1 pthread\_create()函数

创建线程

A:依赖的头文件

#include<pthread.h>

B:函数声明

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,

void \*(\*start\_routine) (void \*), void \*arg);

pthread\_t \*thread:传递一个pthread\_t变量地址进来，用于保存新线程的tid（线程ID）

const pthread\_attr\_t \*attr:线程属性设置，如使用默认属性，则传NULL

void \*(\*start\_routine) (void \*):函数指针，指向新线程应该加载执行的函数模块

void \*arg:指定线程将要加载调用的那个函数的参数

返回值:成功返回0，失败返回错误号。以前学过的系统函数都是成功返回0，失败返回-1，而错误号保存在全局变

量errno中，而pthread库的函数都是通过返回值返回错误号，虽然每个线程也都有一个errno，但这是为了兼容其

它函数接口而提供的，pthread库本身并不使用它，通过返回值返回错误码更加清晰。

在gcc编译链接的时候，后面要加上-lpthread

关于pthread\_t

typedef unsigned long int pthread\_t;(32位平台下)

在一个线程中调用pthread\_create()创建新的线程后，当前线程从pthread\_create()

返回继续往下执行，而新的线程所执行的代码由我们传给pthread\_create的函数指针

start\_routine决定。start\_routine函数接收一个参数，是通过pthread\_create的arg参

数传递给它的，该参数的类型为void \*，这个指针按什么类型解释由调用者自己定

义。start\_routine的返回值类型也是void \*，这个指针的含义同样由调用者自己定义。start\_routine返回时，这个线程就退出了，其它线程可以调用pthread\_join得到start\_routine的返回值，类似于父进程调用wait(2)得到子进程的退出状态，稍后详细介绍pthread\_join。

pthread\_create成功返回后，新创建的线程的id被填写到thread参数所指向的内存单

元。我们知道进程id的类型是pid\_t，每个进程的id在整个系统中是唯一的，调用getpid(2)

可以获得当前进程的id，是一个正整数值。线程id的类型是thread\_t，它只在当前进程中

保证是唯一的，在不同的系统中thread\_t这个类型有不同的实现，它可能是一个整数值，

也可能是一个结构体，也可能是一个地址，所以不能简单地当成整数用printf打印，调用

pthread\_self(3)可以获得当前线程的id。

attr参数表示线程属性，本节不深入讨论线程属性，所有代码例子都传NULL给attr参

数，表示线程属性取缺省值，感兴趣的读者可以参考[APUE2e]。

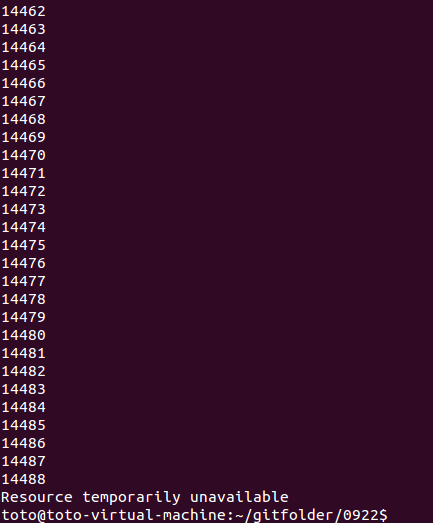
C案例说明（程序编写，测试最大能够创建的线程数）



执行下面命令：



运行./app,结果如下：



2 pthread\_self()函数

获取调用线程tid

A 依赖的头文件

#include<pthread.h>

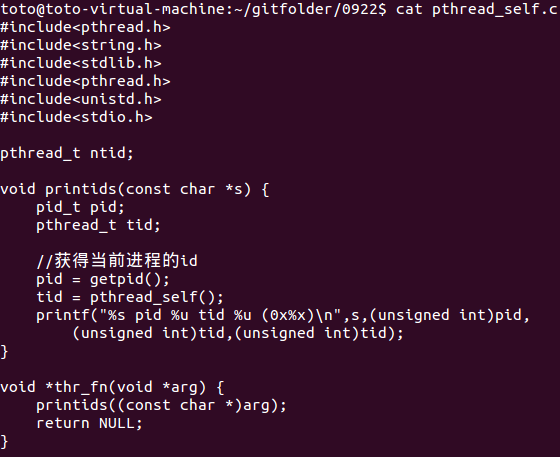
B 函数声明

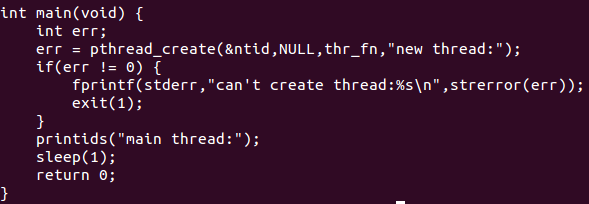
pthread\_t pthread\_self(void);

C 函数说明

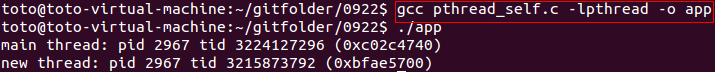
This function always succeeds, returning the calling thread's ID.

D 案例说明1：





运行结果：

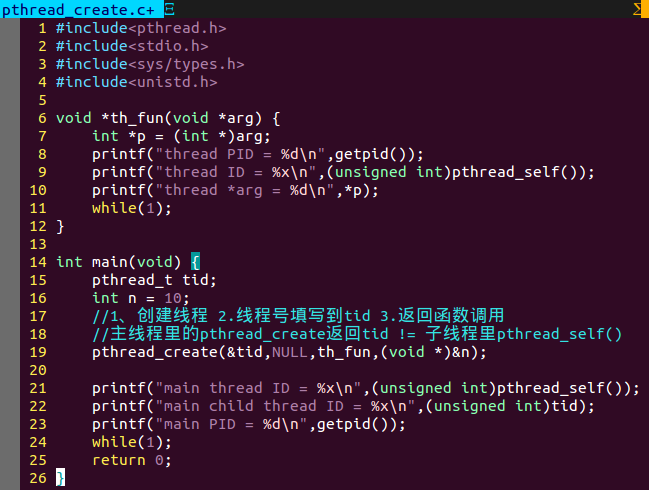


说明：

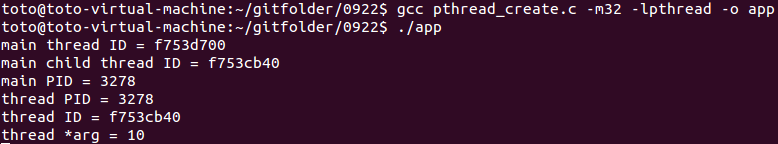
由于pthread\_create的错误码不保存在errno中，因此不能直接用perror(3)打印错误信

息，可以先用strerror(3)把错误码转换成错误信息再打印。

E 案例说明2：



运行结果：



3 pthread\_exit()函数

说明：

如果任意一个线程调用了exit或\_exit，则整个进程的所有线程都终止，由于从main函数return也相当于调用exit,为了防止新创建的线程还没有得到执行就终止，我们在main函数return之前延迟1秒，这只是权宜之计，即使主线程等待1秒，内核也不一定会调度新创建的线程执行。所以这时候要用到pthread\_exit()

pthread\_exit()函数的特点如下：

A: 调用线程退出函数，注意和exit函数的区别，任何线程里exit导致进程退出，其他线程

未工作结束，主控线程退出时不能return或exit。

B：需要注意，pthread\_exit或者return返回的指针所指向的内存单元必须是全局的或者是用malloc分配的，不能在线程函数的栈上分配，因为当其它线程得到这个返回指针时线程函数已经退出了。

C依赖的头文件

#include<pthread.h>

D 函数声明

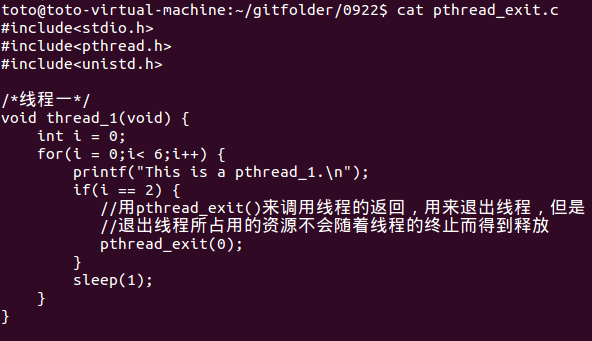
void pthread\_exit(void \*retval);

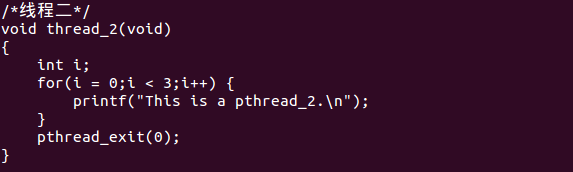
E 功能：使用函数pthread\_exit退出线程，这是线程的主动行为；由于一个进程中的多个线程是共享数据段的，因此通常在线程退出之后，退出线程所占用的资源并不会随着线程的终止而得到释放，但是可以用pthread\_join()函数来同步并释放资源。

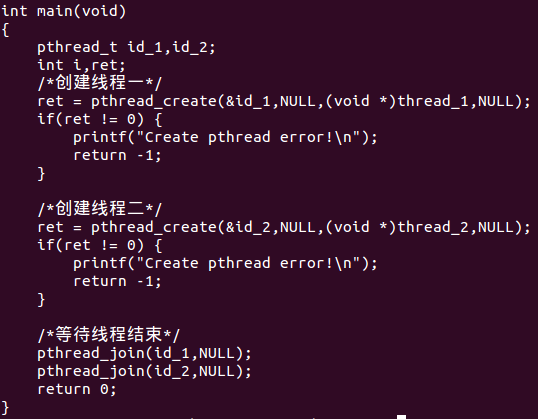
F 说明

retval:pthread\_exit()调用线程的返回值，可由其他函数如pthread\_join来检索获取。

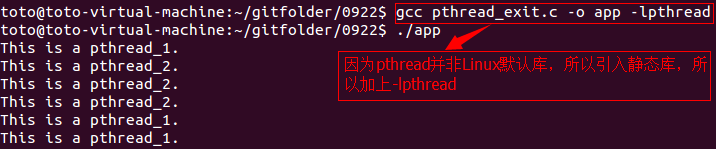
G 案例说明：







运行结果：



此外：pthread\_exit()函数括号中的值是用于传出的，类似exit(0)和exit(1)。这个值供pthread\_join获取，用以判断是正常退出还是异常退出。

4 pthread\_join()函数

A 依赖的头文件

#include<pthread.h>

B 函数声明

int pthread\_join(pthread\_t thread,void \*\*retval);

pthread\_t thread:回收线程的tid

void \*\*retval:接收退出线程传递出的返回值

返回值：成功返回0，失败返回错误号

调用该函数的线程将挂起等待，直到id为thread的线程终止。thread线程以不同的方式终止，通过pthread\_join得到的终止状态是不同的，总结如下：

1. 如果thread线程通过return返回，value\_ptr所指向的单元里存放的是thread线程函数的返回值。
2. 如果thread线程被别的线程调用pthread\_cancel异常终止掉，value\_ptr所指向的单元存放的是常熟PTHREAD\_CANCELED.
3. 如果thread线程是自己调用pthread\_exit终止的，value\_ptr所指向的单元存放的是传给pthread\_exit的参数。
4. 如果对thread线程的终止状态不感兴趣，可以传NULL给value\_ptr参数。

C:函数描述：

The pthread\_join() function waits for the thread specified by thread to

terminate. If that thread has already terminated, then pthread\_join()

returns immediately. The thread specified by thread must be joinable.

If retval is not NULL, then pthread\_join() copies the exit status of

the target thread (i.e., the value that the target thread supplied to

pthread\_exit(3)) into the location pointed to by \*retval. If the tar‐

get thread was canceled, then PTHREAD\_CANCELED is placed in \*retval.

If multiple threads simultaneously try to join with the same thread,

the results are undefined. If the thread calling pthread\_join() is

canceled, then the target thread will remain joinable (i.e., it will

not be detached).

5 pthread\_cancel

在进程内某个线程可以取消另外一个线程

A 依赖的头文件

#include<pthread.h>

B 函数声明

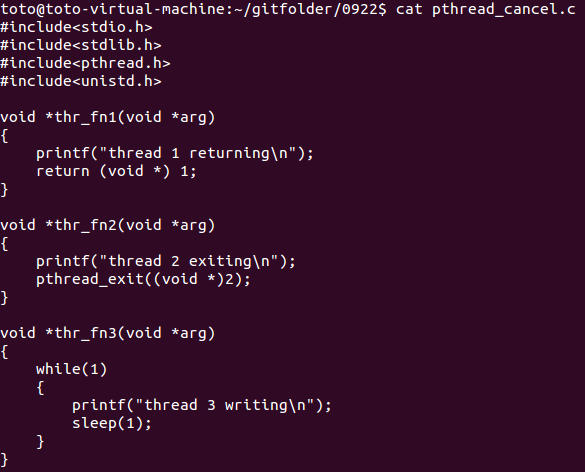
int pthread\_cancel(pthread\_t thread);

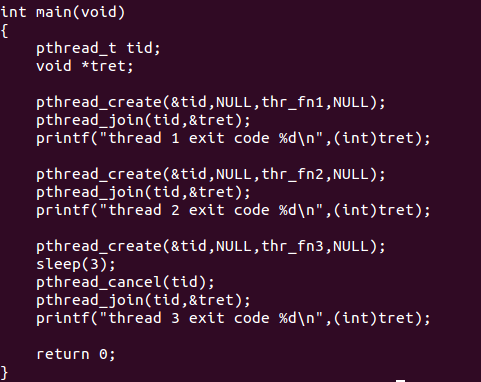
C 函数说明：

被取消的线程，退出值，定义在Linux的pthread库中常数PTHREAD\_CANCELED的值是-1.可以在头文件pthread.h中找到它的定义：

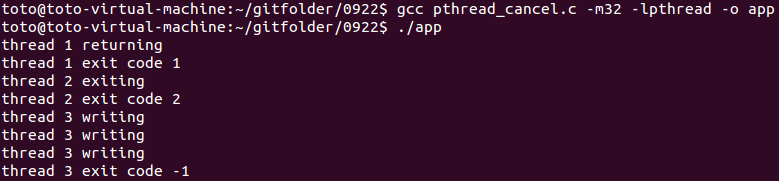
#define PTHREAD\_CANCELED ((void \*) -1)

D 案例说明：





运行结果：



6 pthread\_detach()函数

A 依赖的头文件

#include<pthread.h>

B 函数声明

int pthread\_detach(pthread\_t tid);

pthread\_t tid:分离线程tid

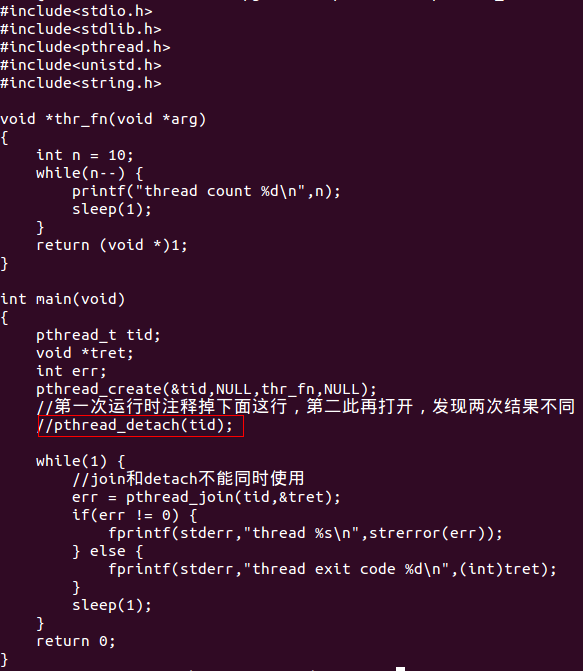
返回值：成功返回0，失败返回错误号

C 函数说明：

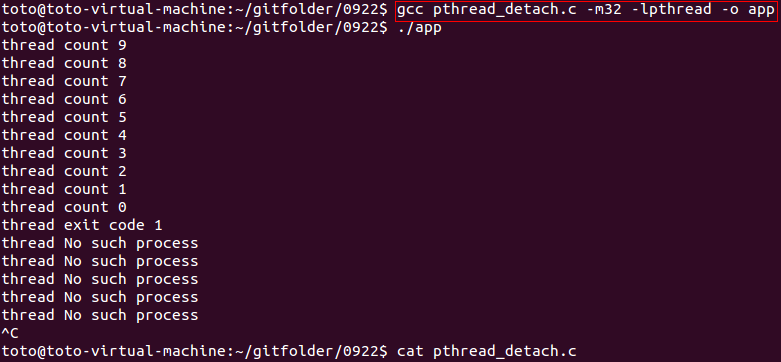
一般情况下，线程终止后，其终止状态一直保留到其它线程调用pthread\_join获取它的

状态为止。但是线程也可以被置为detach状态，这样的线程一旦终止就立刻回收它占用的所有资源，而不保留终止状态。不能对一个已经处于detach状态的线程调用pthread\_join，这样的调用将返回EINVAL。对一个尚未detach的线程调用pthread\_join或pthread\_detach都可以把该线程置为detach状态，也就是说，不能对同一线程调用两次pthread\_join，或者如果已经对一个线程调用了pthread\_detach就不能再调用pthread\_join了。

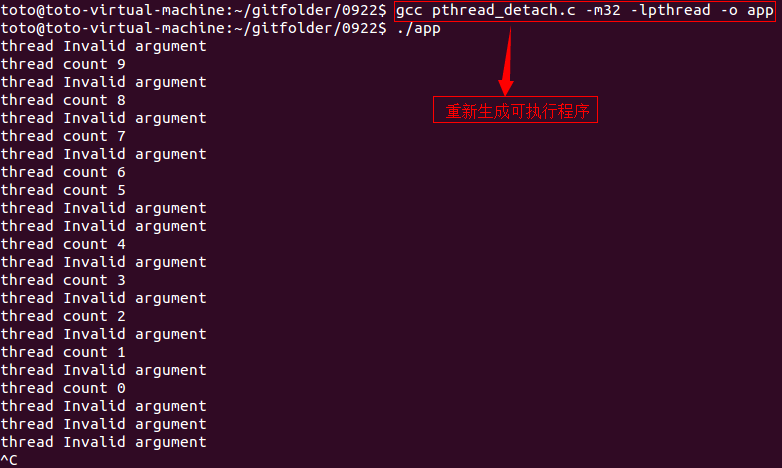
E 案例说明



运行结果：



当把代码不部分的红框中的代码取消注释，在运行时，运行结果如下：



说明：join和detach不能同时使用

7 pthread\_equal()函数

比较两个线程是否相等

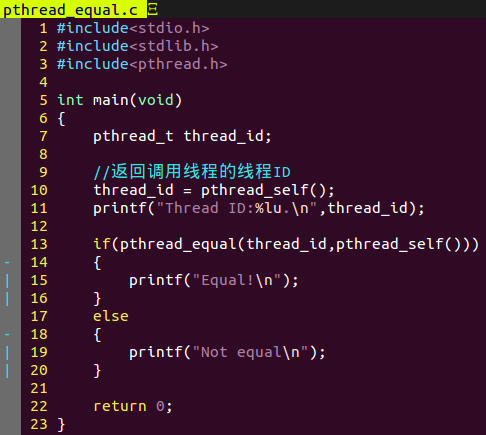
A 依赖的头文件

#include<pthread.h>

B 函数声明

int pthread\_equal(pthread\_t t1,pthread\_t t2);

C 案例说明



运行结果：

