<https://blog.csdn.net/kiterunner/article/details/44888703>

使用ssh2实现shell自动化测试，实际工作中遇到非常多。各种语言都有相应的库可以使用。比如，c/c++语言可以使用libssh2；python可以使用paramkio库等。但这些库网上的帮助，都不是很全，都比较浅显。shell自动化，最基本的三个需求，一个是适合多重类型的操作系统；二是要能够支持交互式shell，比如使用sudo执行时，需要输入密码；三是读数据时要非阻塞的。

paramkio好像不支持交互式（shell命令不需要再根据输出输入不同的参数，实际上，这种情况遇到非常多），而且paramkio在windows上也很不好用，要实现非阻塞，找了几天都没能找到好的办法。

libssh2表面上不支持交互式，官方的例子也没有针对交互式shell进行举例，给库的使用者带来很大的困扰，比较难入门。实际上libssh2库是支持交互式shell的，而且支持多种OS，实现起来也相当的简单，只是过程摸索过于痛苦。关于libssh2的编译，也比较简单，这里就不列举了。libssh2是c语言的，样例写起来很繁琐，因此我用c++进行封装，学习起来比较直观。

实际libssh2两个官方的样例scp.c和scp\_echo.c已经提供了交互式shell的实现方式，只是scp\_echo.c有关ssh2登陆认证方式，没有像scp.c使用了ssh2的键盘交互式方式，一般ssh2服务器默认提供这个认证方式。scp\_echo.c不容易调试通过。

首先，先看一下实现Ssh2类的使用代码：

**[cpp]** [view plain](https://blog.csdn.net/kiterunner/article/details/44888703) [copy](https://blog.csdn.net/kiterunner/article/details/44888703)

1. #include <iostream>
2. #include "ssh2.h"
4. **int** main(**int** argc, **const** **char** \* argv[])
5. {
6. **using** **namespace** std;
7. **using** **namespace** fish;
9. Ssh2 ssh("127.0.0.1");
10. ssh.Connect("test","xxxxxx");
11. Channel\* channel = ssh.CreateChannel();
12. channel->Write("cd /;pwd");
13. cout<<channel->Read()<<endl;
14. channel->Write("ssh 127.0.0.1");
15. cout<<channel->Read(":")<<endl;
16. channel->Write("xxxxxx");
17. cout<<channel->Read()<<endl;
18. channel->Write("pwd");
19. cout<<channel->Read()<<endl;
20. **delete** channel;
21. **return** 0;
22. }

读写都是非阻塞的，这个实际使用非常方便。代码就不详细讲解了。

Ssh2类的实现代码如下:

**[cpp]** [view plain](https://blog.csdn.net/kiterunner/article/details/44888703) [copy](https://blog.csdn.net/kiterunner/article/details/44888703)

1. **class** Ssh2
2. {
3. **public**:
4. Ssh2(**const** string &srvIp, **int** srvPort=22 );
5. ~Ssh2(**void**);
7. **bool** Connect( **const** string &userName, **const** string &userPwd);
8. **bool** Disconnect(**void**);
10. Channel\* CreateChannel(**const** string &ptyType = "vanilla");
12. **public**:
13. **static** **void** S\_KbdCallback(**const** **char**\*, **int**, **const** **char**\*, **int**, **int**, **const** LIBSSH2\_USERAUTH\_KBDINT\_PROMPT\*, LIBSSH2\_USERAUTH\_KBDINT\_RESPONSE\*, **void** \*\*a );
14. **static** string s\_password;
16. **private**:
17. string m\_srvIp;
18. **int**    m\_srvPort;
19. string m\_userName;
20. string m\_password;
21. **int**    m\_sock;
22. LIBSSH2\_SESSION \*m\_session;
23. };

由于ssh2服务器要通过键盘交互式是通过回调函数实现认证的，所以使用了两个静态成员，一个是成员函数，一个是成员变量（当然，这种实现方式比较丑，暂时没有找到好的方法）。这里代码，没有考虑到多线程安全，因为实际需求不是很严格。当然，要支持多线程下ssh2登陆认证，也很容易实现。

**[cpp]** [view plain](https://blog.csdn.net/kiterunner/article/details/44888703) [copy](https://blog.csdn.net/kiterunner/article/details/44888703)

1. string Ssh2::s\_password;
3. **void** Ssh2::S\_KbdCallback(**const** **char** \*name, **int** name\_len,
4. **const** **char** \*instruction, **int** instruction\_len,
5. **int** num\_prompts,
6. **const** LIBSSH2\_USERAUTH\_KBDINT\_PROMPT \*prompts,
7. LIBSSH2\_USERAUTH\_KBDINT\_RESPONSE \*responses,
8. **void** \*\*abstract)
9. {
10. (**void**)name;
11. (**void**)name\_len;
12. (**void**)instruction;
13. (**void**)instruction\_len;
14. **if** (num\_prompts == 1)
15. {
16. responses[0].text   = strdup(s\_password.c\_str());
17. responses[0].length = (**int**)s\_password.size();
18. }
19. (**void**)prompts;
20. (**void**)abstract;
21. }
23. Ssh2::Ssh2(**const** string &srvIp, **int** srvPort)
24. :m\_srvIp(srvIp),m\_srvPort(srvPort)
25. {
26. m\_sock = -1;
27. m\_session = NULL;
28. libssh2\_init(0);
29. }
31. Ssh2::~Ssh2(**void**)
32. {
33. Disconnect();
34. libssh2\_exit();
35. }
37. **bool** Ssh2::Connect(**const** string &userName, **const** string &userPwd)
38. {
39. m\_sock = socket(AF\_INET, SOCK\_STREAM, 0);
41. sockaddr\_in sin;
42. sin.sin\_family = AF\_INET;
43. sin.sin\_port = htons(22);
44. sin.sin\_addr.s\_addr = inet\_addr(m\_srvIp.c\_str());
45. **if** ( connect( m\_sock, (sockaddr\*)(&sin), **sizeof**(sockaddr\_in) ) != 0 )
46. {
47. Disconnect();
48. **return** **false**;
49. }
51. m\_session = libssh2\_session\_init();
52. **if** ( libssh2\_session\_handshake(m\_session, m\_sock) )
53. {
54. Disconnect();
55. **return** **false**;
56. }
58. **int** auth\_pw = 0;
59. string fingerprint = libssh2\_hostkey\_hash(m\_session, LIBSSH2\_HOSTKEY\_HASH\_SHA1);
60. string userauthlist = libssh2\_userauth\_list( m\_session, userName.c\_str(), (**int**)userName.size() );
61. **if** ( strstr( userauthlist.c\_str(), "password") != NULL )
62. {
63. auth\_pw |= 1;
64. }
65. **if** ( strstr( userauthlist.c\_str(), "keyboard-interactive") != NULL )
66. {
67. auth\_pw |= 2;
68. }
69. **if** ( strstr(userauthlist.c\_str(), "publickey") != NULL)
70. {
71. auth\_pw |= 4;
72. }
74. **if** (auth\_pw & 1)
75. {
76. /\* We could authenticate via password \*/
77. **if** ( libssh2\_userauth\_password(m\_session, userName.c\_str(), userPwd.c\_str() ) )
78. {
79. Disconnect();
80. **return** **false**;
81. }
82. }
83. **else** **if** (auth\_pw & 2)
84. {
85. /\* Or via keyboard-interactive \*/
86. s\_password = userPwd;
87. **if** (libssh2\_userauth\_keyboard\_interactive(m\_session, userName.c\_str(), &S\_KbdCallback) )
88. {
89. Disconnect();
90. **return** **false**;
91. }
92. }
93. **else**
94. {
95. Disconnect();
96. **return** **false**;
97. }

100. **return** **true**;
101. }
103. **bool** Ssh2::Disconnect(**void**)
104. {
105. **if**( m\_session )
106. {
107. libssh2\_session\_disconnect(m\_session, "Bye bye, Thank you");
108. libssh2\_session\_free(m\_session);
109. m\_session = NULL;
110. }
111. **if**( m\_sock != -1 )
112. {
113. #ifdef WIN32
114. closesocket(m\_sock);
115. #else
116. close(m\_sock);
117. #endif
118. m\_sock = -1;
119. }
120. **return** **true**;
121. }
123. Channel\* Ssh2::CreateChannel(**const** string &ptyType)
124. {
125. **if**( NULL == m\_session )
126. {
127. **return** NULL;
128. }
130. LIBSSH2\_CHANNEL \*channel = NULL;
131. /\* Request a shell \*/
132. **if** ( !(channel = libssh2\_channel\_open\_session(m\_session)) )
133. {
134. **return** NULL;
135. }
137. /\* Request a terminal with 'vanilla' terminal emulation
138. \* See /etc/termcap for more options
139. \*/
140. **if** ( libssh2\_channel\_request\_pty(channel, ptyType.c\_str() ) )
141. {
142. libssh2\_channel\_free(channel);
143. **return** NULL;
144. }
146. /\* Open a SHELL on that pty \*/
147. **if** ( libssh2\_channel\_shell(channel) )
148. {
150. libssh2\_channel\_free(channel);
151. **return** NULL;
152. }
154. Channel \*ret = **new** Channel(channel);
155. cout<<ret->Read()<<endl;
156. **return** ret;
157. }

这里要关注两个函数调用，一个是libssh2\_channel\_request\_pty指定channel类型，如果使用xterm，就可以使用彩色显示。libssh2\_channel\_shell用来指定是交互式shell。

下面是Channel类的定义：

**[cpp]** [view plain](https://blog.csdn.net/kiterunner/article/details/44888703) [copy](https://blog.csdn.net/kiterunner/article/details/44888703)

1. **const** **int** CHANNEL\_READ\_TIMEOUT = 3000;
3. **class** Channel
4. {
5. **public**:
6. Channel(LIBSSH2\_CHANNEL \*channel);
7. ~Channel(**void**);
9. string Read( **const** string &strend = "$", **int** timeout = CHANNEL\_READ\_TIMEOUT );
10. **bool**   Write(**const** string &data);
11. **private**:
12. Channel(**const** Channel&);
13. Channel operator=(**const** Channel&);
14. **private**:
15. LIBSSH2\_CHANNEL \*m\_channel;
16. };

Channel类的实现：

**[cpp]** [view plain](https://blog.csdn.net/kiterunner/article/details/44888703) [copy](https://blog.csdn.net/kiterunner/article/details/44888703)

1. Channel::Channel( LIBSSH2\_CHANNEL\*channel)
2. :m\_channel(channel)
3. {
5. }
7. Channel::~Channel(**void**)
8. {
9. **if** (m\_channel)
10. {
11. libssh2\_channel\_free(m\_channel);
12. m\_channel = NULL;
13. }
14. }
16. string Channel::Read( **const** string &strend, **int** timeout )
17. {
18. string data;
19. **if**( NULL == m\_channel )
20. {
21. **return** data;
22. }
24. LIBSSH2\_POLLFD \*fds = **new** LIBSSH2\_POLLFD;
25. fds->type = LIBSSH2\_POLLFD\_CHANNEL;
26. fds->fd.channel = m\_channel;
27. fds->events = LIBSSH2\_POLLFD\_POLLIN | LIBSSH2\_POLLFD\_POLLOUT;
29. **if**( timeout % 50 )
30. {
31. timeout += timeout %50;
32. }
33. **while**(timeout>0)
34. {
35. **int** rc = (libssh2\_poll(fds, 1, 10));
36. **if** (rc < 1)
37. {
38. timeout -= 50;
39. usleep(50\*1000);
40. **continue**;
41. }
43. **if** ( fds->revents & LIBSSH2\_POLLFD\_POLLIN )
44. {
45. **char** buffer[64\*1024] = "";
46. **size\_t** n = libssh2\_channel\_read( m\_channel, buffer, **sizeof**(buffer) );
47. **if** ( n == LIBSSH2\_ERROR\_EAGAIN )
48. {
49. //fprintf(stderr, "will read again\n");
50. }
51. **else** **if** (n <= 0)
52. {
53. **return** data;
54. }
55. **else**
56. {
57. data += string(buffer);
58. **if**( "" == strend )
59. {
60. **return** data;
61. }
62. **size\_t** pos = data.rfind(strend);
63. **if**( pos != string::npos && data.substr(pos, strend.size()) == strend  )
64. {
65. **return** data;
66. }
67. }
68. }
69. timeout -= 50;
70. usleep(50\*1000);
71. }
73. cout<<"read data timeout"<<endl;
74. **return** data;
75. }
77. **bool** Channel::Write(**const** string &data)
78. {
79. **if**( NULL == m\_channel )
80. {
81. **return** **false**;
82. }
84. string send\_data = data + "\n";
85. **return** libssh2\_channel\_write\_ex( m\_channel, 0, send\_data.c\_str(), send\_data.size() ) == data.size();
86. //return true;
87. }

这里主要关注libssh2\_poll函数，主要是检测通道是否有数据可以读。read函数简单做了封装，增加期望结果，有超时。Ssh2是支持多个Channel读写的，可以同时打个多个Channel，分别执行不同的操作，对于ssh2有登陆数限制时，还是很相当有用的。目前Ssh2客户端基本可用了。大家不妨试试。有关libssh2库再简单的例子，估计网上还没有吧，当然，有关libssh2安装，网上是有一大把。

用到的头文件包含如下：

**[cpp]** [view plain](https://blog.csdn.net/kiterunner/article/details/44888703) [copy](https://blog.csdn.net/kiterunner/article/details/44888703)

1. #ifdef WIN32
2. #include <windows.h>
3. #include <winsock2.h>
4. #else
5. #include <sys/socket.h>
6. #include <netinet/in.h>
7. #include <unistd.h>
8. #include <arpa/inet.h>
10. #endif
12. #include <sys/types.h>
13. #include <fcntl.h>
14. #include <errno.h>
15. #include <stdio.h>
16. #include <ctype.h>
18. #include <libssh2.h>
19. #include <libssh2\_sftp.h>

c++比较复杂，也想在python中使用libssh2的便利性，扩展python语言。实现起来，当然要复杂一些。但复杂度也很低，只需要使用boost::python类，扩展python就行了。windows和linux下，都不复杂。后续有时间再整理。

备注：

1、执行过长命令时，结果返回时前面有一段未知的输出，比如ls 11111111111222222222222222223333333333333334444444444444444输出时，前面好像有问题，不知是不是libssh2的bug。有时间去咨询。