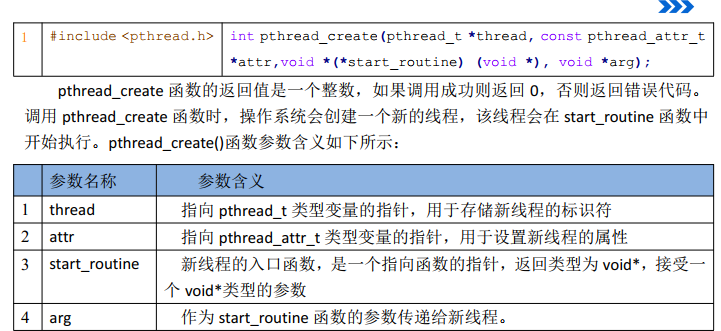
# 9.1 线程的创建 （线程：操作系统中分配资源的最小单位）

**（1）线程是什么？和进程的区别？**

线程是进程中的一个执行流程，相对于进程， 线程的创建和销毁开销较小， 可以更加高效地利用系统资源， 提高程序的并发性能和响应能力

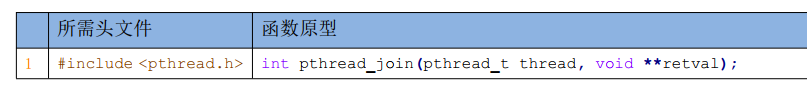
进程是一个独立的执行单元， 具有自己的内存空间和系统资源， 而线程则是在进程内部共享这些资源的执行路径。好比一个厨房和众多的厨师。

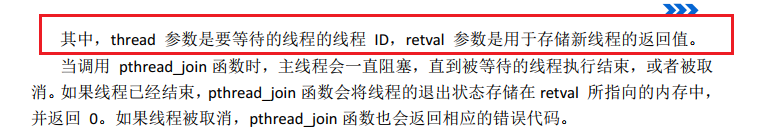
（2） 线程的创建 pthread\_create()函数



# 9.2 线程的回收

为了保证新创建的线程执行完，主程序在技术，就需要**pthread\_join**（）函数，起作用就是等待一个指定的线程结束。

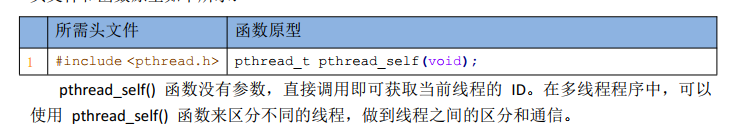




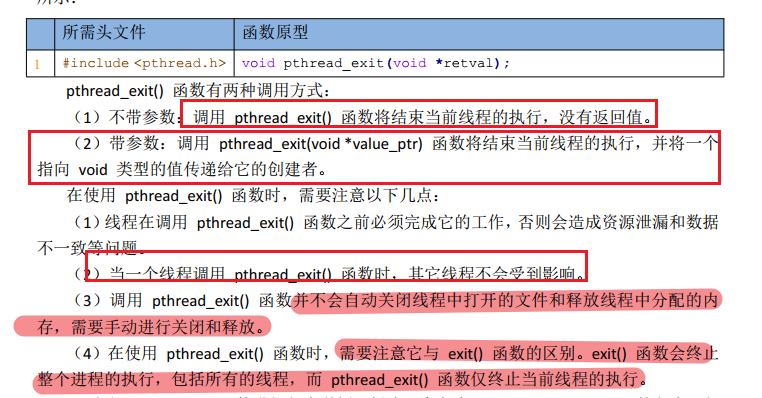
# 9.3 获取线程ID

使用 **pthread\_create** 函数创建一个子线程之后， 会将线程 ID 保存在 thread 参数中， 线程 ID的作用与进程 ID 类似， 用来标识不同的线程， 通常被用来控制线程的创建、 销毁、 同步等操作， 例如上一小节就是根据线程 ID 进行线程的回收

pthread\_self（）函数可以进行线程ID的获取。

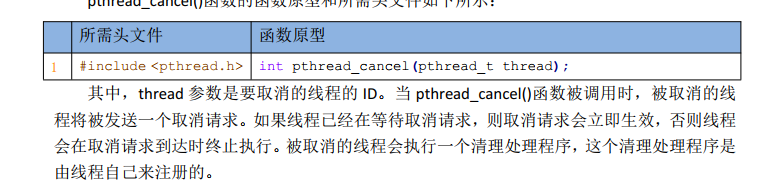


# 9.4 线程的终止 **pthread\_exit()**函数用于终止调用它的线程。回收资源



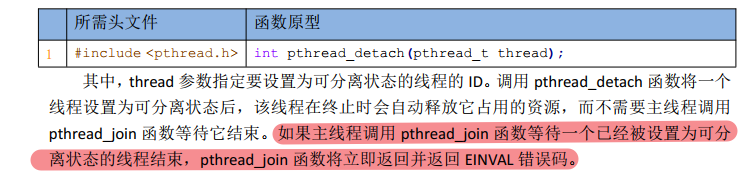
# 9.5 取消线程 **pthread\_cancel()**函数用于取消正在运行的线程的函数

使用pthread\_cancel()函数来取消正在进行的线程时，并不会立即取消，而是在适当的时候取消。



# 9.6 分离线程 **pthread\_detach** 函数用于将一个线程设置为可分离状态。

可分离状态的线程在终止时自动释放其占用的资源， 而无需主线程显式地等待它们结束

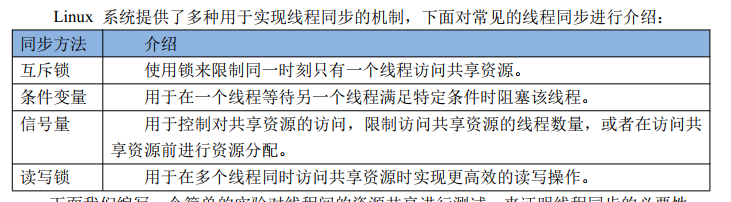


如果线程已经处在分离状态，再使用pthread\_detach（）函数将不会产生任何效果，函数返回值为0。如果调用该函数事失败，会返回一个非零值，表示错误代码。

# 9.7 线程同步

每一个服务进程的运行， 都包含若干进程（Thread） ， 线程是调度的基本单位， 进程则是资源拥有的基本单位。

线程的优势在于资源的共享性，这就出现了一个问题，多个线程的共享资源，对共享资源的并发访问是竞争。



# 9.8 互斥锁

互斥锁是一种常见的线程同步机制， 用于保护共享资源的访问， 使得同一时刻只有一个线程能够访问该资源

互斥锁的基本思想是在访问共享资源前获取锁， 访问结束后释放锁。 在获取锁的期间， 其他线程将被阻塞， 直到当前线程释放锁为止。 由于只有一个线程能够持有锁， 因此可以保证同一时刻只有一个线程访问共享资源， 从而避免了数据竞争和并发错误



