之前讲解过内核线程、轻量级进程、用户线程三种线程概念解惑（线程≠轻量级进程）,但是一直对其中提到的线程的实现模型比较迷惑,这次就花了点时间怎么学习了一下子

1 线程的3种实现方式

在传统的操作系统中，拥有资源和独立调度的基本单位都是进程。在引入线程的操作系统中，线程是独立调度的基本单位，进程是资源拥有的基本单位。在同一进程中，线程的切换不会引起进程切换。在不同进程中进行线程切换,如从一个进程内的线程切换到另一个进程中的线程时，会引起进程切换

根据操作系统内核是否对线程可感知，可以把线程分为内核线程和用户线程

名称	描述
用户级线程(User-LevelThread, ULT)	由应用程序所支持的线程实现,内核意识不到用户级线程的实现
内核级线程(Kemel-LevelThread, KLT)	内核级线程又称为内核支持的线程

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

有些情况下,也把内核级线程叫做轻量级进程(LWP),但是这个是一个不准备的描述,其实LWP的术语是借自于SVR4/MP和Solaris 2.x系统中,有些系统将LWP称为虚拟处理器,将之称为轻量级进程的原因可能是,在内核线程的支持下，LWP是独立的调度单元，就像普通的进程一样。所以LWP的最大特点还是每个LWP都有一个内核线程支持

2 用户级线程(多对一模型)

2.1 线程的用户级线程实现方式

在用户级线程中，

有关线程管理的所有工作都由应用程序完成，内核意识不到线程的存在. 应用程序可以通过使用线程库设计成多线程程序. 通常，应用程序从单线程起始，在该线程中开始运行，在其运行的任何时刻，可以通过调用线程库中的派生例程创建一个在相同进程中运行的新线程。

用户级线程仅存在于用户空间中，此类线程的创建、撤销、线程之间的同步与通信功能，都无须利用系统调用来实现。用户进程利用线程库来控制用户线程。由于线程在进程内切换的规则远比进程调度和切换的规则简单，不需要用户态/核心态切换，所以切换速度快。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为了在操作系统中加入线程支持，采用了在用户空间增加运行库来实现线程，这些运行库被称为“线程包”，用户线程是不能被操作系统所感知的。用户线程多见于一些历史悠久的操作系统，例如Unix操作系统

用户级线程驻留在用户空间或模式。运行时库管理这些线程，它也位于用户空间。它们对于操作系统是不可见的，因此无法被调度到处理器内核。每个线程并不具有自身的线程上下文。因此，就线程的同时执行而言，任意给定时刻每个进程只能够有一个线程在运行，而且只有一个处理器内核会被分配给该进程。对于一个进程，可能有成千上万个用户级线程，但是它们对系统资源没有影响。运行时库调度并分派这些线程。

下图说明了用户级线程的实现方式，

如同在图中看到的那样，库调度器从进程的多个线程中选择一个线程，然后该线程和该进程允许的一个内核线程关联起来。内核线程将被操作系统调度器指派到处理器内核。用户级线程是一种”多对一”的线程映射

2.2 用户级线程的特点

内核对线程包一无所知。从内核角度考虑，就是按正常的方式管理，即单线程进程（存在运行时系统）

2.3 用户级线程的优点

用户线程的优点主要有

1. 可以在不支持线程的操作系统中实现。

2. 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，因为保存线程状态的过程和调用程序都只是本地过程

3. 允许每个进程定制自己的调度算法，线程管理比较灵活。这就是必须自己写管理程序，与内核线程的区别

4. 线程能够利用的表空间和堆栈空间比内核级线程多

5. 不需要陷阱，不需要上下文切换，也不需要对内存高速缓存进行刷新，使得线程调用非常快捷

6. 线程的调度不需要内核直接参与，控制简单。

2.4 用户线程的缺点

用户线程的缺点主要有

1. 线程发生I/O或页面故障引起的阻塞时，如果调用阻塞系统调用则内核由于不知道有多线程的存在，而会阻塞整个进程从而阻塞所有线程，因此同一进程中只能同时有一个线程在运行

2. 页面失效也会产生类似的问题。

3. 一个单独的进程内部，没有时钟中断，所以不可能用轮转调度的方式调度线程

4. 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

补充
在用户级线程中，每个进程里的线程表由运行时系统管理。当一个线程转换到就绪状态或阻塞状态时，在该线程表中存放重新启动该线程所需的信息，与内核在进程表中存放的进程的信息完全一样

3 内核级线程

3.1 线程的内核级线程实现

在内核级线程中，

内核线程建立和销毁都是由操作系统负责、通过系统调用完成的。在内核的支持下运行，无论是用户进程的线程，或者是系统进程的线程，他们的创建、撤销、切换都是依靠内核实现的。

线程管理的所有工作由内核完成，应用程序没有进行线程管理的代码，只有一个到内核级线程的编程接口. 内核为进程及其内部的每个线程维护上下文信息，调度也是在内核基于线程架构的基础上完成。图2-2(b)说明了内核级线程的实现方式。

内核线程驻留在内核空间，它们是内核对象。有了内核线程，每个用户线程被映射或绑定到一个内核线程。用户线程在其生命期内都会绑定到该内核线程。一旦用户线程终止，两个线程都将离开系统。这被称作”一对一”线程映射，

1. 线程的创建、撤销和切换等，都需要内核直接实现，即内核了解每一个作为可调度实体的线程

2. 这些线程可以在全系统内进行资源的竞争

3. 内核空间内为每一个内核支持线程设置了一个线程控制块（TCB），内核根据该控制块，感知线程的存在，并进行控制

如图所示,即内核级线程的实现方式, 每个用户线程都直接与一个内核线程相关联.

操作系统调度器管理、调度并分派这些线程。运行时库为每个用户级线程请求一个内核级线程。操作系统的内存管理和调度子系统必须要考虑到数量巨大的用户级线程。您必须了解每个进程允许的线程的最大数目是多少。操作系统为每个线程创建上下文。进程的每个线程在资源可用时都可以被指派到处理器内核。

3.2 内核线程的特点

当某个线程希望创建一个新线程或撤销一个已有线程时，它进行一个系统调用

3.3 内核线程的优点

内核线程的优点:

1. 多处理器系统中，内核能够并行执行同一进程内的多个线程

2. 如果进程中的一个线程被阻塞，能够切换同一进程内的其他线程继续执行（用户级线程的一个缺点）

3. 所有能够阻塞线程的调用都以系统调用的形式实现，代价可观

4. 当一个线程阻塞时，内核根据选择可以运行另一个进程的线程，而用户空间实现的线程中，运行时系统始终运行自己进程中的线程

5. 信号是发给进程而不是线程的，当一个信号到达时，应该由哪一个线程处理它？线程可以“注册”它们感兴趣的信号

4 组合方式

在一些系统中，使用**组合方式的多线程实现**,线程创建完全在用户空间中完成，线程的调度和同步也在应用程序中进行. 一个应用程序中的多个用户级线程被映射到一些（小于或等于用户级线程的数目）内核级线程上。

下图说明了用户级与内核级的组合实现方式,在这种模型中，每个内核级线程有一个可以轮流使用的用户级线程集合

posix线程调度是一个混合模型，很灵活，足以在标准的特定实现中支持用户级和内核级的线程。模型中包括两级调度–线程及和内核实体级。线程级与用户级线程类似，内核实体由内核调度。由线程库来决定它需要多少内核实体，以及他们是如何映射的。

POSIX 引入了一个线程调度竞争范围(thread-scheduling contention scope)的概念，这个. 概念赋予了程序员一些控制权，使它们可以控制怎样将内核实体映射为线程。线程的contentionscope属性可是PTHREAD_SCOPE_PROCESS,也可以是PTHREAD_SCOPE_SYSTEM。带有PTHREAD_SCOPE_PROCESS属性的线程与它所在的进程中的其他线程竞争处理器资源。带有PTHREAD_SCOPE_SYSTEM属性的线程很像内核级线程，他们在全系统的范围内竞争处理器资源。POSIX的一种映射方式将PTHREAD_SCOPE_SYSTEM线程和内核实体之间绑定起来。

内核级线程创建时先设置线程属性PTHREAD_SCOPE_SYSTEM，代码如下，:

```
pthread_attr_t attr;
```

```
pthread_attr_init(&attr);
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM); //设置内核级的线程,以获取较高的响应速度
//创建线程

ret = pthread_create(&iAcceptThreadId, &attr, AcceptThread, NULL);
```

POSIX的标准中定义了两个值:

PTHREAD_SCOPE_SYSTEM和PTHREAD_SCOPE_PROCESS，前者表示与系统中所有线程一起竞争CPU时间，后者表示仅与同进程中的线程竞争CPU

默认为PTHREAD_SCOPE_PROCESS。目前LinuxThreads仅实现了PTHREAD_SCOPE_SYSTEM一值。

关于线程的绑定，牵涉到另外一个概念：轻进程（LWP：Light Weight Process）。轻进程可以理解为内核线程，它位于用户层和系统层之间。系统对线程资源的分配、对线程的控制是通过轻进程来实现的，一个轻进程可以控制一个或多个线程。默认状况下，启动多少轻进程、哪些轻进程来控制哪些线程是由系统来控制的，这种状况即称为非绑定的。绑定状况下，则顾名思义，即某个线程固定的”绑”在一个轻进程之上。被绑定的线程具有较高的响应速度，这是因为CPU时间片的调度是面向轻进程的，绑定的线程可以保证在需要的时候它总有一个轻进程可用。通过设置被绑定的轻进程的优先级和调度级可以使得绑定的线程满足诸如实时反应之类的要求。

设置线程绑定状态的函数为pthread_attr_setscope，它有两个参数，第一个是指向属性结构的指针，第二个是绑定类型，它有两个取值：PTHREAD_SCOPE_SYSTEM（绑定的）和PTHREAD_SCOPE_PROCESS（非绑定的）。

5 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。