网络编程中close函数和shutdown函数的区别

在Linux C网络编程中，一共有两种方法来关闭一个已经连接好的网络通信，它们就是close函数和shutdown函数，它们的函数原型分别为：

```
#include<unistd.h>
int close(int sockfd)
//返回：0——成功， 1——失败

#include<sys/socket.h>
int shutdown(int sockfd, int howto)
//返回：0——成功， 1——失败
```

close函数和shutdown函数的第一个参数都是代表的是一个文件描述符。我们知道，在Linux操作系统中，一切东西都是当作文件来对待，所有的东西，诸如设备、内存都模拟成文件；当然，网络之间的通信也不例外。每一个通信对话都有一个文件描述符对应着，你对它们之间的操作就像在操作本地的文件一样。在shutdown函数当中，还有一个参数howto，它有一下三种值

1. SHUT_RD：关闭读这一半，此时用户不能再从这个套接字读数据，这个套接口接收到的数据都会被丢弃，对等方不知道这个过程。
2. SHUT_WR：相应地关闭写这一半，此时用户不能再向套接字中写数据，内核会把缓存中的数据发送出去，接着不会再发送数据，对等端将会知道这一点。当对等端试图去读的时候，可能会发生错误。
3. SHUT_RDWR：关闭读与写两半，此时用户不能从套接字中读或写。它相当于再次调用shutdown函数，并且一次指定SHUT_RD，一次指定SHUT_WR。

SHUT_**在函数库里面都是由宏定义的；由于shutdown提供了第二个，它可以精确的控制一个套接字描述符的关闭，这对close函数来说是无法实现的。在多线程环境中，一个描述符可能是被好几个线程复制了，它们与一个文件关联，并且内核维护一个文件引用计数，只有在引用计数为零的情况下close才可以关闭掉这个文件描述符。
使用close函数有两个限制，却可以使用shutdown来避免：

1. close函数把描述符的引用计数减一，仅仅在该计数变为0的时候，才真正的关闭套接字，而使用shutdown函数可以不管引用计数就激发了TCP的正常连接终止序列；
2. close函数终止读和写两个方向的数据传输。既然TCP连接是全双工的，有时候我们需要告知对端我们已经完成了数据发送，我们仅仅需要关闭数据发送的一个通道，但是我们还是可以接收到对端发送过来的数据，这种控制只有利用shutdown函数才能实现。

本文链接: 【】（）