Subject: Re: SSH package for U++
Posted by Oblivion on Sun, 12 Aug 2018 10:51:11 GMT
View Forum Message <> Reply to Message

Hello Mirek,

I've commited the first party of updates.

A summary:

- Ssh, SshSession, and SFtp classes are refactored.

- Non-blocking mode and multithreaded methods are removed.

- The package now uses a "pseudo-blocking" technique, similar to TcpSocket's.

- Accordingly, the Ssh::Cmd and Ssh::ComplexCmd methods are removed in favor of Ssh::Run and Ssh::Do() commands.
  Run() command is the new command execution encgine. It'll hopefully take care of the possible thread safety problems.
  Do() method is protected by a single static mutex, and should be thread-safe. Multithreaded execution should be possible.

- Abort mechanism refactored.

- Sftp::Read and SFtp::Write methods are further simplified.

- SFtp::WhenProgress method is removed in favour of GetDone() method (as the esisting WhenProgress method is now pretty useless).

- SshChannel and its derivatives are currently disabled. They will be re-added gradually.

I believe the new code can be easily maintained and changed in the future. (if we plan to take advantage of coroutines, etc.)

As for the SFtpStream and its variants: They are quite handy indeed. Thanks!

Quote:
I would like to learn a bit about this. Can you give me some link(s)?

Sorry, I should've phrase my words better: SSH and its relevant protocols are just fine. The problem I mentioned arises with implementations (on server or client side).
As you know, ssh is inherently an asynchronous communication protocol relying on channel

multiplexing. So both servers and clients need to keep track of every package they sent and received.
In some cases where multiple channels are requested at the same time (be it in non-blocking-ST or in MT) channel requests fail with "Failed waiting for channel success" message.
The reason seems to be that SSH_CHANNEL_SUCCESS message for some channels are either somehow get lost in the process or not delivered in due time.
(Yet we get a valid handle for the requested channel from libssh2. So while we got a valid channel handle in the end, the subsequent calls would still fail.)
This is possibly a result of a race condition (at implementaion level).

I wrote that locking/unlocking mechanism as a workaround for this issue. Basically it functioned as a crude bu effective "WouldBlock()".
It allowed us to maintain the non-blocking behaviour while sequentially initalizing the requested channels.

Best regards,
Oblivion
