从Fawcett的课程网页上，一共下载了至少3个不同版本的SocketCommunicator，其中有两个是功能比较齐全的，他们分别在我的SVN下的：

Pro1: SU\CSE687-OOD\Programs\Assigments\Project #4\SocketBlocks\SocketCommunicator

Pro2: SU\CSE687-OOD\Examples\SocketCommunicator

它们都用到了以下技术：

Lock

Thread & Thread\_Processing

Block & Blocking Queue(BQueue<>)

Socket

不同之处在于，Pro1使用std:string进行消息收发，而Pro2使用了一个Message类来进行消息收发。

Message类封装了一些处理XML的功能。

另外，Pro1还添加了MsgHandler和FileHandler用来分别处理消息和文件的收发，这两个类中都包含了BQueue<std::string>，因而没有处理XML的能力。

完成Project #4的关键在于，构建合理的消息和文件的收发进程

例如Pro1中的

class MsgReceiver\_Proc : public Thread\_Processing<MsgReceiver\_Proc>

class FileReceiver\_Proc : public Thread\_Processing<FileReceiver\_Proc>

class FileSender\_Proc : public Thread\_Processing<FileSender\_Proc>

class MsgSender\_Proc : public Thread\_Processing<MsgSender\_Proc>

或例如Pro2中的

class SendThreadProc : public Thread\_Processing<SendThreadProc>

class ConnectHandler : public Thread\_Processing<ConnectHandler>

class ListenProc : public Thread\_Processing<ListenProc>

它们的共同点在于:

1. 继承Thread\_Processing模板，该模板使用该类作为模板参数，这是为了类对象可在线程工作
2. 一个特殊参数的构造函数，传入可以获得BQueue的对象
3. Pro1的格式为：MsgReceiver\_Proc(IMsgHandler\* )
4. Pro2的格式为：SendThreadProc(Socket& , BQueue<Message>& )

之所以有这样一个构造函数，是因为这些类的对象是在另一个类（Sender & Receiver）中构造产生的，另一个类会传入需要的信息给类对象，并交给线程对象Thread构造出一个线程

1. 实现了继承下来的visual void run()函数，这是在Thread对象中调用的方法，线程中的Socket获取到了消息或文件就激活run。
2. run中首先获取BQueue和其他信息，进行死循环操作，等待消息的到来，当有消息进入时，就开始处理BQueue中的消息，当BQueue为空，又进入消息等待

因此，完成Project #4，需要的就是重写Pro1中的四个Proc，并对不同的消息作出不同的响应方式，而不是简单的打印到console。

不过，这样其实我们对于上述提到的各种技术，都没有领悟和掌握，但是为了赶project，不妨留到以后研究吧

Pro1是功能比较强大的版本，因此建议使用该版本。

所有Sender和Receiver的功能都集成在了一个Communicator类之中。只要自己决定新建的实例要作为Sender还是Receiver即可。

对于Sender，主要使用：

sndr.setFileSource(".\\debug\\toSend\\");

sndr.postMessage("fourth and still longer and most verbose message");

sndr.postFile("test.txt");

两个post方法直接都是调用了，接着BlockHandler会处理socket连接和发送（暂不清楚，以后再研究）

对于Sender用于发送的postMessage和possFile方法在其内部主要进行了三个步骤：startMsgBlock

midMsgBlock

endMsgBlock

其不同之处在于，分别给予type类型指示：Block::message或Block::file；另外对于File还会打开文件，并反复调用midMsgBlock方法，将文件中的内容发送过去。socket发送的具体方式，应该是在Communicator::postBlock中实现【暂时不深入调查】

另一端，Receiver首先会建立起ListenProc线程，一直等待其它端口来连接socket，当建立连接的时候，再建立起BlockProc线程等待对方放送消息过来，然后从sock中获取Block，然后根据Block的类型判别下边需要处理的是Message还是File。

线程会把内容交给对应的Handle方法(handleMsgBlock或handleFileBlock)，方法中再调用对应Handler的post方法：

MessageHandler：postMessge()将消息打入到BlockingQueue中

FileHandler：postFileName()根据名字建立文件，并将filename打入到BlockingQueue;

postFileBlock()向打开的文件中写入传过来的文件内容。当结束是将文件关闭。

MessageHandler和FileHandler中的会建立起Proc线程，并不断希望从BlockingQueue（通过deQ()方法）获取内容进行操作。当Queue无内容的时候，会形成等待状态。Handler和内部的Proc线程到底是如何进行通知，并不清楚。

从deQ中的WaitForSingleObject(\_hEvent,INFINITE);和enQ中的SetEvent(\_hEvent);来看应该是通过事件激发来完成的。

对于Handler中的Proc，会通过模板，在Handler实例化出一个对象的时候，在构造函数中建立Proc对象，并放在线程中运行。

根据上边的分析，对于Communicator来说，两个Handler不能缺少的，所以在初始化Communicator的对象时候一定通过attach方法将handler放进对象中如下：

Communicator rcvr(rep);

MsgHandler<MsgReceiver\_Proc> rMsgHandler;

rMsgHandler.setCommunicator(&rcvr);

rcvr.attachMsgHandler(&rMsgHandler);

FileHandler<FileReceiver\_Proc> rFileHandler;

rFileHandler.setFileDestination(".\\debug\\received2\\");

rFileHandler.setCommunicator(&rcvr);

rcvr.attachFileHandler(&rFileHandler);

rcvr.listen();

Pro1的线程情况：

|  |  |  |
| --- | --- | --- |
| 名字 | 创建者 | 作用 |
| listenProc | Communicator::listen() | 监听其他Comm对自己的Socket的连接请求 |
| BlockProc | listenProc::run() | 接收Socket的Block，并根据Block类型展开处理 |
|  |  |  |
|  |  |  |
| MsgReceiver\_Proc | MsgHandler::MsgHandler() | 响应从Sender那发来的普通消息 |
| FileReceiver\_Proc | FileHandler::FileHandler() | 响应从Sender那发来的文件消息 |
| MsgSender\_Proc | MsgHandler::MsgHandler() | 响应从Receiver那发来的普通消息 |
| FileSender\_Proc | FileHandler::FileHandler() | 响应从Receiver那发来的文件消息 |
|  |  |  |
|  |  |  |

因此关键之处，跟前面讲到的一样，还是要对四个Proc进行完善。

虽然说基本的结构如上所讲，但是在代码中还是有很多地方看到不停地在调用clone函数来构建对象的新实例，对于这一点依然不是很清楚。有待进一步查看。

Question for Project #4

如果用static library,那么我们就需要build solution两次才能得到正确的library，并让其它project进行使用。但是这个时候会出现warning和error，于是就会造成失分。如何只build 单个project而不是整个solution。

当有多个Client像Repository发送连接请求的时候，如何区别发送过来的消息是哪个Client的？因为所有的内容都是线程安全的，所以服务器端应该是可以做到被多个客户同时进行访问的吧？

还是说会为每一个客户做一个Server供其连接

需要Delete功能吗？如果需要，如何处理？

当Client的WebForm开始运行的时候，在用户登录之后，我们是让Client和Repository之间一直保持连接，还是每次用户有动作的时候建立连接并post消息？（个人认为应该是一直保持连接状态）