简单 FTP 协议演示客户端 MyFtp 设计文档

作者：夏鸣远 5070309676 F0703024

目录

实验任务与目的

实验目的

使用 C++语言实现 Windows 下的 ftp 客户端程序，加深对 ftp 协议工作原理的理解并了解网络编程（包括 tcp 的使用）。

完成情况

本次实验的成果实现了 ftp 协议基本的连接、登陆、切换目录、枚举目录项、下载文件功能，并使用良好的图形化用户接口供用户使用，同时注意错误控制例如登陆时的失败管理、TCP 超时异常管理等。

本程序并非使用传统 VC+MFC 库 开发而成，而是使用了有较好的可移植性以及物件模型的 Qt 实现，选择 Qt 的原因主要有：

- MFC 并不是严格意义上的 C++库，而是一个和 windows 平台有着莫大关系的专用库，而 Qt 起源于 Linux KDE，一直注重程序的可移植性，目前可以在包括 Linux、Windows、Mac、Windows CE、Nokia Symbian OS 等平台上运行，支持 C++、Java 以及各类脚本语言，同时有丰富的库支持，不仅可以高效的完成 GUI，也可以使用 XML、Network、OpenGL、Script 等功能。
- Qt 实现严谨，编码严格，与 C++结合地非常好，使用了新颖的机制例如信号（signal）、槽（slot）、事件 （event）、元类型编译器（MOC）同时提供了良好的编码环境、高效的库成员以及各类设计模式的支持。对于想使用 Qt 快速开发 GUI 的用户可以通过快速使用丰富的库完成开发（拥有类似于 VB 的简易 GUI 设计以及 java 的丰富库函数），对于想使用 Qt 进行高级编程或者高性能编程的用户，丰富而且书写十分规范的库无疑是最佳选择。
- Qt 支持源代码层面的移植，在不同平台上（嵌入式、微机等）只需要将 Qt 的程序重新编译即可以在平台上运行，同时兼顾了一致性和效率（不同于面向二进制代码兼容的 JAVA）。此外 Qt 完全符合软件工程的要求使得从源头杜绝不良编码习惯的产生。

FTP 协议简介

FTP 的历史

FTP 的发展经历了很多年。附录 III 是按年代编辑的关于 FTP 的 RFC 文档。其中包括最早在 1971 年提出用在 M.I.T.主机上的文件传输机制（RFC 114），以及 在 RFC 141 中的注释和讨论。RFC 172 提供了一个在主机（包括终端 IMP）间基于用户层协议的文件传输方法。RFC 265 做为其修订，通过附加评论重定义了 FTP，RFC 281 建议进一步改进。“Set Data Type”在传输中应用在 1982 年 1 月的 RFC 294 中提出。

RFC 354 废弃了 RFC 264 和 265。文件传输协议被定义为 ARPANET 上主机间的文件传输协议，

FTP 的主要作用则被定义为用来在主机间高效可靠地传输文件以及对远程存储的方便使用。RFC 385 进一步讨论了协议的错误，重点和一些附加内容，RFC 414 提供了使用中的 FTP 状态报告。1973 年的 RFC 430（被其它 RFC 多次引用），提供了对 FTP 的进一步讨论。最后，一个“官方的”FTP 文件在 RFC 454 中发布。

至 1973 年 1 月，FTP 协议又有了大量的变化，但总体的结构仍保持一致。为了应对这些变化，RFC 542 中发布了新的“官方”规范。但很多基于老规范的应用并没有被更新。

1974 年，RFC 607 和 614 继续讨论 FTP。RFC 624 提出进一步改变设计和一些小的修补。1975 年，RFC 686 用题为“Leaving Well Enough Alone”讨论早期以及近期 FTP 版本的差别。RFC 691 对 RFC 686 中关于打印文件部分做了小的修订。

在之前所有努力的基础上，通过将底层的传输协议由 NCP 改为 TCP，应用 TCP 的 FTP 协议在 RFC 765 中诞生了。

FTP 工作原理

FTP 使用命令头和参数的方式（例如 USER kenmark 中 USER 是命令头 kenmark 是参数）发送和请求服务，即在 control channel 上所有的命令以及反馈都是通过字符串形式的命令行完成的（可以用 telnet 检验）

FTP 数据传输有两种使用模式：主动和被动。主动模式要求客户端和服务器端同时打开并且监听一个端口以建立连接。在这种情况下，客户端由于安装了防火墙会产 生一些问题。所以，创立了被动模式。被动模式只要求服务器端产生一个监听相应端口的进程，这样就可以绕过客户端安装了防火墙的问题。

一个主动模式的 FTP 连接建立要遵循以下步骤：

1. 客户端打开一个随机的端口（端口号大于 1024，在这里，我们称它为 x），同时一个 FTP 进程连接至服务器的 21 号命令端口。此时，该 tcp 连接 的源端口为客户端指定的随机端口 x，目的端口（远程端口）为服务器上的 21 号端口。
2. 客户端开始监听端口（x+1），同时向服务器发送一个端口命令（通过服务器的 21 号命令端口），此命令告诉服务器客户端正在监听的端口号并且已准 备好从此端口接收数据。这个端口就是我们所知的数据端口。
3. 服务器打开 20 号源端口并且建立和客户端数据端口的连接。此时，源端口为 20，远程数据端口为（x+1）。
4. 客户端通过本地的数据端口建立一个和服务器 20 号端口的连接，然后向服务器发送一个应答，告诉服务器它已经建立好了一个连接。

典型的 ftp 会话解析

220 Serv-U FTP Server v6.0 for WinSock ready...IMPORTANT!!!============To safely use secure-FTP with Serv-U, using SSL/TLS to encrypt the FTPsession and file tranfers, you *MUST* create your own certificate. To dothis go to the 'Server | Settings | SSL Certificate' tab and fill in yourown entries for the certificate presented there (you can get contextsensitive help for this by pressing F1).

ftp>> USER kenmark

331 User name okay, need password.

```
ftp>> PASS kenmark
230 User logged in, proceed.
ftp>> PASV
227 Entering Passive Mode (127,0,0,1,237,216)
ftp>> LIST
150 Opening ASCII mode data connection for /bin/ls.
226 Transfer complete.
```

- 链接建立后第一条其实是欢迎信息，一般为多行例如
  220-hi
  220-i am kenmark
- 接着发出 USER <用户名>启动登陆，服务器会告知需要/不需要密码
- 使用 PASS 输入密码，服务器会返回是否正确 5 开头的回复是强制中断，2 则为成功
- PASV 命令转入被动数据传输模式，即一旦要发生数据传输，由 client 用 socket 去链接主机(127,0,0,1,237,216)表示的 socket 为 127.0.0.1:237<<8|216（注意最后 port 的计算）
- 然后调用 LIST 获取目录信息，这是一次典型的数据传输，服务器会返回 150 表示让 client 去之前 pasv 指示的 socket 去接受二进制数据
- 最后服务器会回复 226 表示传输成功

Qt 简介

Qt（发音同 cute[1]）是一个跨平台的 C++应用程序开发框架，有时又被称为 C++部件工具箱。Qt 被用在 KDE 桌面环境、Opera、OPIE、VoxOx、Google Earth、Skype 和 VirtualBox 的 开发中。它是诺基亚（Nokia）的 Qt Development Frameworks 部门的产品。

经过多年发展，Qt 不但拥有了完善的 C++图形库，而且近年来的版本逐渐集成了数据库、OpenGL 库、多媒体库、网络、脚本库、XML 库、WebKit 库等等，其内核库 也加入了进程间通信、多线程等模块，极大的丰富了 Qt 开发大规模复杂跨平台应用程序的能力，真正意义上实现了其研发宗旨“Code Less; Create More; Deploy Anywhere.”。

由于各家编译器良莠不齐。Qt 本身为了跨平台兼容性，只能以“最低兼容规格”来设计。因此 Qt 必须具备 RTTI、动态创建、 Persistence/Serialization 的基础建设，以及建构出自己的容器组件。

下列模块提供一般的软件开发

- QtCore—QtCore 模块是所有基于 Qt 的应用程序的基础，提供信号与槽的对象间通信机制、IO、事件和对象处理、多线程
- QtGui—包含了开发图形用户界面应用程序所需的功能。使用其支持的各个平台的本地图形 API。支持反锯齿、向量形变。支持 ARGB 顶层 widget
- QtMultimedia
- QtNetwork—提供了网络程序设计功能。支持通用协议，如 HTTP、FTP 和 DNS，包括对异步 HTTP 1.1 的支持。与较低层的 TCP/IP 和 UDP 协议，如 QTcpSocket、QTcpServer 和 QUdpSocket
- QtOpenGL—提供在应用程序中使用 OpenGL 和 OpenGL ES 加入 3D 图形。在 Windows 平台上亦支持 Direct3D
- QtOpenVG
- QtScript— 包含完全集成的 ECMA 标准脚本引擎。提供信号与槽机制简化对象间通信和

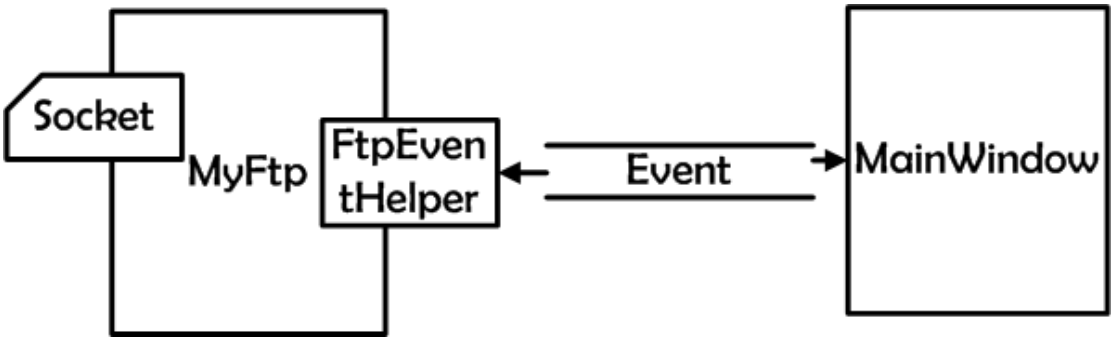
QtScript 调试器。

- QtScriptTools—额外的 Qt Script 组件
- QtSql—将数据库集成至应用程序。支持所有主要的数据库驱动包括 ODBC、MySQL、PSQL、SQLite、 ibase、Oracle、Sybase、DB2。
- QtSvg—支持 SVG 格 式
- QtWebKit—集成 WebKit，提供了 HTML 浏览器引擎，便于在本地应用程序中嵌入网络内容和服务。
- QtXml—提供了 XML 文档的阅读器和编写器、支持 SAX 和 DOM。
- QtXmlPatternsl—提供了 XQuery 和 XPath 引擎支持。
- Phonon—集成 Phonon，支持跨平台应用程序播放音频和视频内容。
- Qt3Support—模块提供兼容 Qt 3.x.x 版本的程序库
- 作业于 Qt 附带工具的模块
- QtDesigner—提供扩充 Qt Designer 的类。
- QtUiTools
- QtHelp—协助集成在线文件到应用程序中。
- QtTest—提供单元测试框架和鼠标和键盘模拟功能。集成 Visual Studio 和 KDevelop。
- 下列模块用于 Unix 开发
- QtDBus
- 下列模块用于 Windows 开发
- QAxContainer
- QAxServer

软件设计

概述

本次实验用到了 QtCore、QtGUI、GtNetwork、QtUiTools
软件主要分成两个部分，用到了 TCP Socket、GUI（多种控件、Dock 类控件等）、自定义事件、多线程、正则表达式、信号和槽（Qt 特有）、异常控制等技术。
整个程序分为两个部分，分别是 UI 和 ftp PI（Protocol Interpreter 协议解析器），其中协议解析器使用 TCP Socket 与 ftp 服务器通讯，本身使用多线程与 UI 控制分开。解析和产生命令时使用了正则表达式。UI 部分维护整个主窗口的使用、控件的处理（使用了 Qt 物件模型中经典的信号与槽的方式）、命令反馈以及产生。两者通过自定义事件来通讯并保持显示和实际功能的同步性。
下面是软件的架构图：



用户界面 MainWindow

用户界面的设计图如下（使用了 Qt 自带工具 QtCreator 完成）
主要包括四个部分，位于窗口顶端的是连接信息部分，包括需要用户输入的主机信息，登陆信息以及一个 Combo 保存常用的连接信息，两个功能按钮分别用于保存连接信息以及连接/中断一个会话，复选框用于选择匿名登录。
中间部分两个白框（实际是 Table）用于显示本机目录和远程目录信息，通过选择目录下的项目并使用鼠标双击可以完成下载以及切换目录等功能。
下方是状态栏（用于显示连接状态）以及一个 Dock（设计图中不显示）用于显示 FTP 命令信息，Dock 可以停靠在主窗口的四周。
主窗口遵从经典的 Qt 设计，使用 signal/slot 的设计。Qt 中所有的 Widget 状态变化（例如一个按钮被按下，输入框内容改变）会通过 signal 的方式传出，通过 connect 到一个特别的处理函数（slot）可以实现从一个对象（例如按钮）到拥有处理函数的另一个对象（例如显示控制类）的传递。这样使得一个对象的状态变化可以用一种极为灵活的方式通知到另一个对象的处理函数，与传统 C++调用以及 MFC 的回调有较大的不同，使用起来更安全和方便。
主窗口实现的功能包括：

- 可 dock 的 ftp 命令窗口，实时显示 PI 收到以及发送命令的情况
- 本地和远程的目录形式的选择列表，通过简单的鼠标双机可以和资源管理器有类似的目录进入退出功能，同时可以完成下载文件
- 常用连接保存功能，用户可以将一个登陆信息（Host，port，username，password）组起一个名字保存起来，以后需要使用时直接从下拉式菜单里选择而不需要重复输入，方便用户使用 ftp。

由于图形化用户界面设计比较繁琐而且并不是程序的主要部分，所以这里略去，如有兴趣可以参考源代码文件中的 mainwindow.cpp 和 mainwindow.h

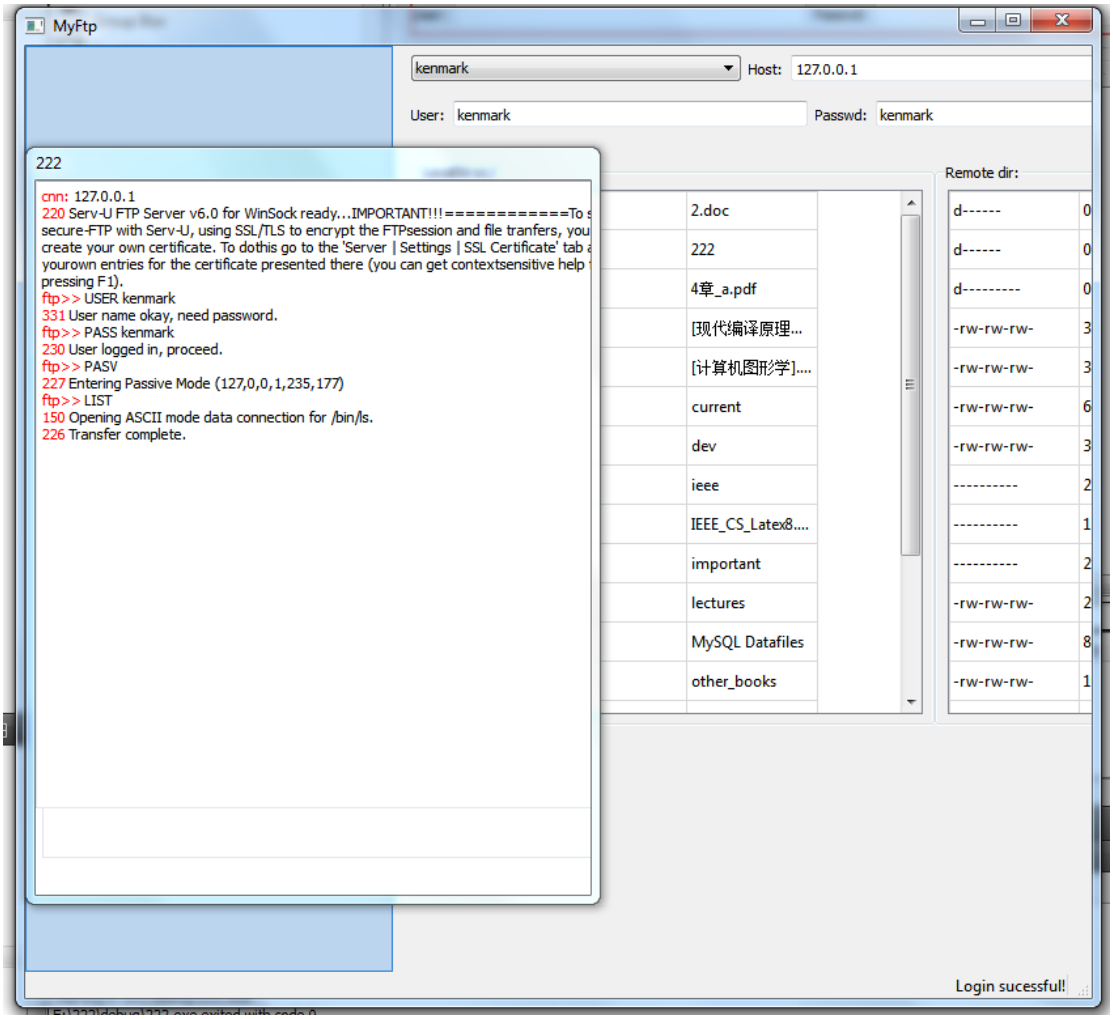


FIG.Dockable 浮动框

FTP PI 设计

FTP PI 将传统的 ftp 指令组合成为一个个功能并通过响应事件的方式暴露给主窗口进行调用，这样可以降低两者的耦合程度，便于拓展以及单元测试。本类被设计集成字 QThread 以达到能够与主窗口分离执行的功能。

PI 目前支持的功能有：

- 创建一个 PI：要求传入主机的信息包括 ip 和端口，但并不连接，等待启动消息队列后连接 socket 并处理消息。值得注意的是，socket 资源归 PI 线程所有，需要在进程开始运行后才能申请，如果在构造函数中申请，则会属于主线程（GUI 线程）导致 PI 无法和主线程同时运行。
- login 功能：通过 FtpLogin 事件传入用户名和密码，登陆服务器，会通过事件 UpdateLogin 通知主线程登陆的情况。
- Cd 功能：通过 FtpCd 事件传入需要改变的目录名，PI 会通知服务器改变当前工作目录并返回是否成功。
- List 功能：通过 FtpList 事件传入目录路径，PI 会通知服务器传送目录下所有文件的信息，完成后通过时间 UpdateDir 传回主线程。
- downloadFile：传入文件名表示需要获取的远端文件名以及 QFile 对象表示本机的存储

点，PI 会向服务器请求文件，完成后返回。

事件机制

事件是一个异步操作，Qt 中允许给所有继承自 QObject 的对象发送 event，通过重载处理函数可以截获事件并进行处理。值得注意的是，事件是通过每个线程独享的事件队列分发的（不可以通过向线程 B 发送属于线程 A 中的资源的事件来让 B 线程执行对 A 线程资源的管理，一个典型的例子是 socket。Socket 是线程独享的资源，每当打开一个 socket 之后，Qt 会把 socket 的 notifier 绑定在事件队列上，当有新的数据包到达时 notifier 会通过事件的方式通知 socket 进行进一步处理（例如触发一个 signal 并带动其连接到的槽 slot）。所以在设计本次程序时，为了防止 GUI 进程因为 socket 传输等待而假死，决定使用多线程的方式，为使用某个线程（PI，GUI）中的服务时，通过事件通知的方式是一个可行并高效的同步方式。下图描绘了一次典型的事件处理过程：

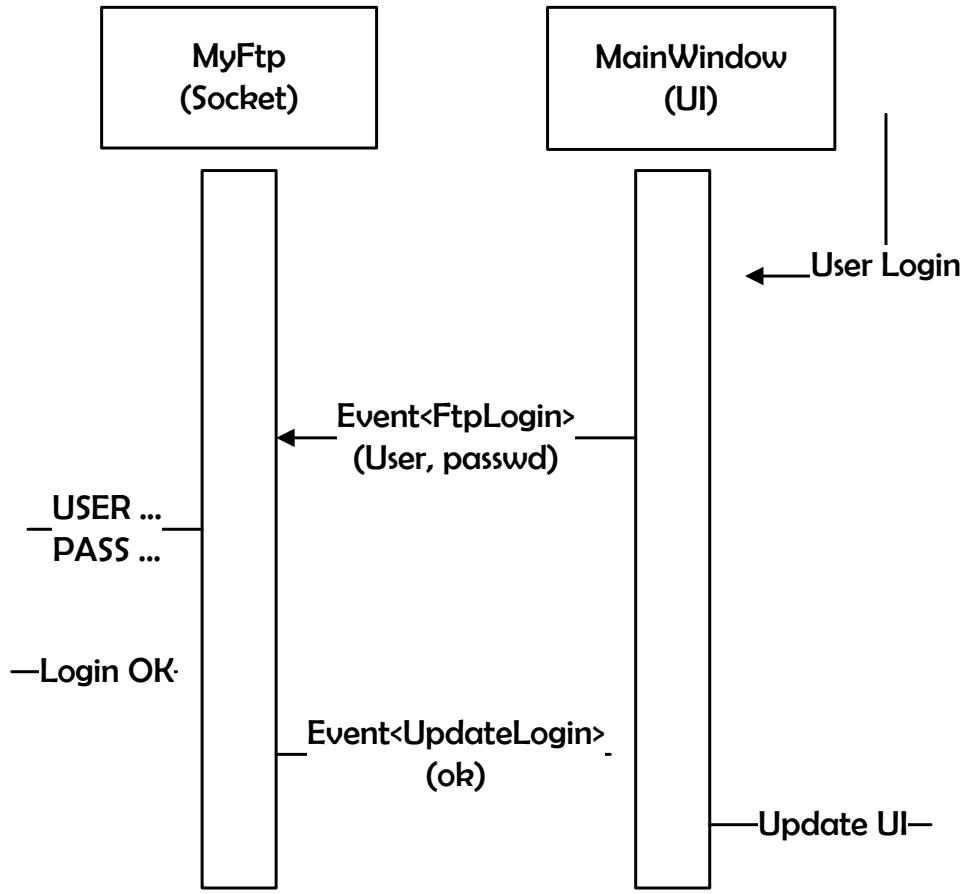


FIG.一次典型事件处理

本次程序中使用到的事件如下表所示：

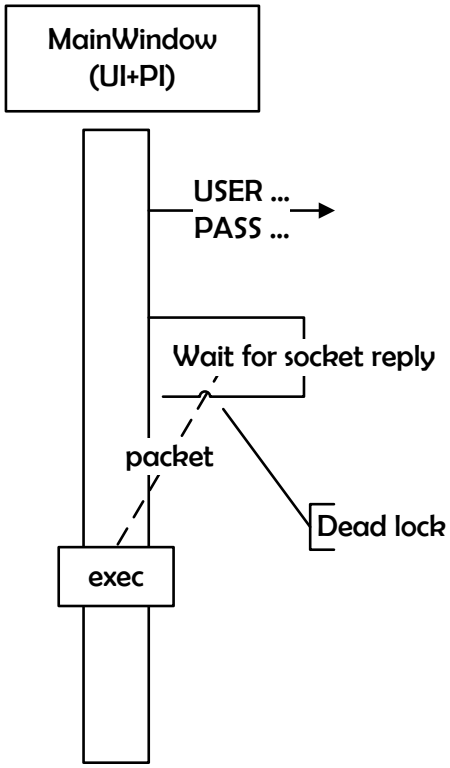
事件名	发送方	接收方	意义
FtpLogin	主线程	PI	发起一次连接请求，传递用户名密码
FtpList	主线程	PI	查询指定目录下的所有文件信息，传递目录路径
FtpCd	主线程	PI	切换工作目录
FtpDown	主线程	PI	下载文件，传入远端文件名以及文件对象
UpdateLog	PI	主线程	更新 log 信息，有 FTP 指令发生，传入 log 头和信息

UpdateDir	PI	主线程	更新目录信息，传入目录原始数据

难点解析

多线程处理 socket

这个是最早碰到的问题，当初想通过单线程的简单方式完成工作。但是发生的问题是：当从服务器等待某些 reply 时，服务器的确发出了 reply 但是 client 却在无限等待，通过翻找材料发现：socket 当有新的包时首先触发中断，qt 底层在事件队列中插入一个 socketDataReady 的事件等待处理，而可能用户在之前已经开始使用忙等待或者阻塞等待的方式开始等待，使得他等待的数据抵达事件需要在它完成后经过事件队列的分发才能到达，造成死锁，时间流程图如下：



解决方法是通过多线程方式，详细工作流程见“一次典型的事件处理流程”

资源分割和同步

前面已经提及一些资源是线程私有的（涉及事件分发的），例如 PI 无法直接修改主窗口中一个 log 框的内容，而主窗口无法直接操作 socket 给服务器发送命令。这种机缘分割造成的是同步的困难。幸运的是 Qt 提供了自定义事件的功能，使得发送发可以通过异步的事件向另一个线程请求服务，同时通过异步的方式接受反馈。而服务实在拥有资源的线程中完成的。参照“一次典型的时间处理”，由于 UI 的操作（输入、更新）只有 UI 线程能够访问，所以必须由他全权完成，而 PI 需要更新时需要通过 event 向其请求(Event<UpdateLogin>)，同样的，通过 socket 发送命令只能由 PI 完成，MainWindow 发送消息 Event<FtpLogin>请求该服

务，并且在收到 Event<UpdateLogin>时表示服务器有反馈。

正则表达式

此外在技术实现上还有个小问题即正则表达式的问题，问题被简化如下：
假设有字串”aaa　　bbb”，使用正则串”([^]*) *([^\n]*)”进行匹配（在 C++里面\需要双写\\）则结果\1 和\2 是怎样?显然\1=”aaa”因为匹配到第一个空格就无法匹配了，然而\2 却获得了”bbb”即空格也被加入了。理论上如果从左到右这些空格会被” *”全部消去，从而\2 获得理论上应该是”bbb”。问题的根源是现行的正则比较都是贪婪式的，而且是需要抓取部分文字是贪婪的，即当” *”（不需要抓取部分，单比较部分）和”([^\n]*)”同时出现时，以后者匹配更多字符优先，所以”bbb”之前的字符全归后者，找到的方法是强制让” *”也变为抓取部分，则正则比较引擎会按照从左到右的优先次序匹配和抓取字符。

调试与构建

构建过程

本软件使用 Qt 完成，需要有 Qt 的环境进行编译，有两种最简单的构建方式：
使用 QCreater 这个简易 IDE 构建一个工程，并使用其整合的 gdb 功能进行调试，这是最简单的方式，本软件开发过程中使用了这一方法，QtCreator 的使用不是本文档的重点，详细可以参考：http://qt.nokia.com/products/developer-tools/
第二种方式可以通过 console 模式构建，来到纯源代码目录下，纯源代码包括：
Mainwindow.cpp mainwindows.h 主窗口
MyFtp.cpp MyFtp.cpp MyFtp PI
Main.cpp main 函数
MyEvent.h 自定义事件
ConnectionPoint.h 方便用户保存的连接信息
FtpEventHelper 帮助截获从主 GUI 线程发来的事件并调用 MyFtp PI 相应功能的辅助类
Mainwindow.ui 祝窗口的用户界面描述文件

使用 Qt command prompt 来到目录下调用
qmake –project # 使用 Qt 自带构建系统 qmake 创建一个工程
qmake #使用 qmake 读取工程文件并产生传统 makefile
make #使用 gnu make 来产生程序
其中第一步 qmake 会扫描目录下所有的源代码并产生一个工程文件，例如：

```
# ----------------------------------------------
# Project created by QtCreator 2010-06-22T10:27:31
# ----------------------------------------------
TARGET = 222
QT += network
TEMPLATE = app
```

```
SOURCES += main.cpp \
        mainwindow.cpp \
        MyFtp.cpp
HEADERS += mainwindow.h \
        ConnectionPoint.h \
        MyFtp.h \
        MyEvent.h \
        FtpEventHelper.h
FORMS += mainwindow.ui
```

第二步会产生大量 makefile 包括 makefile.debug 和 makefile.release

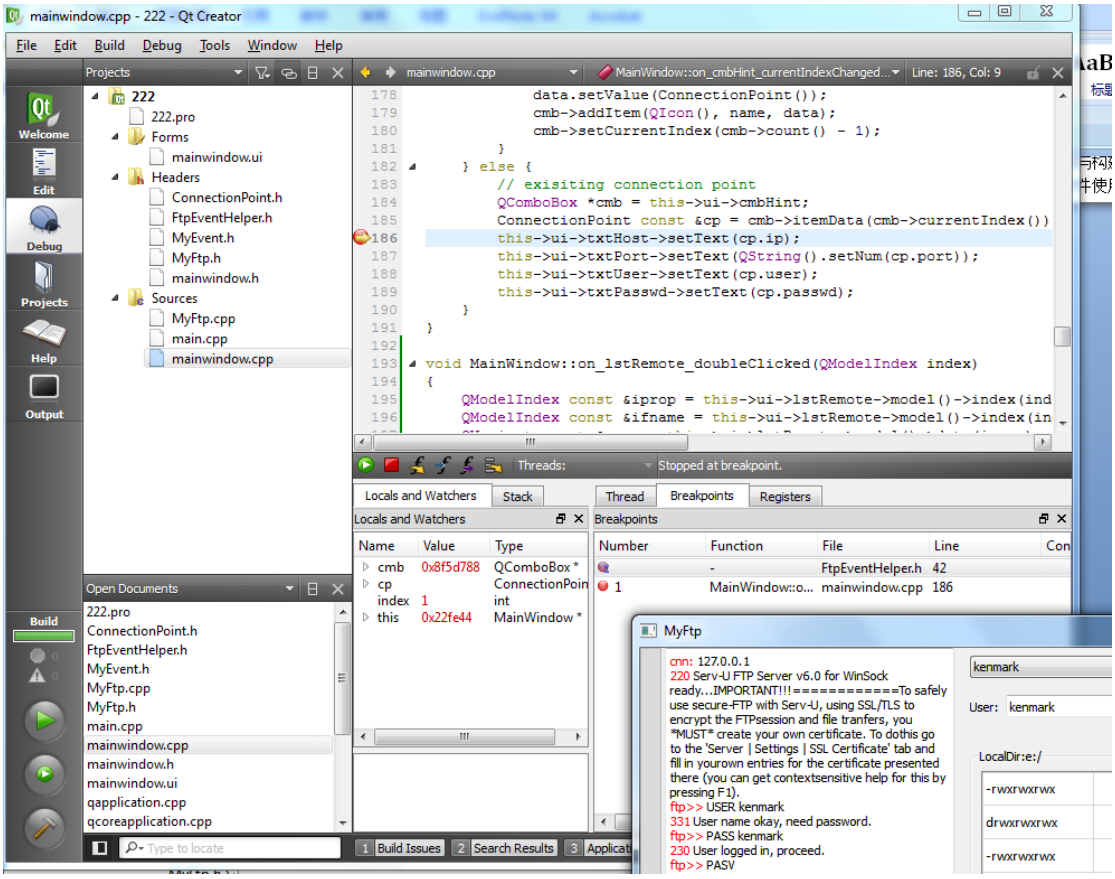
第三部可以使用-f 标志选取 debug 或者 release 模式构建

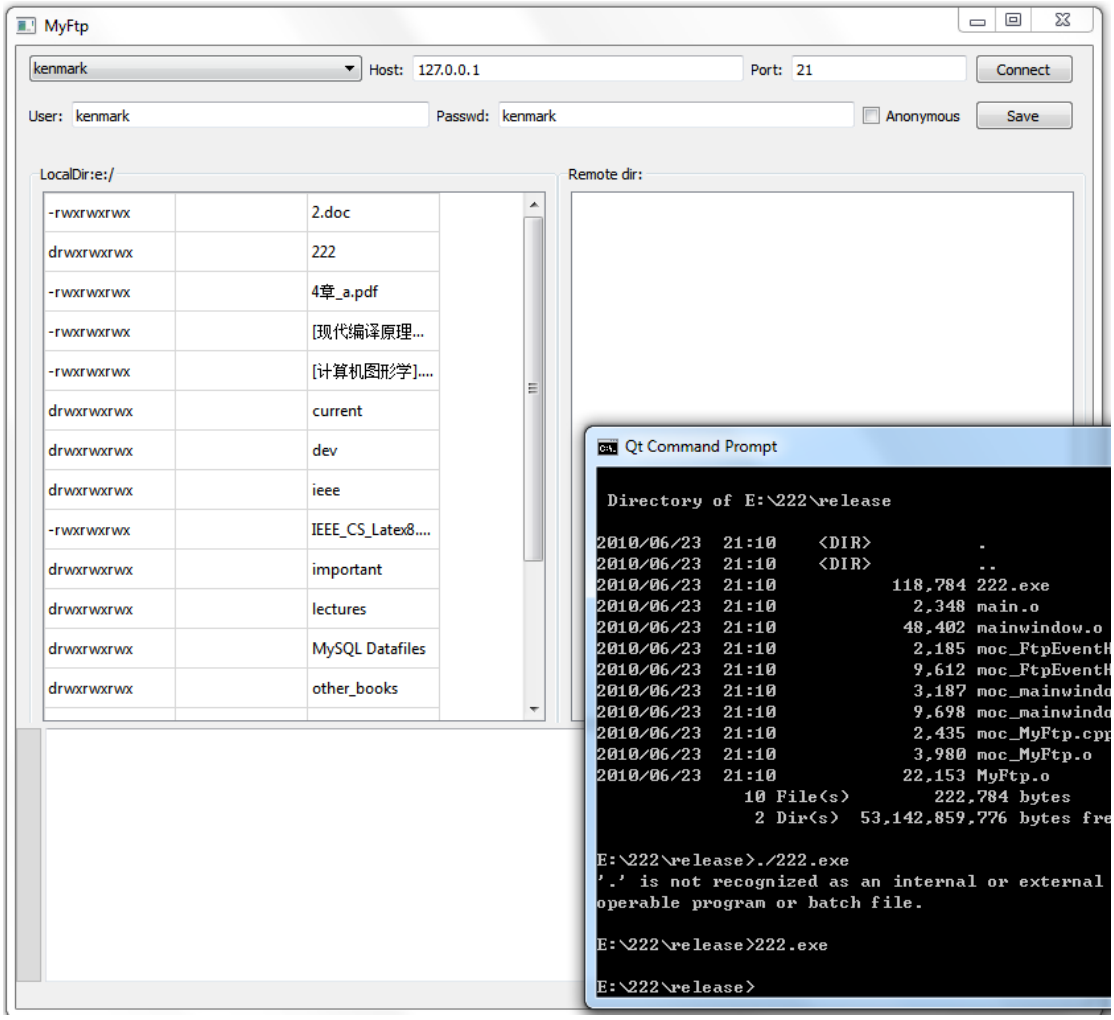
命令行构建：

```
E:\222>mingw32-make release
mingw32-make -f Makefile.Release
mingw32-make[1]: Entering directory `E:/222'
c:\Qt\2010.03\qt\bin\uic.exe mainwindow.ui -o ui_mainwindow.h
g++ -c -O2 -frtti -fexceptions -mthreads -Wall -DUNICODE -DQT_LARGEFILE_SUPPORT
-DQT_DLL -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -DQT_THREAD_S
UPPORT -DQT_NEEDS_QMAIN -I"c:\Qt\2010.03\qt\include\QtCore" -I"c:\Qt\2010.03\qt\
include\QtNetwork" -I"c:\Qt\2010.03\qt\include\QtGui" -I"c:\Qt\2010.03\qt\includ
e" -I"c:\Qt\2010.03\qt\include\ActiveQt" -I"release" -I"." -I"c:\Qt\2010.03\qt\m
kspecs\win32-g++" -o release\main.o main.cpp
g++ -c -O2 -frtti -fexceptions -mthreads -Wall -DUNICODE -DQT_LARGEFILE_SUPPORT
-DQT_DLL -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -DQT_THREAD_S
UPPORT -DQT_NEEDS_QMAIN -I"c:\Qt\2010.03\qt\include\QtCore" -I"c:\Qt\2010.03\qt\
include\QtNetwork" -I"c:\Qt\2010.03\qt\include\QtGui" -I"c:\Qt\2010.03\qt\includ
e" -I"c:\Qt\2010.03\qt\include\ActiveQt" -I"release" -I"." -I"c:\Qt\2010.03\qt\m
```

构建完成后可能需要将 qt 的 dll 设置为系统扫描目录，以保证 qt 运行时动态链接库可以被链接到。

下图分别是开发界面（调试）和脱离环境的运行模式：


mainwindow.cpp - 222 - Qt Creator
File Edit Build Debug Tools Window Help
Projects
222
222.pro
Forms
mainwindow.ui
Headers
ConnectionPoint.h
FtpEventHelper.h
MyEvent.h
MyFtp.h
mainwindow.h
Sources
MyFtp.cpp
main.cpp
mainwindow.cpp
Welcome
Edit
Debug
Projects
Help
Output
Build
mainwindow.cpp
MainWindow::on_cmbHint_currentIndexChanged...
Line: 186, Col: 9
178 data.setValue(ConnectionPoint());
179 cmb->addItem(QIcon(), name, data);
180 cmb->setCurrentIndex(cmb->count() - 1);
181 }
182 } else {
183 // exisiting connection point
184 QComboBox *cmb = this->ui->cmbHint;
185 ConnectionPoint const &cp = cmb->itemData(cmb->currentIndex())
186 this->ui->txtHost->setText(cp.ip);
187 this->ui->txtPort->setText(QString().setNum(cp.port));
188 this->ui->txtUser->setText(cp.user);
189 this->ui->txtPasswd->setText(cp.passwd);
190 }
191 }
192
193 void MainWindow::on_lstRemote_doubleClicked(QModelIndex index)
194 {
195 QModelIndex const &iprop = this->ui->lstRemote->model()->index(ind
196 QModelIndex const &ifname = this->ui->lstRemote->model()->index(in
Threads:
Stopped at breakpoint.
Locals and Watchers
Stack
Thread
Breakpoints
Registers
Locals and Watchers
Name Value Type
cmb 0x8f5d788 QComboBox *
cp ConnectionPoin
index 1 int
this 0x22fe44 MainWindow *
Breakpoints
Number Function File Line
- FtpEventHelper.h 42
1 MainWindow::o... mainwindow.cpp 186
Open Documents
222.pro
ConnectionPoint.h
FtpEventHelper.h
MyEvent.h
MyFtp.cpp
MyFtp.h
main.cpp
mainwindow.cpp
mainwindow.h
mainwindow.ui
qapplication.cpp
qcoreapplication.cpp
Type to locate
1 Build Issues
2 Search Results
3 Applicat
MyFtp
cnn: 127.0.0.1
220 Serv-U FTP Server v6.0 for WinSock ready...IMPORTANT!!==========To safely use secure-FTP with Serv-U, using SSL/TLS to encrypt the FTPsession and file tranfers, you *MUST* create your own certificate. To dothis go to the 'Server | Settings | SSL Certificate' tab and fill in yourown entries for the certificate presented there (you can get contextsensitive help for this by pressing F1).
ftp>> USER kenmark
331 User name okay, need password.
ftp>> PASS kenmark
230 User logged in, proceed.
ftp>> PASV
kenmark
User: kenmark
LocalDir:e:/
-rwxrwxrwx
drwxrwxrwx
-rwxrwxrwx



常见问题收集

- 如果找不到 libgcc_s_dw2-1.dll，请将 qt/2010.3/mingw/bin 加入环境变量 PATH
- 请务必使用 qt 携带的 mingw 版本进行编译，因为安装的 qt 是使用携带版本的 mingw 编译的，如果版本不同可能会有链接上的问题，详细来说，如果之前没有装过 mingw，则没有该问题，如果有，请将 PATH 变量中原有的 mingw 路径删去。

用户手册

打开主界面，输入链接需要的信息，点击 connect 可以看到状态栏显示连接状态是否成功（密码验证错误不会登陆），同时 remote dir 下会出现远端 ftp 的文件结构，之后双极 local dir 中的目录可以进入本机目录。双击 remote Dir 下的文件夹可以进入 ftp 上的文件目录，双击文件可以下载远端文件到本地左边指示的工作目录。
下方的浮动框是 ftp 的协议信息，红色是协议主消息，可供专业人士查看。
如果不想重复输入连接点信息可以在下拉式菜单中选择 new connection point 新建一个连接点，并且填写信息后按下 save 保存，以后只需要在下拉菜单中找到保存项即可直接读取。
关闭软件可以直接点击关闭按钮或者按下 disconnect（如果已经在连接状态下），两者都会

中断当前的会话并退出程序。

实验体会

ftp 协议包含的内容很多，基本的文件功能还有很多，并且新的协议中加入了安全的一些元素，使得协议本身的状态机更加复杂，为了实现每一个转移（异常或者成功）需要花费很多时间，虽然本次实验由于时间有限没能全部实现，但实现了基本的功能，并且使用了新型的库 QT 来完成任务。

实验中碰到了很多问题并且都基本解决了，同时由于使用了 Qt 库，使得很多不良的编码习惯能够在最初被发现，通过实际完成 GUI 和 PI 部分的交互也更深入的了解了 ftp 协议的工作原理。

更多的功能例如 mkdir, upload, delete 可以通过简单在 event 里面增加相应项完成，但是由于异常控制会以几何指数增长（例如没有权限等），在时间有限的情况下并没有实现。

Qt 可以通过简单的重新编译移植到其他平台下，理论上，将工程在 linux 重新编译能够完全实现原有的功能，这也是 Qt 非常吸引人的一方面。

其他的文件协议例如基于 udp 的 tftp 以及安全的文件协议 sftp 都是从 ftp 演变而来对某些方面进行加强的版本，如果未来有可能，也很想实现一番。

实验中参考了 qt 源代码中的 qftp 设计，受到不少启发，同时 qassistant 是不可多得的有两文档和参考材料，几乎囊括了 qt 编程所需要的所有内容。