1线程终止方式

如果需要只终止某个线程而不终止整个线程，可以有三种方法：

A:从主线程函数return.这种方法对主线程不适合，从main函数return相当于调用exit.

B:一个线程可以调用pthread\_cancel终止同一进程中的另一个线程。

C:线程可以调用pthread\_exit终止自己

同一个进程的线程间，pthread\_cancel向另一个线程发终止信号。系统不会马上关闭被取消线程，只有在被取消线程下次系统调用时，才会真正结束线程。或调用pthread\_testcancel,让内核去检测是否需要取消当前线程。

2.线程属性

Linux下线程的属性是可以根据实际项目需要，进行设置，可以改变线程的默认属性，默认属性已经可以解决大多数开发时遇到的问题。如我们对程序的性能提出更高的要求，那么需要设置线程属性，比如可以通过设置线程栈的大小来降低内存的使用，增加最大线程个数。

typedef struct

{

int etachstate; //线程的分离状态

int schedpolicy; //线程调度策略

structsched\_param schedparam; //线程的调度参数

int inheritsched; //线程的继承性

int scope; //线程的作用域

size\_t guardsize; //线程栈末尾的警戒缓冲区大小

int stackaddr\_set; //线程的栈设置

void\* stackaddr; //线程栈的位置

size\_t stacksize; //线程栈的大小

}pthread\_attr\_t;

上面是一个简写的结构体，用来设置线程的信息

属性值不能直接设置，须使用相关函数进行操作，初始化的函数为pthread\_attr\_init，

这个函数必须在pthread\_create函数之前调用。之后须用pthread\_attr\_destroy函数来释

放资源。线程属性主要包括如下属性：作用域（scope）、栈尺寸（stack size）、栈地址

（stack address）、优先级（priority）、分离的状态（detached state）、调度策略和

参数（scheduling policy and parameters）。默认的属性为非绑定、非分离、缺省M的堆

栈、与父进程同样级别的优先级。

2 线程属性初始化

先初始化线程属性，再pthread\_create创建线程

A 依赖的头文件

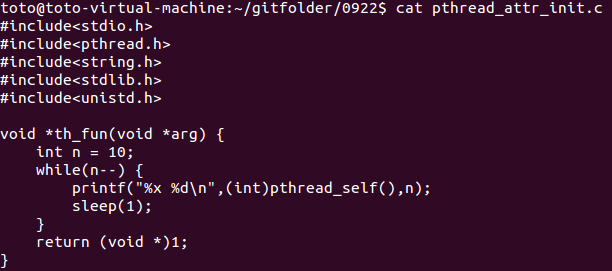
#include<pthread.h>

B 函数声明

int pthread\_attr\_init(pthread\_attr\_t \*attr); //初始化线程属性

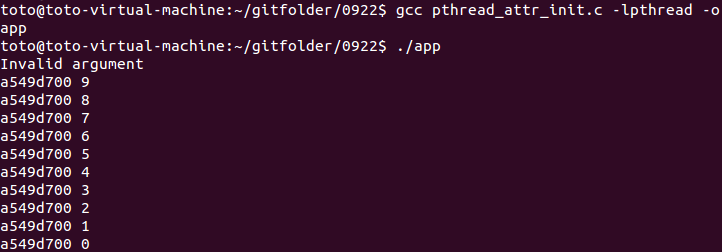
int pthread\_attr\_destroy(pthread\_attr\_t \*attr); //销毁线程属性所占用的资源

案例说明：





运行结果：



总结：之所以会出现Invalid argument是因为分离后又调用了pthread\_join

3 线程的分离状态（detached state）

A 线程的分离状态决定一个线程以什么样的方式来终止自己。

B 非分离状态：线程的默认属性是非分离状态，这种情况下，原有的线程等待创建的线程结束。只有当pthread\_join()函数返回时，创建的线程才算终止，才能释放自己占用的系统资源

C 分离状态：分离线程没有被其他的线程所等待，自己运行结束了，线程也就终止了，马上释放系统资源。应该根据自己的需要，选择适当的分离状态。

线程分离状态的函数：

#include <pthread.h>

int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate); //设置线程属性，分离or非分离

int pthread\_attr\_getdetachstate(pthread\_attr\_t \*attr, int \*detachstate); //获取程属性，分离or非分离

pthread\_attr\_t \*attr:被已初始化的线程属性

int \*detachstate:可选为PTHREAD\_CREATE\_DETACHED（分离线程）和PTHREAD \_CREATE\_JOINABLE（非分离线程）

注意：如果设置一个线程为分离线程，而这个线程运行又非常快，它很可能在pthread\_create函数返回之前就终止了，它终止以后就可能将线程号和系统资源移交给其他的线程使用，这样调用pthread\_create的线程就得到了错误的线程号。要避免这种情况可以采取一定的同步措施，最简单的方法之一是可以在被创建的线程里调用pthread\_cond\_timedwait函数，让这个线程等待一会儿，留出足够的时间让函数pthread\_create返回。设置一段等待时间，是在多线程编程里常用的方法。但是注意不要使用诸如wait()之类的函数，它们是使整个进程睡眠，并不能解决线程同步的问题。

4 线程的栈地址（stack address）

POSIX.1定义了两个常量\_POSIX\_THREAD\_ATTR\_STACKADDR 和\_POSIX\_THREAD\_ATTR\_STACKSIZE检测系统是否支持栈属性。也可以给sysconf函数传递\_SC\_THREAD\_ATTR\_STACKADDR或\_SC\_THREAD\_ATTR\_STACKSIZE来进行检测。

当进程栈地址空间不够用时，指定新建线程使用由malloc分配的空间作为自己的栈空

间。通过pthread\_attr\_setstackaddr和pthread\_attr\_getstackaddr两个函数分别设置和获

取线程的栈地址。传给pthread\_attr\_setstackaddr函数的地址是缓冲区的低地址（不一定

是栈的开始地址，栈可能从高地址往低地址增长）。

#include<pthread.h>

函数声明

int pthread\_attr\_setstackaddr(pthread\_attr\_t \*attr, void \*stackaddr);

int pthread\_attr\_getstackaddr(pthread\_attr\_t \*attr, void \*\*stackaddr);

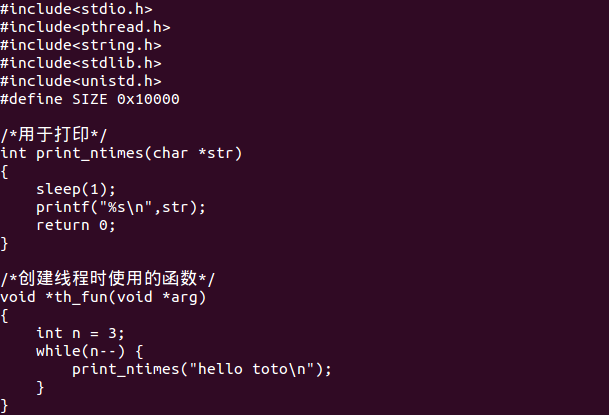
attr: 指向一个线程属性的指针

stackaddr: 返回获取的栈地址

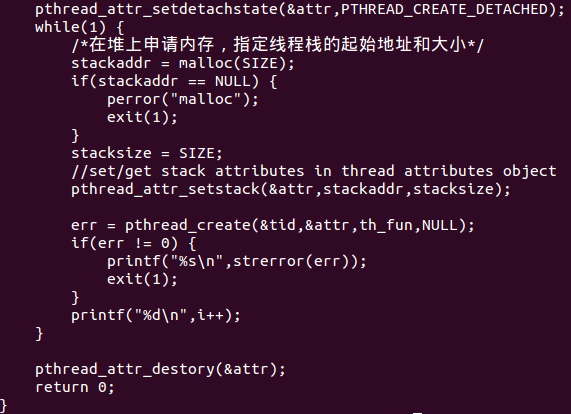
返回值：若是成功返回0,否则返回错误的编号

说明：函数已过时，一般用下面讲到的pthread\_attr\_getstack来代替

案例说明







运行结果：



5 NPTL

查看当前pthread库版本

getconf GNU\_LIBPTHREAD\_VERSION

NPTL实现机制（POSIX）,Native POSIX Thread Library

使用线程库时gcc指定-lpthread

6.注意

A 主线程退出，其它线程不退出，主线程应调用pthread\_exit

B 避免僵尸线程

join

pthread\_detach

pthread\_create指定分离属性

被join线程可能在join函数返回前就释放完自己的所有内存资源，所以不应当返回被回收线程栈中的值;

C malloc和mmap申请的内存可以被其它线程释放

D 如果线程终止时没有释放加锁的互斥量，则该互斥量不能再被使用

E 应避免在多线程模型中调用fork，除非马上exec,子进程中只有调用fork的线程存在，其他线程在子进程中均pthread\_exit

F 信号的复杂语义很难和很多线程共存，应避免在多线程引入信号机制。