线程的取消：

意义：随时杀掉一个线程

int pthread\_cancel(pthread\_t thread);

注意：线程的取消要有取消点才可以，不是说取消就取消，线程的取消点主要是阻塞的系统调用

运行段错误调试：

可以使用gdb调试

使用gdb 运行代码，gdb ./youapp

(gdb) run

等待出现Thread 1 "pcancel" received signal SIGSEGV, Segmentation fault.

输入命令bt（打印调用栈）

(gdb) bt

#0 0x00007ffff783ecd0 in vfprintf () from /lib/x86\_64-linux-gnu/libc.so.6

#1 0x00007ffff78458a9 in printf () from /lib/x86\_64-linux-gnu/libc.so.6

#2 0x00000000004007f9 in main () at pcancel.c:21

确定段错误位置是pcancel.c 21行

如果没有取消点，手动设置一个

void pthread\_testcancel(void);

设置取消使能或禁止

int pthread\_setcancelstate(int state, int \*oldstate);

PTHREAD\_CANCEL\_ENABLE

PTHREAD\_CANCEL\_DISABLE

设置取消类型

int pthread\_setcanceltype(int type, int \*oldtype);

PTHREAD\_CANCEL\_DEFERRED 等到取消点才取消

PTHREAD\_CANCEL\_ASYNCHRONOUS 目标线程会立即取消

线程的清理

必要性： 当线程非正常终止，需要清理一些资源。

void pthread\_cleanup\_push(void (\*routine) (void \*), void \*arg)

void pthread\_cleanup\_pop(int execute)

routine 函数被执行的条件：

1. 被pthread\_cancel取消掉。
2. 执行pthread\_exit
3. 非0参数执行pthread\_cleanup\_pop()

注意：

1. 必须成对使用，即使pthread\_cleanup\_pop不会被执行到也必须写上，否则编译错误。2.pthread\_cleanup\_pop()被执行且参数为0，pthread\_cleanup\_push回调函数routine不会被执行.

3 pthread\_cleanup\_push 和pthread\_cleanup\_pop可以写多对，routine执行顺序正好相反

1. 线程内的return 可以结束线程，也可以给pthread\_join返回值，但不能触发pthread\_cleanup\_push里面的回调函数，所以我们结束线程尽量使用pthread\_exit退出线程。