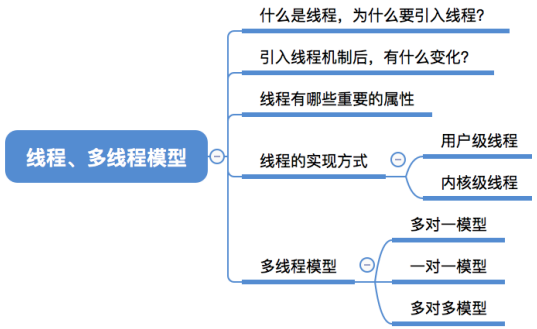
本节内容

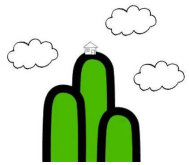
线程概念
多线程模型

知识总览



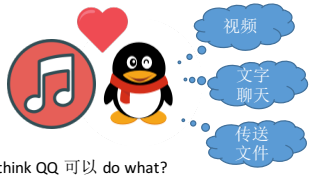
什么是线程，为什么要引入线程？

故事发生在很久以前……



还没引入进程之前，系统中各个程序只能串行执行。

引入了进程之后...

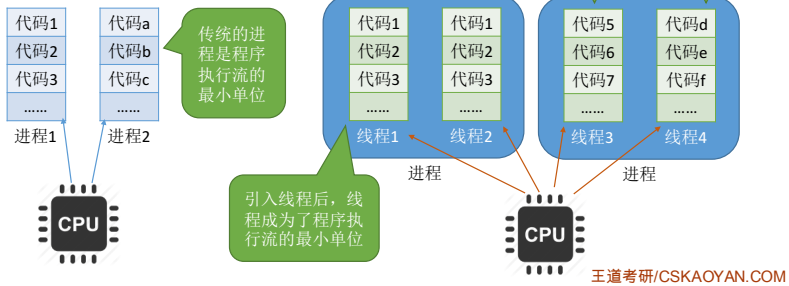


进程是程序的一次执行。但这些功能显然不可能是由一个程序顺序处理就能实现的

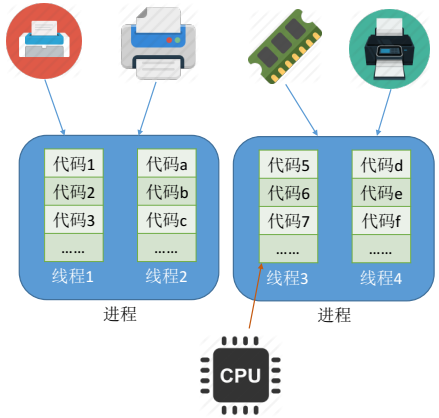
But...Think think QQ 可以 do what?

什么是线程，为什么要引入线程？

有的进程可能需要“同时”做很多事，而传统的进程只能串行地执行一系列程序。为此，引入了“线程”，来增加并发度。



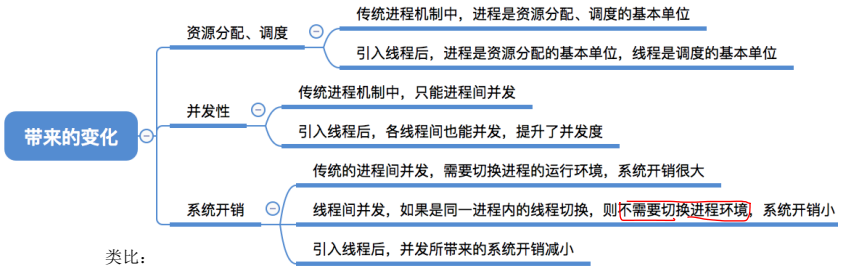
什么是线程，为什么要引入线程？



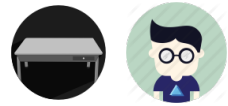
可以把线程理解为“轻量级进程”。
线程是一个基本的CPU执行单元，也是程序执行流的最小单位。
引入线程之后，不仅是进程之间可以并发，进程内的各线程之间也可以并发，从而进一步提升了系统的并发度，使得一个进程内也可以并发处理各种任务（如QQ视频、文字聊天、传文件）

引入线程后，进程只作为除CPU之外的系统资源的分配单元（如打印机、内存地址空间等都是分配给进程的）。
线程则作为处理机的分配单元。

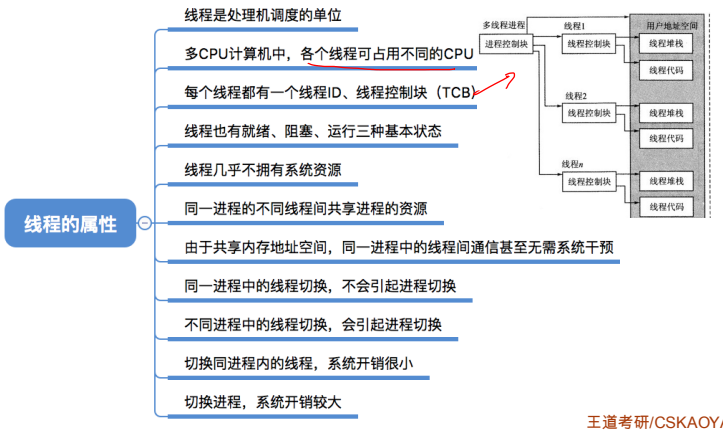
引入线程机制后，有什么变化？



类比：
去图书馆看书。
切换进程运行环境：有一个不认识的人要用桌子，你需要你的书收走，他把自己的书放到桌上
同一进程内的线程切换=你的舍友要用这张书桌，可以不把桌子上的书收走

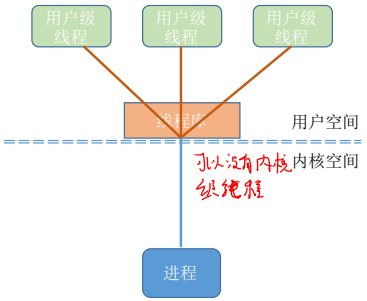


线程的属性



线程的实现方式

用户级线程（User-Level Thread, ULT）

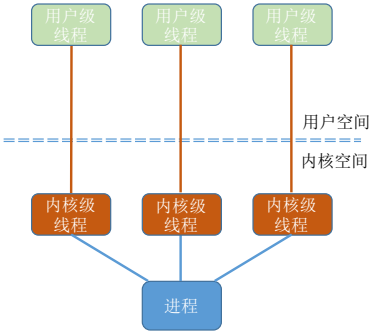


用户级线程由应用程序通过线程库实现。
所有的线程管理工作都由应用程序负责（包括线程切换）
用户级线程中，线程切换可以在用户态下即可完成，无需操作系统干预。
在用户看来，是有多个线程。但是在操作系统内核看来，并意识不到线程的存在。（用户级线程对用户不透明，对操作系统透明）

可以这样理解，“用户级线程”就是“从用户视角看能看到的线程”

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”）

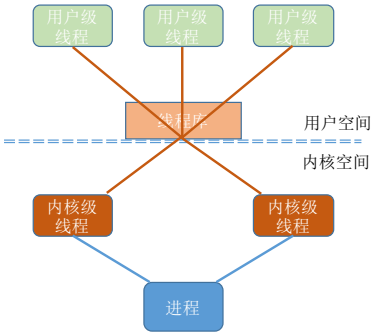


内核级线程的管理工作由操作系统内核完成。线程调度、切换等工作都由内核负责，因此内核级线程的切换必然需要在核心态下才能完成。

可以这样理解，“内核级线程”就是“从操作系统内核视角看能看到的线程”

线程的实现方式

在同时支持用户级线程和内核级线程的系统中，可采用二者组合的方式：将n个用户级线程映射到m个内核级线程上（ n >= m）



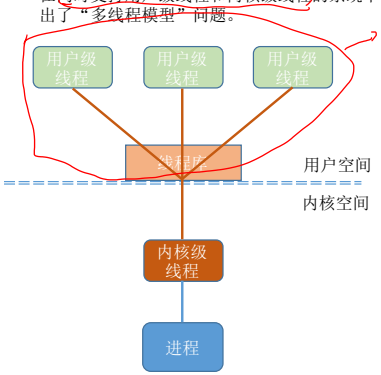
重点重点重点：

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。

例如：左边这个模型中，该进程由两个内核级线程，三个用户级线程，在用户看来，这个进程中有三个线程。但即使该进程在一个4核处理机的计算机上运行，也最多只能被分配到两个核，最多只能有两个用户线程并行执行。

多线程模型

在同时支持用户级线程和内核级线程的系统中，由几个用户级线程映射到几个内核级线程的问题引出了“多线程模型”问题。



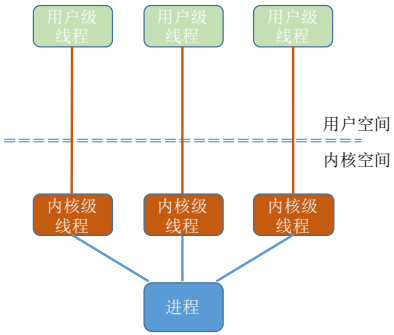
多对一模型：多个用户及线程映射到一个内核级线程。每个用户进程只对应一个内核级线程。

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

多线程模型

在同时支持用户级线程和内核级线程的系统中，由几个用户级线程映射到几个内核级线程的问题引出了“多线程模型”问题。



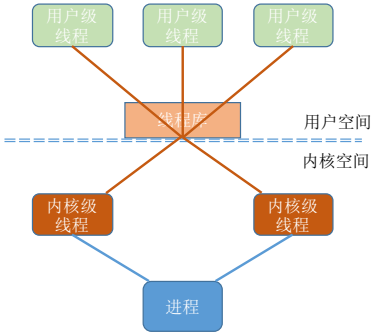
一对一模型：一个用户及线程映射到一个内核级线程。每个用户进程有与用户级线程同数量的内核级线程。

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。

多线程模型

在同时支持用户级线程和内核级线程的系统中，由几个用户级线程映射到几个内核级线程的问题引出了“多线程模型”问题。



多对多模型：n 用户及线程映射到 m 个内核级线程（n >= 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，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。

知识回顾与重要考点