如果两个或多个进程共享其全部或大部分地址空间，进程和线程之间的差别就变得模糊起来，但无论怎样，两者的差别还是有的。共享一个公共地址空间的两个进程仍旧有各自的打开文件、报警定时器以及其他一些单个进程的特性，而在单个进程中的线程，则共享进程全部的特性。另外，共享一个公共地址空间的多个进程决不会拥有用户级线程的效率，这一点是不容置疑的，因为内核还同其管理密切相关。

Pthread中的互斥

Pthread提供许多可以用来同步线程的函数。其基本机制是使用一个可以被锁定和解锁的互斥量来保护每个临界区。一个线程如果想要进入临界区，它首先尝试锁住相关的互斥量。如果互斥量没有加锁，那么这个线程可以立即进入，并且该互斥量被自动锁定以防止其他线程进入。如果互斥量已经被加锁，则调用线程被阻塞，直到该互斥量被解锁。如果多个线程在等待同一个互斥量，当它被解锁时，这些等待的线程中只有一个被允许运行并将互斥量重新锁定。这些互斥锁不是强制性的，而是由程序员来保证线程正确地使用它们。

与互斥量相关的主要函数调用如图2-30所示。就像所期待的那样，可以创建和撤销互斥量。实现它们的函数调用分别是pthread_mutex_init与pthread_mutex_destroy。也可以通过pthread_mutex_lock给互斥量加锁，如果该互斥量已被加锁时，则会阻塞调用者。还有一个调用可以用来尝试锁住一个互斥量，当互斥量已被加锁时会返回错误代码而不是阻塞调用者。这个调用就是pthread_mutex_trylock。如果需要的话，该调用允许一个线程有效地忙等待。最后，pthread_mutex_unlock用来给一个互斥量解锁，并在一个或多个线程等待它的情况下正确地释放一个线程。互斥量也可以有属性，但是这些属性只在某些特殊的场合下使用。

线程调用	描　述
pthread_mutex_init	创建一个互斥量
pthread_mutex_destroy	撤销一个已存在的互斥量
pthread_mutex_lock	获得一个锁或阻塞
pthread_mutex_trylock	获得一个锁或失败
pthread_mutex_unlock	释放一个锁

图2-30　一些与互斥量相关的pthread调用

除互斥量之外，pthread提供了另一种同步机制：条件变量。互斥量在允许或阻塞对临界区的访问上是很有用的，条件变量则允许线程由于一些未达到的条件而阻塞。绝大部分情况下这两种方法是一起使用的。现在让我们进一步地研究线程、互斥量、条件变量之间的关联。

举一个简单的例子，再次考虑一下生产者-消费者问题：一个线程将产品放在一个缓冲区内，由另一个线程将它们取出。如果生产者发现缓冲区中没有空槽可以使用了，它不得不阻塞起来直到有一个空槽可以使用。生产者使用互斥量可以进行原子性检查，而不受其他线程干扰。但是当发现缓冲区已经满了以后，生产者需要一种方法来阻塞自己并在以后被唤醒。这便是条件变量做的事了。

与条件变量相关的pthread调用如图2-31所示。就像你可能期待的那样，这里有专门的调用用来创建和撤销条件变量。它们可以有属性，并且有不同的调用来管理它们（图中没有显示）。与条件变量相关的最重要的两个操作是pthread_cond_wait和pthread_cond_signal。前者阻塞调用线程直到另一其他线程向它发信号（使用后一个调用）。当然，阻塞与等待的原因不是等待与发信号协议的一部分。被阻塞的线程经常是在等待发信号的线程去做某些工作、释放某些资源或是进行其他的一些活动。只有完成后被阻塞的线程才可以继续运行。条件变量允许这种等待与阻塞原子性地进行。当有多个线程被阻塞并等待同一个信号时，可以使用pthread_cond_broadcast调用。

线程调用	描　述
pthread_cond_init	创建一个条件变量
pthread_cond_destroy	撤销一个条件变量
pthread_cond_wait	阻塞以等待一个信号
pthread_cond_signal	向另一个线程发信号来唤醒它
pthread_cond_broadcast	向多个线程发信号来让它们全部唤醒

图2-31　一些与条件变量相关的pthread调用

条件变量与互斥量经常一起使用。这种模式用于让一个线程锁住一个互斥量，然后当它不能获得它期待的结果时等待一个条件变量。最后另一个线程会向它发信号，使它可以继续执行。pthread_cond_wait原子性地调用并解锁它持有的互斥量。由于这个原因，互斥量是参数之一。

值得指出的是，条件变量（不像信号量）不会存在内存中。如果将一个信号量传递给一个没有线程