2.2 线程

在传统的操作系统中，每一个进程都有一个地址空间和一个单一的控制线程。

事实上，这基本就定义了一个进程是什么。然而，在许多情形下，有多个平行运行的，它们看起来就像彼此独立的进程(除了共享地址空间这一点)，在同一地址空间的控制线程是必需的。在接下来的部分我们将会讨论这些情形以及它们的含义。

2.2.1 线程的使用

人们为什么会想到在进程中使用线程呢？事实证明线程，也可以称作迷你进程的存在是合理的。让我们现在对线程做几个测试。使用线程的最主要的原因是因为在许多应用程序中，很多活动都是同时进行的，它们中有些可能会有时阻塞。通过将应用程序分解为几个顺序的，平行运行的线程可以使程序模型变得简单。

我们以前也见过这种讨论，准确的来说是关于进程存在的合理性。在这里，我们不讨论中断，定时器，和上下文切换，我们来考虑进程的并行。在这里我们只给线程加了一个新的特性：能够在并行的实体之间分享地址空间和彼此拥有的数据。这个特性对于特定的应用程序来说是必需的，这也就是为什么多进程在这里不起作用的原因：进程与进程之间不共享地址空间。

线程的第二个优点是跟进程相比，它们更轻量级，从而能够比进程更快更简单地创建和销毁。在许多系统中，创建一个线程比创建一个进程要快10-100倍。尤其是当系统中的线程数量随时间而快速的变化时，线程的这种特性是非常有用的。

线程存在的第三个理由是性能。从CPU的角度来看，线程并没有提升系统的性能。但是考虑到系统中存在大量的计算和IO操作时，线程的存在可以让这些操作同时进行，从而能够大大提高应用程序的速度。

最后，线程的存在对于运行在多核CPU上的系统来说是非常有用的，在多核CPU上能够实现真正的并行。我们将会在第八章讨论这个话题。

通过几个示例可以更轻松的理解为什么线程的存在是非常有用的。来看第一个示例，我们来考虑一个字符处理程序。字符处理程序通常将已经创建好的的文档按照实际打印的格式合理排版后显示在显示屏上。

2.2.5 线程在内核中的实现

现在让我们来看内核是如何实现和管理线程的。在图2-16(b)中我们可以看到在进程中没有运行时系统，同样的也没有线程表。内核使用线程表来追踪系统中的所有线程。一个线程通过内核调用更新内核线程表来实现一个新的线程的创建或者一个已有的线程的销毁动作。

内核的线程表保存着每一个线程的寄存器信息，状态信息以及其他信息。这些信息和用户层线程的信息是一样的，它们唯一的不同之处在于一个保存在内核中，而另一个保存在用户空间中。线程表信息实际上是传统内核用来保存单线程进程的信息的一个子集，即进程状态。内核除了在维护着线程表来实现线程，同样也继续维护着传统进程表来追踪进程。

所有可能阻塞一个线程的调用都被实现为系统调用，与运行系统程序的调用相比，这些系统调用的开销相当大。当一个线程阻塞时，内核可以选择运行同一个进程的其他就绪线程或者其他进程中的就绪进程。与用户层线程相比，当一个用户层线程阻塞时，运行系统只会选择继续运行该线程的父进程中的其他线程，除非内核将CPU的控制权从父进程那里拿走或者父进程中没有其他就绪线程可以运行。

由于内核中线程的创建和销毁会产生相当大的系统开销，一些系统采用比较环保的做法来回收它们的线程。当一个线程被销毁时，它被标记为不可运行的，但是它的内核数据结构没有受到影响。当后面有一个新的线程要被创建时，一个旧的的线程就会被激活，这样可以节省一些开销。线程回收对于用户层线程来说也是可以实现的，但是由于用户层的线程管理所造成开销与内核层比起来相当小，所以这个功能对用户层线程来说是没必要的。

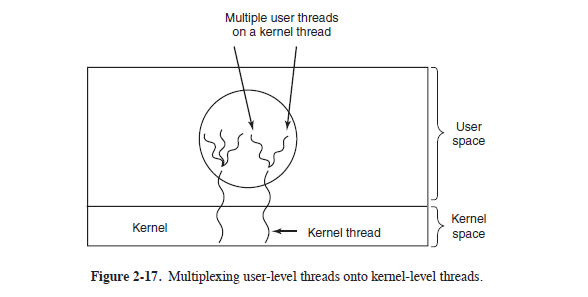
内核线程不需要任何新的非阻塞系统调用。除此之外，如果进程中的一个线程遇到分页阻碍，那么内核可以很轻松地检查当前的进程中是否有其他就绪线程，如果有，那么转而运行这些就绪进程，同时等待上一个线程需要的分页从磁盘中加载到内存中。内核线程最大的缺点就是系统调用所造成的开销太大，因此如果线程操作(创建，终止)很频繁的话，那么系统性能就会收到影响。

尽管内核线程解决了一些问题，但它们并没有解决所有的问题。举个例子，当一个多线程进程创建一个新进程的时候会发生什么？新的进程拥有跟旧进程一样的线程数还是只有一个？在许多情况下，最后的选择取决于新进程接下来要做什么。如果新进程要执行exec系统调用来启动一个新的程序，那么单线程是最好的选择。但是如果新进程继续执行当前的程序，那么复制所有的进程会是最好的选择。

另一种情形是信号量。我们还记得信号量是发送给进程而不是线程的，至少在经典模型中是这样的。当进程接受到一个信号量时，哪个线程应该处理这个信号量？可能线程可以注册它们感兴趣的某一类信号量，因此当一个信号量进来时可以被传递给声明过想要该信号量的线程。但是如果有两个或两个以上的线程同时注册了同样类型的信号量呢？这里只介绍了两种线程问题，实际上还有很多线程问题。

2.2.6 线程的混合实现

人们尝试了不同的方法去综合用户层线程和内核线程的优点。一种方式是使用内核线程然后在其基础上有选择的将一些内核线程扩展成多个用户层线程，如图2-17所示。



通过这种设计，内核就可以只关注内核线程和调度内核线程。基于其中的一些线程可能会扩展出多个用户层线程。这些用户层线程创建，销毁，调度，就像运行在无多线程能力的操作系统上的进程中的用户层线程一样。在这种模型中，每一个内核线程都有一组用户层线程轮流使用它。

2.2.7 调度程序激活机制

尽管内核线程在一些关键方面比用户层线程表现的更好，但它们毫无疑问也更慢。因此，研究者们在寻求更好的解决方法来在不放弃内核线程的优良特性下来改善这些情况。下面我们将会介绍一种由Anderson et al提出的方案，称作调度程序激活机制