Subject: Re: SSH package for U++
Posted by Oblivion on Wed, 08 Aug 2018 12:16:38 GMT
View Forum Message <> Reply to Message

Is there any real-world example where you would need this kind of nonblocking behaviour?

All I can came up is some code that communicates with thousands of ssh servers at once. Looks very unlikely to me....

I use it in an app and for a limited number of ssh channels (usually 10-20).
But frankly, that code remains before the the CoWork improvements and the arrival of AsyncWork.
Nowadays in most such cases I use the async methods.

A short technical info:

- Ssh-based classes use a single callable target queue (FIFO), relying on deferred execution.

- Call to the queue elements are protected by a single static mutex (using INTERLOCKED).

- This allows easy maintenance but has a negative impact on the asynchronicity (it is not really possible to achieve %100 asynchronous operations with the design of libssh2 anyway. Yet the performance gains can be up to %30 in MT mode)

- Queue is populated by two interrelated methods: Cmd/ComplexCmd.
  Cmd: Adds a single ananymous function to the callable target queue. In blocking mode it executes the function immediately. In non-blocking mode it defers the execution.
  ComplexCmd: Firs and foremos,t this method acts as a nest for other Cmds. and other nests. This way it is possible to execute command chains in a consistent way (as if they are a single CMD).

Certainly you are not comfortable with the current implementation. (Besides, I haven't tested it yet bu as far as I can see your Get() implementation in svn won't work in non-blocking mode)

I have a new proposal: What if I get rid of queue mechanism and rewrite the package with only blocking mode and optional async transfer methods(using AsyncWork, and naming them agein SFtp::Asyncxxx)?
It won't take more than a week for me to come up with a working SSH package and the existing public API wont change much (only the NB helpers will be gone).
Besides its SC will be lot cleaner.

In fact, I had a working prototype of this (was using my Job class) I ditched in favour of the existing version.

As to your Get implementation:

```
int SFtp::Get(SFtpHandle handle, void *ptr_, int size0)
{
 int done = 0;                          // <- Can't be used in non-blocking mode.
 char *ptr = (char *)ptr_;
 Cmd(SFTP_START, [=, &done]() mutable {
  int size = size0;
  if(OpCode() == SFTP_START) {
   if(FStat(HANDLE(handle), *sftp->finfo, false) <= 0) OpCode() = SFTP_GET; // <- This is for
higher-level api. should be removed.
   else return false;
  }
  while(size) {
   int rc = libssh2_sftp_read(HANDLE(handle), ptr, min(size, ssh->chunk_size));
   if(rc < 0) {
    if(!WouldBlock(rc))
     SetError(rc);
    return false;
   }
   else {
    if(rc == 0)
     break;
    size -= rc;
    done += rc;
    if(WhenProgress(done, size0))
     SetError(-1, "Read aborted.");
    ssh->start_time = msecs();
   }
  }
  LLOG(Format("%d of %d bytes successfully read.", done, size0));
  return true;
 });
 return done;
}
```

I haven't tested this yet, but it shouldn't work in non-blocking mode. (because the execution will be deferred (Get will immediately return) and there is a local variable ("done"))

Quote:
Are they documented to be the same?

Nope, its my fault.
Best regards,
Oblivion.