2.2.3 POSIX线程

为实现可移植的线程程序，IEEE在IEEE标准1003.1c中定义了线程的标准。它定义的线程包叫做Pthread。大部分UNIX系统都支持该标准。这个标准定义了超过60个函数调用，如果在这里列举一遍就太多了。取而代之的是，我们将仅仅描述一些主要的函数，以说明它是如何工作的。图2-14中列举了这些函数调用。

所有Pthread线程都有某些特性。每一个都含有一个标识符、一组寄存器（包括程序计数器）和一组存储在结构中的属性。这些属性包括堆栈大小、调度参数以及使用线程需要的其他项目。

线程调用	描　述
Pthread_create	创建一个新线程
Pthread_exit	结束调用的线程
Pthread_join	等待一个特定的线程退出
Pthread_yield	释放CPU来运行另外一个线程
Pthread_attr_init	创建并初始化一个线程的属性结构
Pthread_attr_destroy	删除一个线程的属性结构

图2-14　一些Pthread的函数调用

创建一个新线程需要使用pthread_create调用。新创建的线程的线程标识符作为函数值返回。这种调用有意看起来很像fork系统调用，其中线程标识符起着PID的作用，而这么做的目的主要是为了标识在其他调用中引用的线程。

当一个线程完成分配给它的工作时，可以通过调用pthread_exit来终止。这个调用终止该线程并释放它的栈。

一般一个线程在继续运行前需要等待另一个线程完成它的工作并退出。可以通过pthread_join线程调用来等待别的特定线程的终止。而要等待线程的线程标识符作为一个参数给出。

有时会出现这种情况：一个线程逻辑上没有阻塞，但感觉上它已经运行了足够长时间并且希望给另外一个线程机会去运行。这时可以通过调用pthread_yield完成这一目标。而进程中没有这种调用，因为假设进程间会有激烈的竞争性，并且每一个进程都希望获得它所能得到的所有的CPU时间。但是，由于同一进程中的线程可以同时工作，并且它们的代码总是由同一个程序员编写的，因此，有时程序员希望它们能互相给对方一些机会去运行。

下面两个线程调用是处理属性的。Pthread_attr_init建立关联一个线程的属性结构并初始化成默认值。这些值（例如优先级）可以通过修改属性结构中的域值来改变。

最后，pthread_attr_destroy删除一个线程的属性结构，释放它占用的内存。它不会影响调用它的线程。这些线程会继续存在。

为了更好地了解Pthread是如何工作的，考虑图2-15提供的简单例子。这里主程序在宣布它的意图之后，循环NUMBER_OF_THREADS次，每次创建一个新的线程。如果线程创建失败，会打印出一条错误信息然后退出。在创建完所有线程之后，主程序退出。

当创建一个线程时，它打印一条一行的发布信息，然后退出。这些不同信息交错的顺序是不确定的，并且可能在连续运行程序的情况下发生变化。

前面描述的Pthread调用无论如何也不是屈指可数的这几个，还有许多的调用。我们会在讨论“进程与线程同步”之后再来研究其他一些Pthread调用。

2.2.4 在用户空间中实现线程

有两种主要的方法实现线程包：在用户空间中和在内核中。这两种方法互有利弊，不过混合实现方式也是可能的。我们现在介绍这些方法，并分析它们的优点和缺点。

第一种方法是把整个线程包放在用户空间中，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。这种方法第一个，也是最明显的优点是，用户级线程包可以在不支持线