Subject: Re: SSH package for U++
Posted by Oblivion on Thu, 09 Aug 2018 14:24:47 GMT
View Forum Message <> Reply to Message

Hello Mirek,

Quote:
Sure, as I said that was the point where I decided that fully non-blocking mode is "blocking" this kind of interface.

I'm sorry but I really don't understand this one.
Here is the snippet of the working version of the same method (bot in blocking and non-blocking mode) from the now-cancelled-update:

```
int SFtp::Read(SFtpHandle handle, Event<const void*, int>&& consumer, int size) // Data read engine.
{
	int sz = min(size, ssh->chunksize)
  Buffer<char> buffer(sz);
 int rc = libssh2_sftp_read(HANDLE(handle), buffer, sz);
 if(!WouldBlock(rc) && rc < 0)
  SetError(rc);
 if(rc > 0) {
  consumer(buffer, rc);
  sftp->done += rc;
		if(WhenProgress(sftp->done, size))
   SetError(-1, "Read aborted.");
  ssh->start_time = msecs();
 }
 return rc;
}

int SFtp::Get(SFtpHandle* handle, void* buffer, int size)
{

	Clear();
  Cmd(SFTP_GET, [=]() mutable {
  int rc = Read(
   handle,
   [=](const void* p, int sz)
   {
     if(!buffer)
      SetError(-1, "Invalid pointer to read buffer");
    memcpy((char*)(buffer + sftp->done), (char*)p, sz);
   },
   size
  );
  if(rc >= 0)
   sftp->value = sftp->done;
  return rc == 0 || sftp->done == size;
 });
 return sftp->done;
}
```

This works as expected. Note that for any kind of get method all you have to do is simply wrap the "engine" to suit your needs.

Quote:
I think there is a race condition here - two threads can obtain this lock simultaneously. Now I am not sure whether is this supposed to be MT safe, but if not, why atomic, right?

Yep. You see, an ssh shell is a complex environment. You cannot initalize multiple shells, and/or exec channels at once (I don't want to go into details here). This is a limitation of the libssh2. Their initialization have to be in some way serialized. These so-called Lock/Unlock methods handle that serialization when multiple shells or execs were started while maintaining a single threaded non-blocking and/or multithreaded asynchronous initalization. This is what makes multiple shells or exec channels at once (and the SshShellGUI example, for that matter) possible. I added the lock for NB and started to convert it into a thread-safe version. But the idea was yet to be finalized. In the end (in the now-cancelled-update) I decided to go with a RWMutex instead.

Quote:
I am also pretty ambivalent about all those static AsyncWork methods. I think these are better left to client code. Especially if we abandon the non-blocking mode.

Ok, this is not something I would object. Once the new SSH package is done I'll write a small complementary package with a handful of MT convenience functions only and maintain it in my git repo.

To sum up:

If you find it reasonable, I'll rewrite the SSH package with only blocking mode in mind (but with neceasary thread safety, and add its components gradually).

But as I already wrote in my previous messages, let us not change the "single session-multiple channels model" and not exclude any components.
When you start working in environments that rely on ssh ecosystem, as I do, you eventually end

up working with ssh tunnesl, exec, and shells.
Besides the current shell implementation we have is AFAIK one of its kind. :) I don't know any other open source shell component that does what Upp::SshShell does.
Hence it has educational and advertorial value. I mean, If you look up on stackoverflow or other relevant sites you'll see that even a barebone practical ssh shell implementation with libssh2, and also libssh is a mystery to people, let alone a shell that can have running multiple instances at the same time and that even works under Windows dumb-console. We can use this to promote U++, by writing a tutorial, demonstrating the shell in, say, codeproject.

Is this OK?

If so, I'll start writing the new code and commit the changed package within next week.

Best regards,
Oblivion