

内核级线程（Kernel-Level Thread, KLT, 又称“内核支持的线程”）

由操作系统支持的线程



1. 内核级线程的管理工作由操作系统内核完成。

2. 线程调度、切换等工作都由内核负责，因此**内核级线程的切换**必然需要在**核心态**下才能完成。

3. 操作系统会为每个内核级线程建立相应的TCB（Thread Control Block，线程控制块），通过TCB对线程进行管理。“**内核级线程**”就是“**从操作系统内核视角看能看到的线程**”

4. 优缺点

优点：当一个线程被阻塞后，别的线程还可以继续执行，并发能力强。多线程可在多核处理机上并行执行。

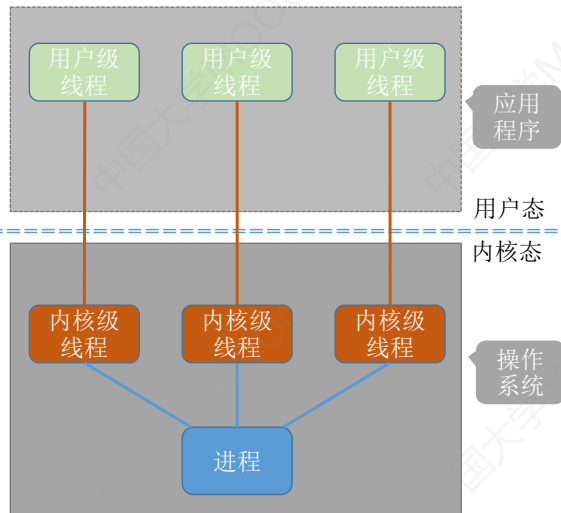
缺点：一个用户进程会占用多个内核级线程，线程切换由操作系统内核完成，需要切换到核心态，因此线程管理的成本高，开销大。

王道考研/CSKAOYAN.COM

8

## 多线程模型

在支持内核级线程的系统中，根据用户级线程和内核级线程的映射关系，可以划分为几种多线程模型



**一对一模型：**一个用户级线程映射到一个内核级线程。每个用户进程有与用户级线程同数量的内核级线程。

优点：当一个线程被阻塞后，别的线程还可以继续执行，并发能力强。多线程可在多核处理机上并行执行。

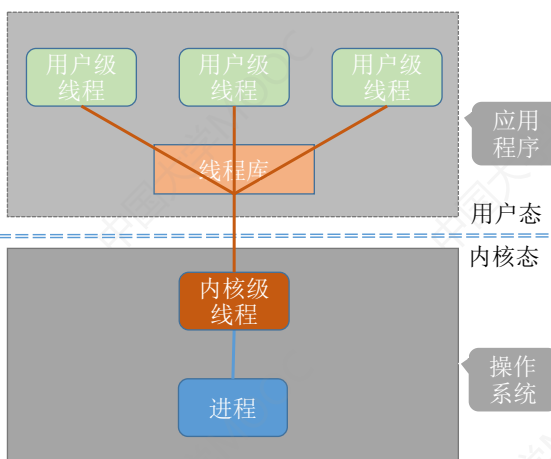
缺点：一个用户进程会占用多个内核级线程，线程切换由操作系统内核完成，需要切换到核心态，因此线程管理的成本高，开销大。

王道考研/CSKAOYAN.COM

9

## 多线程模型

在支持内核级线程的系统中，根据用户级线程和内核级线程的映射关系，可以划分为几种多线程模型



**多对一模型：**多个用户级线程映射到一个内核级线程。且一个进程只被分配一个内核级线程。

优点：用户级线程的切换在用户空间即可完成，不需要切换到核心态，线程管理的系统开销小，效率高

缺点：当一个用户级线程被阻塞后，整个进程都会被阻塞，并发度不高。多个线程不可在多核处理机上并行运行

**重点重点重点：**

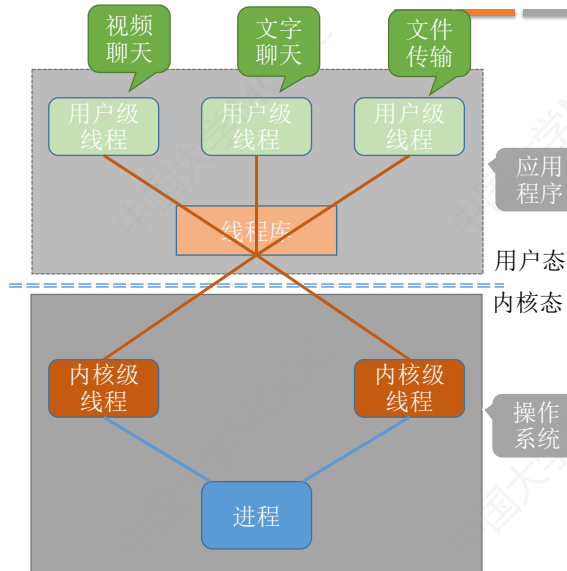
操作系统只“看得见”内核级线程，因此只有**内核级线程才是处理机分配的单位**。

王道考研/CSKAOYAN.COM

10

## 多线程模型

在支持内核级线程的系统中，根据用户级线程和内核级线程的映射关系，可以划分为几种多线程模型



**多对多模型**： $n$  用户及线程映射到  $m$  个内核级线程 ( $n \geq m$ )。每个用户进程对应  $m$  个内核级线程。

克服了多对一模型并发度不高的缺点（一个阻塞全体阻塞），又克服了一对一模型中一个用户进程占用太多内核级线程，开销太大的缺点。

可以这么理解：

用户级线程是“代码逻辑”的载体

内核级线程是“运行机会”的载体

**内核级线程才是处理机分配的单位。**例如：多核 CPU 环境下，左边这个进程最多能被分配两个核。

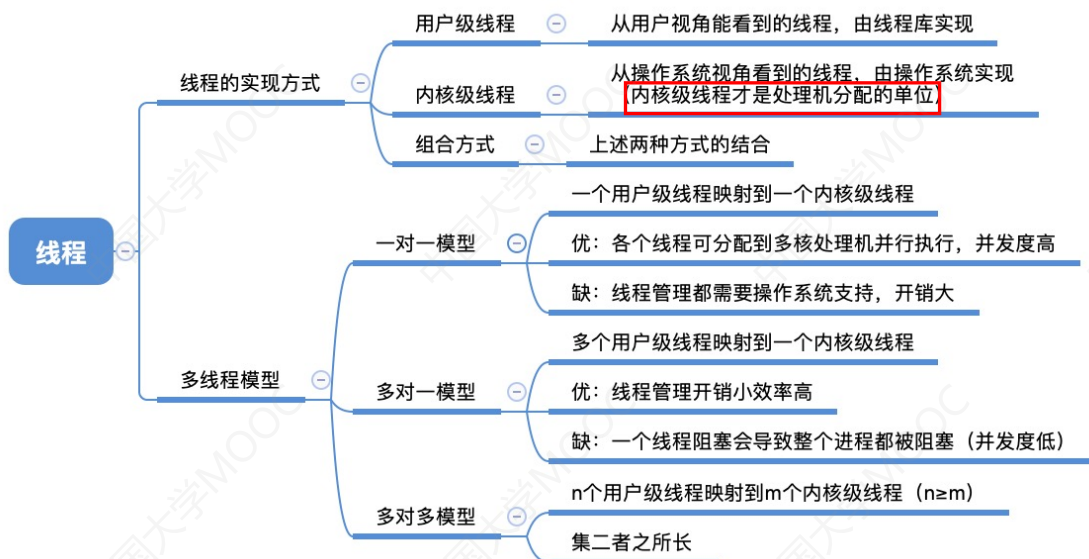
一段“代码逻辑”只有获得了“运行机会”才能被 CPU 执行

内核级线程中可以运行任意一个有映射关系的用户级线程代码，只有两个内核级线程中正在运行的代码逻辑都阻塞时，这个进程才会阻塞

王道考研/CSKAOYAN.COM

11

## 知识回顾与重要考点



王道考研/CSKAOYAN.COM

12



@王道论坛



@王道计算机考研备考



@王道咸鱼老师-计算机考研

@王道楼楼老师-计算机考研



@王道计算机考研



@王道计算机考研



微信视频号

@王道计算机考研



微信公众平台

@王道在线