用户线程和内核线程的区别

1 .内核级线程:切换由内核控制，当线程进行切换的时候，由用户态转化为内核态。切换完毕要从内核态返回用户态；可以很好的利用smp，即利用多核cpu。windows线程就是这样的。

2. 用户级线程内核的切换由用户态程序自己控制内核切换,不需要内核干涉，少了进出内核态的消耗，但不能很好的利用多核Cpu,目前Linux pthread大体是这么做的。

线 程的实现可以分为两类：用户级线程(User-Level Thread)和内核线线程(Kernel-Level Thread)，后者又称为内核支持的线程或轻量级进程。在多线程操作系统中，各个系统的实现方式并不相同，在有的系统中实现了用户级线程，有的系统中实 现了内核级线程。

用户线程指不需要内核支持而在用户程序中实现的线程，其不依赖于操作系统核心，应 用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。不需要用户态/核心态切换，速度快，操作系统内核不知道多线程的存在，因此一个线 程阻塞将使得整个进程（包括它的所有线程）阻塞。由于这里的处理器时间片分配是以进程为基本单位，所以每个线程执行的时间相对减少。

内核线程：由操作系统内核创建和撤销。内核维护进程及线程的上下文信息以及线程切换。一个内核线程由于I/O操作而阻塞，不会影响其它线程的运行。Windows NT和2000/XP支持内核线程。

用户线程运行在一个中间系统上面。目 前中间系统实现的方式有两种，即运行时系统（Runtime System）和内核控制线程。“运行时系统”实质上是用于管理和控制线程的函数集合，包括创建、撤销、线程的同步和通信的函数以及调度的函数。这些函数 都驻留在用户空间作为用户线程和内核之间的接口。用户线程不能使用系统调用，而是当线程需要系统资源时，将请求传送给运行时，由后者通过相应的系统调用来 获取系统资源。内核控制线程：系统在分给进程几个轻型进程（LWP），LWP可以通过系统调用来获得内核提供的服务，而进程中的用户线程可通过复用来关联 到LWP，从而得到内核的服务。

以下是用户级线程和内核级线程的区别：

（1）内核支持线程是OS内核可感知的，而用户级线程是OS内核不可感知的。

（2）用户级线程的创建、撤消和调度不需要OS内核的支持，是在语言（如Java）这一级处理的；而内核支持线程的创建、撤消和调度都需OS内核提供支持，而且与进程的创建、撤消和调度大体是相同的。

（3）用户级线程执行系统调用指令时将导致其所属进程被中断，而内核支持线程执行系统调用指令时，只导致该线程被中断。

（4）在只有用户级线程的系统内，CPU调度还是以进程为单位，处于运行状态的进程中的多个线程，由用户程序控制线程的轮换运行；在有内核支持线程的系统内，CPU调度则以线程为单位，由OS的线程调度程序负责线程的调度。

（5）用户级线程的程序实体是运行在用户态下的程序，而内核支持线程的程序实体则是可以运行在任何状态下的程序。

内核线程的优点：

（1）当有多个处理机时，一个进程的多个线程可以同时执行。

缺点：

（1）由内核进行调度。

用户进程的优点：

（1） 线程的调度不需要内核直接参与，控制简单。

（2） 可以在不支持线程的操作系统中实现。

（3） 创建和销毁线程、线程切换代价等线程管理的代价比内核线程少得多。

（4） 允许每个进程定制自己的调度算法，线程管理比较灵活。

（5） 线程能够利用的表空间和堆栈空间比内核级线程多。

（6） 同一进程中只能同时有一个线程在运行，如果有一个线程使用了系统调用而阻塞，那么整个进程都会被挂起。另外，页面失效也会产生同样的问题。

缺点：

（1）资源调度按照进程进行，多个处理机下，同一个进程中的线程只能在同一个处理机下分时复用

========================================================================================================

内核线程

内核线程只运行在内核态，不受用户态上下文的拖累。

处理器竞争：可以在全系统范围内竞争处理器资源；

使用资源：唯一使用的资源是内核栈和上下文切换时保持寄存器的空间

调度：调度的开销可能和进程自身差不多昂贵

同步效率：资源的同步和数据共享比整个进程的数据同步和共享要低一些。

轻量级进程

轻量级进程(LWP)是建立在内核之上并由内核支持的用户线程，它是内核线程的高度抽象，每一个轻量级进程都与一个特定的内核线程关联。内核线程只能由内核管理并像普通进程一样被调度。

轻量级进程由clone()系统调用创建，参数是CLONE\_VM，即与父进程是共享进程地址空间和系统资源。

与普通进程区别：LWP只有一个最小的执行上下文和调度程序所需的统计信息。

处理器竞争：因与特定内核线程关联，因此可以在全系统范围内竞争处理器资源

使用资源：与父进程共享进程地址空间

调度：像普通进程一样调度

用户线程

用户线程是完全建立在用户空间的线程库，用户线程的创建、调度、同步和销毁全又库函数在用户空间完成，不需要内核的帮助。因此这种线程是极其低消耗和高效的。

处理器竞争：单纯的用户线程是建立在用户空间，其对内核是透明的，因此其所属进程单独参与处理器的竞争，而进程的所有线程参与竞争该进程的资源。

使用资源：与所属进程共享进程地址空间和系统资源。

调度：由在用户空间实现的线程库，在所属进程内进行调度

Linux使用的线程库

LinuxThreads是用户空间的线程库，所采用的是线程-进程1对1模型(即一个 用户线程对应一个轻量级进程，而一个轻量级进程对应一个特定 的内核线程)，将线程的调度等同于进程的调度，调度交由内核完成，而线程的创建、同步、销毁由核外线程库完成（LinuxThtreads已绑定到 GLIBC中发行）。

在LinuxThreads中，由专门的一个管理线程处理所有的线程管理工作。当进程第 一次调用pthread\_create()创建线程时就会先 创建(clone())并启动管理线程。后续进程pthread\_create()创建线程时，都是管理线程作为pthread\_create()的调用 者的子线程，通过调用clone()来创建用户线程，并记录轻量级进程号和线程id的映射关系，因此，用户线程其实是管理线程的子线程。

LinuxThreads只支持调度范围为PTHREAD\_SCOPE\_SYSTEM的调度，默认的调度策略是SCHED\_OTHER。

用户线程调度策略也可修改成SCHED\_FIFO或SCHED\_RR方式，这两种方式支持优先级为0-99,而SCHED\_OTHER只支持0。

SCHED\_OTHER 分时调度策略，

SCHED\_FIFO 实时调度策略，先到先服务

SCHED\_RR 实时调度策略，时间片轮转

SCHED\_OTHER是普通进程的，后两个是实时进程的（一般的进程都是普通进程，系 统中出现实时进程的机会很少）。SCHED\_FIFO、 SCHED\_RR优先级高于所有SCHED\_OTHER的进程，所以只要他们能够运行，在他们运行完之前，所有SCHED\_OTHER的进程的都没有得到 执行的机会。