**linux 下写socket遭遇broken pipe（SIGPIPE C++）**

2011-05-26 16:57:00

标签：[linux](http://blog.51cto.com/tag-linux.html) [socket](http://blog.51cto.com/tag-socket.html) [broken pipe](http://blog.51cto.com/tag-broken%20pipe.html) [SIGPIPE](http://blog.51cto.com/tag-SIGPIPE.html) [C++](http://blog.51cto.com/tag-C++.html)

原代码片段如下，程序在第08行报错，但是try，，，catch无法抓到错误，经过debug分析是由于收到broken pipe的信号。看来程序此时被终止了；那么我期望不被终止，该怎么做了。

01 int sendLen = 0;  
02 int totalLen = 0;  
03 int packSize = pack.size();  
04 while(packSize != totalLen)  
05 {  
06 try  
07 {  
08 sendLen = write(fd, const\_cast<char\*>(pack.c\_str())+totalLen, packSize-totalLen);  
09 totalLen += sendLen;  
10 if(sendLen <= 0)  
11 {  
12 totalLen == 0;  
13 fprintf(stderr,"write fd err . fd == %d - %m/n",fd);  
14 return false;  
15 }  
16 }  
17 catch (std::exception& e)  
18 {  
19 printf("errrno is:%d",errno);  
20 std::cout << "Exception: " << e.what();  
21 }  
22 }  
23 return true;

在徐明刚的指导下，找到如下文章

网摘

linux下写socket的程序的时候，如果尝试send到一个disconnected socket上，就会让底层抛出一个SIGPIPE信号。client端通过 pipe 发送信息到server端后，就关闭client端, 这时server端,返回信息给 client 端时就产生Broken pipe 信号了。 对于产生信号，我们可以在产生信号前利用方法 signal(int signum, sighandler\_t handler) 设置信号的处理。如果没有调用此方法，系统就会调用默认处理方法：中止程序，显示提示信息(就是我们经常遇到的问题)。我们可以调用系统的处理方法，也可以自定义处理方法。 对一个已经收到FIN包的socket调用read方法, 如果接收缓冲已空, 则返回0,这就是常说的表示连接关闭. 但第一次对其调用write方法时, 如果发送缓冲没问题,会返回正确写入(发送). 但发送的报文会导致对端发送RST报文,因为对端的socket已经调用了close, 完全关闭, 既不发送, 也不接收数据. 所以,第二次调用write方法(假设在收到RST之后), 会生成SIGPIPE信号, 导致进程退出.为了避免进程退出, 可以捕获SIGPIPE信号, 或者忽略它,给它设置SIG\_IGN信号处理函数:signal(SIGPIPE, SIG\_IGN);这样, 第二次调用write方法时, 会返回-1, 同时errno置为SIGPIPE.程序便能知道对端已经关闭.

为此代码变更如下；插入10行

01 bool CWWSimulator::SendPack(string& pack, uint32\_t fd)  
02 {   
03 int sendLen = 0;  
04 int totalLen = 0;  
05 int packSize = pack.size();  
06 while(packSize != totalLen)  
07 {  
08 try  
09 {  
10 signal(SIGPIPE, SIG\_IGN);  
11 sendLen = write(fd, const\_cast<char\*>(pack.c\_str())+totalLen, packSize-totalLen);  
12 totalLen += sendLen;  
13 if(sendLen <= 0)  
14 {  
15 totalLen == 0;  
16 fprintf(stderr,"write fd err . fd == %d - %m/n",fd);  
17 return false;  
18 }  
19 }  
20 catch (std::exception& e)  
21 {  
22 printf("errrno is:%d",errno);  
23 std::cout << "Exception: " << e.what();  
24 }  
25 }   
26 return true;  
27 }