pthread_cancel 线程取消以及锁的释放 - chengyang

my.oschina.net/u/178323/blog/32535

```
#include <pthread.h>

       int pthread_cancel(pthread_t
thread);
```

一个线程可以通过此机制向另外一个线程发送结束请求，值得一提的是，接收此请求的线程可以通过本线程的两个属性来决定是否取消以及时同步（延时）取消还是异步（立即）取消。

函数成功返回，并不代表那线程就结束了。

下面看那两个属性设置：

```
#include <pthread.h>

       int pthread_setcancelstate(int state, int
*oldstate);
       int pthread_setcanceltype(int type, int *oldtype);

       Compile and link with -pthread.
```

state和type的值及其意义如下：

PTHREAD_CANCEL_ENABLE：

表明此线程是可取消的，也就是说，在接收一个取消请求到来之后，它注定是要取消的，只不过它有两种取消方式，一种是同步取消，另外一种是异步取消，下面会详说的。

PTHREAD_CANCEL_DISABLE：

表明此线程是不可取消的，那么接收取消请求之后，它依然自我的执行。

PTHREAD_CANCEL_DEFERRED：

表明在线程是可取消的情况下，它是同步（延时）取消的。所谓同步取消的意思就是说，在收到一个取消请求之后，它会继续执行下去，直到找到下一个取消点进行退出。

那么什么是取消点呢？：

取消点是在程序在运行的时候检测是否收到取消请求，是否允许允许操作执行的点。下面的POSIX线程函数就是取消点：□□
□□ pthread_join()
□□ pthread_cond_wait()
□□ pthread_cond_timedwait()
□□ pthread_testcancel()
□□ sem_wait()
□□ sigwait()

还有很多，可以参考man 7 threads

```
#include <pthread.h>

       void pthread_testcancel(void);

       Compile and link with -
pthread.
```

此函数是创建一个取消点。

PTHREAD_CANCEL_ASYNCHRONOUS：

表明是异步取消方式，也就是说线程在收到取消请求之后，立即取消退出。

在默认的情况下，一个线程是可取消的并且是同步取消的。

这里，我们要考虑到一个线程退出后它后续处理的问题，比如说，如果一个线程正执行到一个锁内时，已经获得锁了，这时，它因为异常退出了，那么此时就是一个死锁的问题。这时我们可使用下面的两个函数：

```
#include <pthread.h>

       void pthread_cleanup_push(void (*routine)(void
*),
                                 void *arg);
       void pthread_cleanup_pop(int execute);
```

这是man文档上描述的，比别人中文翻译好多了：

These functions manipulate the calling thread's stack of thread-cancellation clean-up handlers.

A clean-up handler is a function that is automatically executed when a thread is canceled (or in

various other circumstances described below); it might, for example, unlock a mutex so that it

becomes available to other threads in the process.

就是说在一个线程结束的时候，会自动执行一个clean-up函数柄，这个函数柄里有一个stack（栈），我们之前就通过pthread_cleanup_push往这个栈里压入了一个函数，我们压入很多个，然后退出的时候，clean-up函数柄会自动的从这些栈里拿出函数进行执行。

如果此函数没有异常退出，那这些栈的函数怎么办呢？我们可以通过phread_cleanup_pop弹出这些栈里的函数，注意，此时的参数要为0，如果非0的话，弹出的同时也会执行这些函数的。

那之前讲到的死锁问题我们可以这样解决：

```
    pthread_cleanup_push(pthread_mutex_unlock, (void *)
&mutex);

    pthread_mutex_lock(&mutex);

    /* do some work */

    pthread_mutex_unlock(&mutex);

    pthread_cleanup_pop(0);
```