2.2.7 调度程序激活机制

尽管内核级线程在一些关键点上优于用户级线程，但无可争议的是内核级线程的速度慢。因此，研究人员一直在寻找在保持其优良特性的前提下改进其速度的方法。下面我们将介绍Anderson等人（1992）设计的这样一种方法，称为*调度程序激活*（scheduler activation）机制。Edler等人（1988）以及Scott等人（1990）就相关的工作进行了深入讨论。

调度程序激活工作的目标是模拟内核线程的功能，但是为线程包提供通常在用户空间中才能实现的更好的性能和更大的灵活性。特别地，如果用户线程从事某种系统调用时是安全的，那就不应该进行专门的非阻塞调用或者进行提前检查。无论如何，如果线程阻塞在某个系统调用或页面故障上，只要在同一个进程中有任何就绪的线程，就应该有可能运行其他的线程。

由于避免了在用户空间和内核空间之间的不必要转换，从而提高了效率。例如，如果某个线程由于等待另一个线程的工作而阻塞，此时没有理由请求内核，这样就减少了内核-用户转换的开销。用户空间的运行时系统可以阻塞同步的线程而另外调度一个新线程。

当使用调度程序激活机制时，内核给每个进程安排一定数量的虚拟处理器，并且让（用户空间）运行时系统将线程分配到处理器上。这一机制也可以用在多处理器中，此时虚拟处理器可能成为真实的CPU。分配给一个进程的虚拟处理器的初始数量是一个，但是该进程可以申请更多的处理器并且在不用时退回。内核也可以取回已经分配出去的虚拟处理器，以便把它们分给需要更多处理器的进程。

使该机制工作的基本思路是，当内核了解到一个线程被阻塞之后（例如，由于执行了一个阻塞系统调用或者产生了一个页面故障），内核通知该进程的运行时系统，并且在堆栈中以参数形式传递有问题的线程编号和所发生事件的一个描述。内核通过在一个已知的起始地址启动运行时系统，从而发出了通知，这是对UNIX中信号的一种粗略模拟。这个机制称为*上行调用*（upcall）。

一旦如此激活，运行时系统就重新调度其线程，这个过程通常是这样的：把当前线程标记为阻塞并从就绪表中取出另一个线程，设置其寄存器，然后再启动之。稍后，当内核知道原来的线程又可运行时（例如，原先试图读取的管道中有了数据，或者已经从磁盘中读入了故障的页面），内核就又一次上行调用运行时系统，通知它这一事件。此时该运行时系统按照自己的判断，或者立即重启动被阻塞的线程，或者把它放入就绪表中稍后运行。

在某个用户线程运行的同时发生一个硬件中断时，被中断的CPU切换进核心态。如果被中断的进程对引起该中断的事件不感兴趣，比如，是另一个进程的I/O完成了，那么在中断处理程序结束之后，就把被中断的线程恢复到中断之前的状态。不过，如果该进程对中断感兴趣，比如，是该进程中的某个线程所需要的页面到达了，那么被中断的线程就不再启动，代之为挂起被中断的线程。而运行时系统则启动对应的虚拟CPU，此时被中断线程的状态保存在堆栈中。随后，运行时系统决定在该CPU上调度哪个线程：被中断的线程、新就绪的线程还是某个第三种选择。

调度程序激活机制的一个目标是作为上行调用的信赖基础，这是一种违反分层次系统内在结构的概念。通常，n层提供$n+1$层可调用的特定服务，但是n层不能调用$n+1$层中的过程。上行调用并不遵守这个基本原理。

2.2.8 弹出式线程

在分布式系统中经常使用线程。一个有意义的例子是如何处理到来的消息，例如服务请求。传统的方法是将进程或线程阻塞在一个receive系统调用上，等待消息到来。当消息到达时，该系统调用接收消息，并打开消息检查其内容，然后进行处理。

不过，也可能有另一种完全不同的处理方式，在该处理方式中，一个消息的到达导致系统创建一个处理该消息的线程，这种线程称为*弹出式线程*，如图2-18所示。弹出式线程的关键好处是，由于这种线程相当新，没有历史——没有必须存储的寄存器、堆栈诸如此类的内容，每个线程从全新开始，每一个线程彼此之间都完全一样。这样，就有可能快速创建这类线程。对该新线程指定所要处理的消息。使用弹出式线程的结果是，消息到达与处理开始之间的时间非常短。