**1. 关于join**

join是三种同步线程的方式之一。另外两种分别是互斥锁（mutex）和条件变量（condition variable）。

**调用pthread\_join()将阻塞自己**，一直到要等待加入的线程运行结束。

可以用pthread\_join()获取线程的返回值。

一个线程对应一个pthread\_join()调用，对同一个线程进行多次pthread\_join()调用是逻辑错误。

join or detach

**线程分两种：一种可以join，另一种不可以。**该属性在创建线程的时候指定。

joinable线程可在创建后，用pthread\_detach()显式地分离。但分离后不可以再合并。该操作不可逆。 为了确保移植性，在创建线程时，最好显式指定其join或detach属性。似乎不是所有POSIX实现都是用joinable作默认。

**2. pthread\_detach**

创建一个线程默认的状态是joinable, 如果一个线程结束运行但没有被join,则它的状态类似于进程中的Zombie Process,即还有一部分资源没有被回收（退出状态码），所以创建线程者应该调用pthread\_join来等待线程运行结束，并可得到线程的退出代码，回收其资源（类似于wait,waitpid)

但是**调用pthread\_join(pthread\_id)后，如果该线程没有运行结束，调用者会被阻塞**，在**有些情况下我们并不希望如此**，比如在Web服务器中当主线程为每个新来的链接创建一个子线程进行处理的时候，主线程并不希望因为调用pthread\_join而阻塞（因为还要继续处理之后到来的链接），**这时可以在子线程中加入代码 pthread\_detach(pthread\_self())** 或者父线程调用 pthread\_detach(thread\_id)（**非阻塞，可立即返回**）

**这将该子线程的状态设置为detached，则该线程运行结束后会自动释放所有资源。**

**总结：**

如果进程中的某个线程执行了pthread\_detach(th)，则th线程将处于DETACHED状态，这使得th线程在结束运行时自行释放所占用的内存资源，同时也无法由pthread\_join()同步，pthread\_detach()执行之后，对th请求pthread\_join()将返回错误

一个可join的线程所占用的内存仅当有线程对其执行了pthread\_join()后才会释放，因此为了避免内存泄漏，所有线程的终止，要么已设为DETACHED，要么就需要使用pthread\_join()来回收

**注：我们程序在使用中，都设置为DETACHED**