Subject: Adding network proxy support to U++
Posted by Oblivion on Sun, 12 Oct 2014 13:45:33 GMT
View Forum Message <> Reply to Message

Hello guys,

One thing I miss in the U++'s network/socket framework is a generic network proxy support.
I am aware that some of the net classes (HttpClient/Pop3, for example) have built-in proxy support, but then, they support only the HTTP_CONNECT tunneling.
While this might be sufficient for the above mentioned classes, other socket operations can and might require listening/binding, in which case SOCKS proxy protocol(s) seem(s) to be the only (or a better) way to go, given the robustness of SOCKS protocol(s).
Also implementing the same code for every different class doesn't seem to me a good idea.

So here's the issue:
What started as a simple proxy code path for my FTP class now turned into a full-blown network proxy class supporting HTTP_CONNECT, SOCKS v4\4a\5 protocols.
Writing a synchronous generic network proxy class is actually very simple, given the simplicity of these widely used protocols.
On the other hand, writing an asynchronous proxy class is a bit harder, because of the increased code complexity, so I used HttpClient as a model.
Currenty I have a working (a)synchronous network class prototype called "NetProxy", which can connect via HTTP or SOCKS servers, using HTTP_CONNECT,SOCK4/4a/5 protocols, which allows both connecting and binding (only on Socks protocol) sockets through the respective proxy servers.

I would like to ask you two questions and hear your opinions on the topic:

1) In its present form, NetProxy class is a simple delegate/wrapper class, which takes a reference to an already constructed TcpSocket instance in its constructor, or through the NetProxy::Client() method, and simply stores a pointer to the socket. The thing is, I am not sure if this is the right or

way to go, or would it be better to have a TcpSocket derived class, say, something like ProxiedTcpSocket? (One advantage of delegate/wrapper class over TcpSocket derived class could be that the existing app code can be modified to take advantage of the proxy connections without too much hassle.)

2) Is an asynchronous (non-blocking) mode which gives some complexity to both api and internal code is good, or would a simple synchronous (blocking) mode suffice?

To give you the picture, here is stripped down version of the actual NetProxy prototype example (in async mode, but with only one socket.):

```
CONSOLE_APP_MAIN
{
 TcpSocket  s;
```

```
// NetProxy proxy(s, "127.0.0.1", 1080, "anonymous", "anonymous", NetProxy::SOCKS5);

 NetProxy proxy;
 proxy.Client(s);
 proxy.Server("127.0.0.1", 1080);

 proxy.Auth("anonymous", "anonymous");
 proxy.Remote("ftp.uni-erlangen.de", 21);
 SocketWaitEvent we;
 we.Add(s);
  we.Wait(100);
 for(;;) {

  proxy.Do();
  if(!proxy.InProgress()) {
   if(proxy.IsFailure()) {
    Cout() << "Proxy failed.\r\n";
    break;

   }
   else
   if(proxy.IsSuccess()) {
    Cout() << s.GetLine() << "\r\n";
    Cout() << "Success!";
    break;
   }
  }
 }
 s.Close();
}
```

What do you think?
If everything goes well, and after I have reliable code, I will upload the NetProxy to the Bazaar section, in case someone might find it useful.

(I'm not sure if this is the right place to post this, so please feel free to move this topic to the appropriate forum).

Regards,
Oblivion