## 摘要

**实验目的：**本次计网实验的实验目的是设计一个FTP客户端，并要求其具有图形化界面、上传、下载功能。

**实验内容及亮点：**使用socket方式编程，从创建socket、建立TCP连接开始，实现FTP、协议的功能，未调用第三方控件。

编程环境为VS2017+Qt5.12，编程语言为C++11。

本实验亮点如下：

1.调用WINSOCK2.H 内容，封装了一个FTP工具库（FTPTools），实现上传，下载，文件列表自动同步，登入登出等功能接口，并处理服务器的响应信息。该工具类封装性良好，可在任何FTP相关工程内直接调用。

2.利用Qt信号/槽机制，实现图形化界面，并通过DnD（Drag and Drop）技术，完成拖拽上传功能。

**关键词：**FTP工具库封装，DnD拖拽上传，文件列表自动同步，Qt信号/槽编程

# 目录

[摘要 3](#_Toc535317461)

[1．实验目的及意义 6](#_Toc535317462)

[1.1实验目的 6](#_Toc535317463)

[1.2实验意义 6](#_Toc535317464)

[2．实验背景 7](#_Toc535317465)

[2.1 FTP协议概述 7](#_Toc535317466)

[2.1.1使用方式 7](#_Toc535317467)

[2.1.2传输方式 8](#_Toc535317468)

[2.1.3命令端口 8](#_Toc535317469)

[2.1.4数据端口 8](#_Toc535317470)

[2.1.5主动模式 (PORT) 9](#_Toc535317471)

[2.1.6被动模式 (PASV) 9](#_Toc535317472)

[2.1.7主要用到的 FTP 命令 9](#_Toc535317473)

[2.1.8 FTP 响应码 10](#_Toc535317474)

[2.2 socket编程概述 10](#_Toc535317475)

[2.2.1 概述 10](#_Toc535317476)

[2.2.2 主要命令 10](#_Toc535317477)

[2.2.2.1 创建 11](#_Toc535317478)

[2.2.2.2 绑定 12](#_Toc535317479)

[2.2.2.3连接 13](#_Toc535317480)

[2.2.2.4监听 13](#_Toc535317481)

[2.2.3 客户端编程模式 14](#_Toc535317482)

[2.3 Qt GUI框架概述 15](#_Toc535317483)

[2.3.1 Qt简介 15](#_Toc535317484)

[2.3.2 Qt图形用户界面 15](#_Toc535317485)

[2.3.3 信号与槽机制 15](#_Toc535317486)

[3. 实验设计 16](#_Toc535317487)

[3.1 FTPTools库 16](#_Toc535317488)

[3.1.1 概述 16](#_Toc535317489)

[3.1.2 外部接口介绍 17](#_Toc535317490)

[3.1.2.1 CreateSocketAddr\_in 接口 17](#_Toc535317491)

[3.1.2.2 ConnectServer接口 17](#_Toc535317492)

[3.1.2.3 LogIn接口 18](#_Toc535317493)

[3.1.2.4 SendCommand接口 18](#_Toc535317494)

[3.1.2.5 DownloadFile接口 19](#_Toc535317495)

[3.1.2.6 ListFile接口 20](#_Toc535317496)

[3.1.2.7 UploadFile接口 20](#_Toc535317497)

[3.1.2.8 PassiveMode接口 21](#_Toc535317498)

[3.1.2.9 FlushBuffer接口 23](#_Toc535317499)

[3.2 FTPClient主体程序 24](#_Toc535317500)

[3.2.1 工程结构 24](#_Toc535317501)

[3.2.2 界面展示 24](#_Toc535317502)

[3.2.3 功能实现 25](#_Toc535317503)

[3.2.3.1登入及查询文件列表 25](#_Toc535317504)

[3.2.3.2下载文件 27](#_Toc535317505)

[3.2.3.3登出 28](#_Toc535317506)

[3.2.3.4选中文件列表中的文件 29](#_Toc535317507)

[3.2.3.5上传文件 29](#_Toc535317508)

[4.结论 31](#_Toc535317509)

[4.1 实验收获 31](#_Toc535317510)

[4.2 未来工作 31](#_Toc535317511)

## 1．实验目的及意义

### 1.1实验目的

本实验是使学生熟悉网络规划与设计的基本知识和方法、掌握网络系统软件与应用软件开发的方法，能将所学的操作系统、数据库、软件工程、计算机网络等方面的知识集成到一起，规划、安装、调试实际网络系统、开发实际软件系统。

本实验是使学生掌握网络系统软件的开发方法、开发平台的使用、与实际数据库的集成方法。用C++实现FTP客户端。

### 1.2实验意义

该实验是理论知识和动手能力的综合体现。通过本实验，学生能掌握网络的规划与设计、安装过程；掌握网络系统软件、网络应用软件的开发方法、开发平台的使用、与实际数据库的集成方法。

## 2．实验背景

### 2.1 FTP协议概述

文件传输协议（FTP）作为网络共享文件的传输协议，在网络应用软件中具有广泛的应用。FTP的目标是提高文件的共享性和可靠高效地传送数据。

在传输文件时，FTP 客户端程序先与服务器建立连接，然后向服务器发送命令。服务器收到命令后给予响应，并执行命令。FTP 协议与操作系统无关，任何操作系统上的程序只要符合 FTP 协议，就可以相互传输数据。本文主要基于 LINUX 平台，对 FTP 客户端的实现原理进行详尽的解释并阐述如何使用 C 语言编写一个简单的 FTP 客户端。

相比其他协议，如 HTTP 协议，FTP 协议要复杂一些。与一般的 C/S 应用不同点在于一般的C/S 应用程序一般只会建立一个 Socket 连接，这个连接同时处理服务器端和客户端的连接命令和数据传输。而FTP协议中将命令与数据分开传送的方法提高了效率。

FTP 使用 2 个端口，一个数据端口和一个命令端口（也叫做控制端口）。这两个端口一般是21 （命令端口）和 20 （数据端口）。控制 Socket 用来传送命令，数据 Socket 是用于传送数据。每一个 FTP 命令发送之后，FTP 服务器都会返回一个字符串，其中包括一个响应代码和一些说明信息。其中的返回码主要是用于判断命令是否被成功执行了。

#### 2.1.1使用方式

**FTP的任务是从一台计算机将文件传送到另一台计算机，不受**[**操作系统**](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)**的限制，其使用方式大致包含以下两种类型**

1）

启动ftp客户程序工作的一途径是使用ie[浏览器](https://baike.baidu.com/item/%E6%B5%8F%E8%A7%88%E5%99%A8)，用户只需要在ie地址栏中输入如下格式的url地址：ftp：[//](https://baike.baidu.com/item///)[用户名：口令@]ftp服务器[域名](https://baike.baidu.com/item/%E5%9F%9F%E5%90%8D)：[[端口号](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3%E5%8F%B7)]

(在CMD命令行下也可以用上述方法连接，通过put命令和get命令达到上传和下载的目的，通过ls命令列出目录，除了上述方法外还可以在cmd下输入ftp回车，然后输入open IP来建立一个连接，此方法还适用于linux下连接ftp服务器)

通过ie浏览器启动ftp的方法尽管可以使用，但是速度较慢，还会将密码暴露在ie浏览器中而不安全因此有提出以下第二点

2）

需要进行远程文件传输的计算机必须安装和运行ftp客户程序。在windows操作系统的安装过程中，通常都安装了tcp/ip协议软件，其中就包含了ftp客户程序。但是该程序是字符界面而不是图形界面，这就必须以[命令提示符](https://baike.baidu.com/item/%E5%91%BD%E4%BB%A4%E6%8F%90%E7%A4%BA%E7%AC%A6)的方式进行操作，很不方便。

因此一般都安装并运行专门的ftp客户程序。

1．在本地电脑上登陆到国际[互联网](https://baike.baidu.com/item/%E4%BA%92%E8%81%94%E7%BD%91)。

2．搜索有[文件共享](https://baike.baidu.com/item/%E6%96%87%E4%BB%B6%E5%85%B1%E4%BA%AB)主机或者个人电脑(一般有专门的FTP服务器网站上公布的,上面有进入该主机或个人电脑的名称，口令和路径).

3．当与远程主机或者对方的个人电脑建立连接后,用对方提供的用户名和口令登陆到该主机或对方的个人电脑。

4．在远程主机或对方的个人电脑登陆成功后，就可以上传你想跟别人分享的东西或者下载别人授权共享的东西(这里的东西是指能放到电脑里去又能在显示屏上看到的东西).

5．完成工作后关闭FTP[下载软件](https://baike.baidu.com/item/%E4%B8%8B%E8%BD%BD%E8%BD%AF%E4%BB%B6),切断连接。

#### 2.1.2传输方式

FTP的传输有两种方式：ASCII、二进制。

**ASCII传输方式**

假定用户正在拷贝的文件包含的简单ASCII码文本，如果在远程机器上运行的不是UNIX，当文件传输时ftp通常会自动地调整文件的内容以便于把文件解释成另外那台计算机存储文本文件的格式。

但是常常有这样的情况，用户正在传输的文件包含的不是文本文件，它们可能是程序，数据库，字处理文件或者[压缩文件](https://baike.baidu.com/item/%E5%8E%8B%E7%BC%A9%E6%96%87%E4%BB%B6)。在拷贝任何非文本文件之前，用binary 命令告诉ftp逐字拷贝。

**二进制传输模式**

在二进制传输中，保存文件的位序，以便原始和拷贝的是逐位一一对应的。即使目的地机器上包含位序列的文件是没意义的。例如，macintosh以二进制方式传送[可执行文件](https://baike.baidu.com/item/%E5%8F%AF%E6%89%A7%E8%A1%8C%E6%96%87%E4%BB%B6)到Windows系统，在对方系统上，此文件不能执行。

如在ASCII方式下传输[二进制文件](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6%E6%96%87%E4%BB%B6)，即使不需要也仍会转译。这会损坏数据。（ASCII方式一般假设每一字符的第一有效位无意义，因为ASCII字符组合不使用它。如果传输二进制文件，所有的位都是重要的。）

#### 2.1.3命令端口

一般来说，客户端有一个 Socket 用来连接 FTP 服务器的相关端口，它负责 FTP 命令的发送和接收返回的响应信息。一些操作如“登录”、“改变目录”、“删除文件”，依靠这个连接发送命令就可完成。

#### 2.1.4数据端口

对于有数据传输的操作，主要是显示目录列表，上传、下载文件，我们需要依靠另一个 Socket来完成。

如果使用被动模式，通常服务器端会返回一个端口号。客户端需要用另开一个 Socket 来连接这个端口，然后我们可根据操作来发送命令，数据会通过新开的一个端口传输。

如果使用主动模式，通常客户端会发送一个端口号给服务器端，并在这个端口监听。服务器需要连接到客户端开启的这个数据端口，并进行数据的传输。

下面对 FTP 的主动模式和被动模式做一个简单的介绍。

#### 2.1.5主动模式 (PORT)

主动模式下，客户端随机打开一个大于 1024 的端口向服务器的命令端口 P，即 21 端口，发起连接，同时开放N +1 端口监听，并向服务器发出 “port N+1” 命令，由服务器从它自己的数据端口 (20) 主动连接到客户端指定的数据端口 (N+1)。

FTP 的客户端只是告诉服务器自己的端口号，让服务器来连接客户端指定的端口。对于客户端的防火墙来说，这是从外部到内部的连接，可能会被阻塞。

#### 2.1.6被动模式 (PASV)

为了解决服务器发起到客户的连接问题，有了另一种 FTP 连接方式，即被动方式。命令连接和数据连接都由客户端发起，这样就解决了从服务器到客户端的数据端口的连接被防火墙过滤的问题。

被动模式下，当开启一个 FTP 连接时，客户端打开两个任意的本地端口 (N > 1024 和 N+1) 。

第一个端口连接服务器的 21 端口，提交 PASV 命令。然后，服务器会开启一个任意的端口 (P > 1024 )，返回如“227 entering passive mode (127,0,0,1,4,18)”。 它返回了 227 开头的信息，在括号中有以逗号隔开的六个数字，前四个指服务器的地址，最后两个，将倒数第二个乘 256 再加上最后一个数字，这就是 FTP 服务器开放的用来进行数据传输的端口。如得到 227 entering passive mode (h1,h2,h3,h4,p1,p2)，那么端口号是 p1\*256+p2，ip 地址为h1.h2.h3.h4。这意味着在服务器上有一个端口被开放。客户端收到命令取得端口号之后, 会通过 N+1 号端口连接服务器的端口 P，然后在两个端口之间进行数据传输。

#### 2.1.7主要用到的 FTP 命令

FTP 每个命令都有 3 到 4 个字母组成，命令后面跟参数，用空格分开。每个命令都以 "\r\n"结束。

要下载或上传一个文件，首先要登入 FTP 服务器，然后发送命令，最后退出。这个过程中，主要用到的命令有 USER、PASS、SIZE、REST、CWD、RETR、PASV、PORT、QUIT。

USER: 指定用户名。通常是控制连接后第一个发出的命令。“USER drz\r\n”： 用户名为drz登录。

PASS: 指定用户密码。该命令紧跟 USER 命令后。“PASS 123456\r\n”：密码为 123456。

SIZE: 从服务器上返回指定文件的大小。“SIZE file.txt\r\n”：如果 file.txt 文件存在，则返回该文件的大小。

PASV: 让服务器在数据端口监听，进入被动模式。如：“PASV\r\n”。

PORT: 告诉 FTP 服务器客户端监听的端口号，让 FTP 服务器采用主动模式连接客户端。如：“PORT h1,h2,h3,h4,p1,p2”。

RETR: 下载文件。“RETR file.txt \r\n”：下载文件 file.txt。

STOR: 上传文件。“STOR file.txt\r\n”：上传文件 file.txt。

QUIT: 关闭与服务器的连接。

#### 2.1.8 FTP 响应码

客户端发送 FTP 命令后，服务器返回响应码。

响应码用三位数字编码表示：

第一个数字给出了命令状态的一般性指示，比如响应成功、失败或不完整。

第二个数字是响应类型的分类，如 2 代表跟连接有关的响应，3 代表用户认证。

第三个数字提供了更加详细的信息。

第一个数字的含义如下：

1 表示服务器正确接收信息，还未处理。

2 表示服务器已经正确处理信息。

3 表示服务器正确接收信息，正在处理。

4 表示信息暂时错误。

5 表示信息永久错误。

第二个数字的含义如下：

0 表示语法。

1 表示系统状态和信息。

2 表示连接状态。

3 表示与用户认证有关的信息。

4 表示未定义。

5 表示与文件系统有关的信息。

### 2.2 socket编程概述

#### 2.2.1 概述

在计算机通信领域，socket 被翻译为“套接字”，它是计算机之间进行通信的一种约定或一种方式。通过 socket 这种约定，一台计算机可以接收其他计算机的数据，也可以向其他计算机发送数据。

socket 的典型应用就是 Web 服务器和浏览器：浏览器获取用户输入的URL，向服务器发起请求，服务器分析接收到的URL，将对应的网页内容返回给浏览器，浏览器再经过解析和渲染，就将文字、图片、视频等元素呈现给用户。

#### 2.2.2 主要命令

以下对命令的介绍包括其描述、参数以及返回值

##### 2.2.2.1 创建

**描述**

1. resource socket\_create （int $domain ，int $type ，int $protocol ）

创建并返回套接字资源，也称为通信端点。典型的网络连接由2个套接字组成，一个执行客户端角色，另一个执行服务器的角色。

**参数：**domain type protocol

**domain**

该domain参数指定套接字使用的协议族。

| 可用的地址/协议族 | |
| --- | --- |
| 域 | 描述 |
| AF\_INET | 基于IPv4 Internet协议。TCP和UDP是该协议族的常见协议。 |
| AF\_INET6 | IPv6基于Internet的协议。TCP和UDP是该协议族的常见协议。 |
| AF\_UNIX | 本地通讯协议族 高效率和低开销使它成为一种很好的IPC（进程间通信）形式。 |

**type**

该type参数选择套接字使用的通信类型。

| 可用套接字类型 | |
| --- | --- |
| 类型 | 描述 |
| SOCK\_STREAM | 提供顺序，可靠，全双工，基于连接的字节流。可以支持带外数据传输机制。TCP协议基于此套接字类型。 |
| SOCK\_DGRAM | 支持数据报（无连接，固定最大长度不可靠的消息）。UDP协议基于此套接字类型。 |
| SOCK\_SEQPACKET | 为固定最大长度的数据报提供顺序可靠的基于双向连接的数据传输路径; 每个读取呼叫都需要消费者读取整个数据包。 |
| SOCK\_RAW | 提供原始网络协议访问。这种特殊类型的套接字可以用来手动构造任何类型的协议。此套接字类型的常见用途是执行ICMP请求（如ping）。 |
| SOCK\_RDM | 提供不保证订购的可靠数据报层。这很可能在您的操作系统上未实现。 |

**protocol**

该protocol参数设置domain在返回的套接字上进行通信时指定的特定协议。可以使用[getprotobyname（）](http://php.net/manual/en/function.getprotobyname.php)通过名称检索正确的值。如果希望的协议是TCP，或者UDP是相应的常量 SOL\_TCP，SOL\_UDP 也可以使用。

| 常用协议 | |
| --- | --- |
| 名称 | 描述 |
| ICMP | 互联网控制消息协议主要由网关和主机用来报告数据报通信中的错误。“ping”命令（在大多数现代操作系统中存在）是ICMP的示例应用程序。 |
| UDP | 用户数据报协议是具有固定记录长度的无连接，不可靠的协议。由于这些方面，UDP需要最小量的协议开销。 |
| TCP | 传输控制协议是一种可靠的基于连接的面向流的全双工协议。TCP保证按照发送顺序接收所有数据包。如果任何数据包在通信过程中丢失，TCP将自动重传数据包，直到目标主机确认该数据包。由于可靠性和性能原因，TCP实现本身决定了底层数据报通信层的适当八位字节边界。因此，TCP应用程序必须允许部分记录传输的可能性。 |

**返回值**

socket\_create（）在成功或FALSE错误时返回套接字资源。实际的错误代码可以通过调用[socket\_last\_error（）](http://php.net/manual/en/function.socket-last-error.php)来检索 。此错误代码可能会传递给 [socket\_strerror（）](http://php.net/manual/en/function.socket-strerror.php)以获取错误的文本说明

##### 2.2.2.2 绑定

**描述**

1. **bool** socket\_bind （resource $socket ，string $address [，**int** $port= 0 ]）

将名称给定义address到由socket描述的套接字socket。这必须在使用[socket\_connect（）](http://php.net/manual/en/function.socket-connect.php) 或[socket\_listen（）](http://php.net/manual/en/function.socket-listen.php)建立连接之前完成。

**参数：socket address port （可选的）**

**socket**

使用[socket\_create（）](http://php.net/manual/en/function.socket-create.php)创建的有效套接字资源。

**address**

如果套接字是AF\_INET家族，则 address是以虚线符号表示的IP（例如127.0.0.1）。

如果套接字是AF\_UNIX系列，则 address是Unix域套接字的路径（例如/tmp/my.sock）。

**port （可选的）**

该port参数仅在绑定AF\_INET套接字时使用，并指定要在其上侦听连接的端口。

**返回值**

返回TRUE成功或FALSE失败。

可以使用[socket\_last\_error（）](http://php.net/manual/en/function.socket-last-error.php)检索错误代码。该代码可能会传递给[socket\_strerror（）](http://php.net/manual/en/function.socket-strerror.php)以获取错误的文本解释。

##### 2.2.2.3连接

**描述**

1. **bool** socket\_connect （resource $socket ，string $address [，**int** $port= 0 ]）

**参数：socket address port**

**socket**

**address**

该address参数是在点分表示法表示的IPv4地址（例如127.0.0.1）如果 socket是AF\_INET，有效的IPv6地址（例如:: 1）如果启用IPv6支持和socket是AF\_INET6 或Unix域套接字的路径名，如果套接字家庭是 AF\_UNIX。

**port**

该port参数仅用于连接到一个AF\_INET或一个 AF\_INET6套接字时必须使用，并指定要在其上进行连接的远程主机上的端口。

**返回值**

返回TRUE成功或FALSE失败。可以使用[socket\_last\_error（）](http://php.net/manual/en/function.socket-last-error.php)检索错误代码 。该代码可能会传递给 [socket\_strerror（）](http://php.net/manual/en/function.socket-strerror.php)以获取错误的文本解释。

##### 2.2.2.4监听

**描述**

1. **bool** socket\_listen ( resource $socket [, **int** $backlog = 0 ] )

socket\_create( ) 函数创建一个套接字，然后通过 socket\_bing( ) 函数进行绑定刚创建的套接字，通过此函数进行监听！

socket\_listen( ) 仅限用于 SOCK\_STREAM（流式套接字）或者 SOCK\_SEQPACKET（通常用于非网络协议，例如X.25，或是广播协议AX.25）;

**参数：scoket backlog**

**scoket**

通过scoket\_create( ) 函数创建的 socket;

**backlog**

最多积压的传入连接将排队等待处理。如果一个连接请求到达队列的完整的客户端可能会收到一个错误的econnrefused，指示或如果底层协议支持重传请求可能被忽视，重试可能会成功。

注：  
传递给待定参数的最大数量取决于底层平台。在Linux上，它默默地截断somaxconn。在Win32中，如果通过somaxconn，基础服务程序 socket 将积压到最大化的合理值。在这个平台上没有标准的规定来查找实际的积压值。

**返回值**

成功时返回 TRUE， 或者在失败时返回 FALSE。错误代码可以检索与socket\_last\_error()。这个代码可以通过socket\_strerror()得到错误的文本解释。

#### 2.2.3 客户端编程模式

以下主要介绍几种Socket编程的模式。主要包括基本的阻塞Socket、非阻塞Socket、I/O多路复用。其中，阻塞和非阻塞是相对于套接字来说的，而其他的模式本质上来说是基于Socket的并发模式。I/O多路复用又包括了linux和windows下的这两种常见模式。

**阻塞模式**

阻塞模式是最基本的Socket编程模式，在各种关于网络编程的书籍中都是入门的例子。就像其名所说，阻塞模式的Socket会阻塞当前的线程，直到结果返回，否则会一直等待。

**非阻塞模式**

非阻塞模式是相对阻塞模式来说，Socket并不会阻塞当前线程，非阻塞模式不会等到结果返回，而会立即运行下去。

这里需要注意，阻塞/非阻塞、同步/异步之前的区别。在本质上它们是不同的。同步和异步是相对操作结果来说，会不会等待结果结果返回。而阻塞和非阻塞是相对线程是否被阻塞来说的。其实，这两者存在本质的区别，它们的修饰对象是不同的。阻塞和非阻塞是指进程访问的数据如果尚未就绪，进程是否需要等待，简单说这相当于函数内部的实现区别，也就是未就绪时是直接返回还是等待就绪。而同步和异步是指访问数据的机制,同步一般指主动请求并等待I/O操作完毕的方式,当数据就绪后在读写的时候必须阻塞,异步则指主动请求数据后便可以继续处理其它任务,随后等待I/O,操作完毕的通知,这可以使进程在数据读写时也不阻塞。因为两者在表现上经常相同，所以经常被混淆。

**I/O多路复用**

I/O多路复用是一种并发服务器开发技术(处理多个客户端的连接)。通过该技术，系统内核缓冲I/O数据，当某个I/O准备好后，系统通知应用程序该I/O可读或可写，这样应用程序可以马上完成相应的I/O操作，而不需要等待系统完成相应I/O操作，从而应用程序不必因等待I/O操作而阻塞。  
在linux下主要有select、poll、epoll三种模型，在freeBSD下则有kqueue，windwos下select、事件选择模型、重叠I/O和完成端口等。

### 2.3 Qt GUI框架概述

#### 2.3.1 Qt简介

Qt是一个跨平台的C++应用程序开发框架。广泛应用于开发GUI程序，这种情况下又被称为部件工具箱(widget toolkits)。也可用于开发非GUI程序，比如控制台工具和服务器。它是Digia公司的产品。Qt使用标准的C++和特殊的代码生成扩展(成为元对象编译器(Meta Object Compiler，moc))以及一些宏。通过语言绑定，其他的编程语言也可以使用Qt。

Qt是自由且开放源代码的软件，在GNU宽通用公共许可证(LGPL)条款下发布。所有版本都支持广泛的编译器，包括GCC的C++编译器和Visual Studio。

#### 2.3.2 Qt图形用户界面

Qt的图形用户界面的基础是QWidget。Qt中所有类型的GUI组件如按钮、标签、工具栏等都派生自QWidget，而QWidget本身则为QObject的子类。Widget负责接收鼠标，键盘和来自窗口系统的其他事件，并描绘了自身显示在屏幕上。每一个GUI组件都是一个widget，widget还可以作为容器，在其内包含其他Widget。

QWidget不是一个抽象类别。兵器可以被放置在一个已存在的用户界面中；若是Widget没有指定父Widget，当他显示时就是一个独立的视窗、或是一个顶层widget。QWidget显示能力包含了透明化及Double-Buffering。Qt提供一种托管机制，当Widget于创建时指定父对象，就可把自己的生命周期交给上层对象管理，当上层对象被释放时，自己也被释放。确保对象不再使用时都会被删除。

#### 2.3.3 信号与槽机制

Qt利用信号与槽(signals/slots)机制取代传统的callback来进行对象之间的沟通。当操作事件发生的时候，对象会提交一个信号(signal)；而槽(slot)则是一个函数，接受特定信号并且运行槽本身设置的动作。信号与槽之间，则透过QObject的静态方法connect来链接。

信号在任何运行点上皆可发射，甚至可以在槽里再发射另一个信号，信号与槽的链接不限定为一对一的链接，一个信号可以链接到多个槽或多个信号链接到同一个槽，甚至信号也可链接到信号。

以往的callback缺乏类型安全，再调用处理函数时，无法确定传递的是正确类型的参数。但信号和其接受的槽之间传递的数据类型必须要相匹配，否则编译器会提出警告。信号和槽可接受任何数量、任何类型的参数，所以信号与槽机制是完全类型安全。

信号与槽机制也确保了低耦合性，发送信号的类别并不知道是哪个槽会接受，也就是说一个信号可以调用所有可用的槽。此机制会确保当在"连接"信号和槽时，槽会接受信号的参数并且正确运行。

## 3. 实验设计

### 3.1 FTPTools库

#### 3.1.1 概述

该类使用WINSOCK2.H 以及 WS2tcpip.h 头文件，调用如上2.2描述的socket接口，完成了基于FTP的上传，下载，文件列表自动同步，登入登出等功能，实现了FTP的基础接口功能如下：

|  |
| --- |
| void CreateSocketAddr\_in(struct sockaddr\_in \* addr,const string &ipaddr,int port);  bool ConnectServer(SOCKET s, const struct sockaddr\_in \*name);  int LogIn(SOCKET s, const string &username, const string &password);  void DownloadFile(SOCKET s,const string &filename,const string &dir);  void ListFile(SOCKET s,string &filelist);  void UploadFile(SOCKET s, string &file); |
|  |

同时，有如下私有方法用于实现上述接口：

|  |
| --- |
| void SendCommand(SOCKET s,const string &command);  bool WSA();  void CloseSocket(SOCKET s);  SOCKET PassiveMode(SOCKET s); |
|  |

调用该工具库，只需要包含该库的头文件FTPTools.h，即可调用公用接口进行FTP编程。

#### 3.1.2 外部接口介绍

##### 3.1.2.1 CreateSocketAddr\_in 接口

**接口实现**：该接口将socket地址、客户端IP地址以及端口号封装在一起。

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | addr | sockaddr\_in \* | Socket地址 |
| 2 | ipaddr | const string & | 服务端IP地址 |
| 3 | port | int | 端口号 |

代码如下：

|  |
| --- |
| void FtpTools::CreateSocketAddr\_in(struct sockaddr\_in \* addr,const string &ipaddr,int port)  {  memset(addr, 0, sizeof(struct sockaddr\_in));  struct in\_addr s;  inet\_pton(AF\_INET,ipaddr.c\_str(), (void \*)&s);  (\*addr).sin\_addr.s\_addr = s.s\_addr;  (\*addr).sin\_family = AF\_INET;  (\*addr).sin\_port = htons(port);  return;  } |

##### 3.1.2.2 ConnectServer接口

接口实现：与服务器通过socket取得连接

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |
| 2 | name | **const struct sockaddr\_in \*** | Socket名 |
| 3 | port | int | 端口号 |

代码如下：

|  |
| --- |
| bool FtpTools::ConnectServer(SOCKET s,const struct sockaddr\_in \* name)  {  if (connect(s, (struct sockaddr \*)name, sizeof(\*name)) == SOCKET\_ERROR) {  return false;  }  return true;  } |

##### 3.1.2.3 LogIn接口

接口实现：登录函数，传递用户名以及用户密码

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |
| 2 | username | const string & | 从界面中读取的用户名 |
| 3 | password | const string & | 从界面中读取的密码 |

代码如下：

|  |
| --- |
| int FtpTools::LogIn(SOCKET s, const string &username, const string &password)  {  this->SendCommand(s, "USER "+username+"\r\n");    char readbuffer[3];  recv(s, readbuffer, 3, 0);  if (readbuffer[0]!='3'||readbuffer[1]!='3'||readbuffer[2]!='1') {  return -2;  }  this->SendCommand(s, "PASS "+password+"\r\n");  memset(readbuffer, 0, 3);  recv(s, readbuffer, 3, 0);  if (readbuffer[0] != '2' || readbuffer[1] != '3' || readbuffer[2] != '0') {  return -1;  }  return 1;  } |

##### 3.1.2.4 SendCommand接口

接口实现：清空当前接收buffer，预备接收来自服务器的消息，并向服务器端命令端口发送命令请求

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |
| 2 | command | const string & | 向命令端口发送命令 |

代码如下：

|  |
| --- |
| void FtpTools::SendCommand(SOCKET s, const string &command)  {  FlushBuffer(s);  send(s, command.c\_str(), command.size() + 1, 0);  return;  } |

##### 3.1.2.5 DownloadFile接口

接口实现：发送SendCommand请求,需要下载的读缓冲区的次数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |
| 2 | filename | const string & | 需要下载的文件名 |
| 3 | dir | const string & | 文件路径 |

代码如下：

|  |
| --- |
| void FtpTools::DownloadFile(SOCKET s, const string & filename, const string & dir)  {    SendCommand(s, "SIZE " + filename+"\r\n");  char readbuffer[50];  recv(s, readbuffer, 50, 0);  string size = readbuffer;  string size\_sub = size.substr(4,size.size()-6);  int filesize\_int = std::stoi(size\_sub);  std::ofstream mcfile;  mcfile.open(dir+filename);  SOCKET data\_s = PassiveMode(s);  SendCommand(s, "RETR " + filename + "\r\n");  int max = filesize\_int/500+1;  char recvbuffer[500];  for (int i = 0; i < max; i++) {  recv(data\_s, recvbuffer, 500, 0);  mcfile << recvbuffer;  }  mcfile.close();  } |

##### 3.1.2.6 ListFile接口

接口实现：列出当前文件

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |
| 2 | filelist | const string & | 文件列表 |

代码如下：

|  |
| --- |
| void FtpTools::ListFile(SOCKET s,string &filelist)  {  SOCKET data\_s = PassiveMode(s);  SendCommand(s, "NLST\r\n");  char readbuffer[1000];  while (true)  {  int ret = recv(data\_s, readbuffer, 1000, 0);  filelist += readbuffer;  if (ret < 1000)  {  break;  }  }  } |

##### 3.1.2.7 UploadFile接口

接口实现：从客户端向服务器端上传文件

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |
| 2 | file | const string & | 需要上传的文件的文件名 |

代码如下：

|  |
| --- |
| void FtpTools::UploadFile(SOCKET s, string & file)  {  std::ifstream mcfile;  mcfile.open(file);  std::stringstream ss;  ss << mcfile.rdbuf();  string str(ss.str());  SOCKET data\_s = PassiveMode(s);  string filename = file.substr(file.find\_last\_of('/')+1);  SendCommand(s, "STOR " + filename + "\r\n");  send(data\_s,str.c\_str(),str.size()+1, 0);  mcfile.close();    char readbuffer[50];  recv(s, readbuffer, 50, 0);    } |

##### 3.1.2.8 PassiveMode接口

接口实现：启用被动模式

|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |

代码如下：

|  |
| --- |
| SOCKET FtpTools::PassiveMode(SOCKET s)  {  ip\_port\* ipp = new ip\_port;  this->SendCommand(s, "PASV");  char readbuffer2[100];  recv(s, readbuffer2, 100, 0);  if (readbuffer2[0] != '2' || readbuffer2[1] != '2' || readbuffer2[2] != '7') {  return 0;  }  string port1, port2;  string ip(readbuffer2);  int count = 0;  for (int i = 0; i < 100; i++) {  if (readbuffer2[i] == ',') {  count++;  }  if (count == 4) {  i++;  ip = ip.substr(27, i - 28);  for (; readbuffer2[i] != ','; i++) {  port1 += readbuffer2[i];  }  i++;  for (; readbuffer2[i] != ')'; i++) {  port2 += readbuffer2[i];  }  break;  }  }  std::replace(ip.begin(), ip.end(), ',', '.');  ipp->ip = ip;  ipp->port = std::stoi(port1) \* 256 + std::stoi(port2);    SOCKET data\_s;  data\_s = socket(AF\_INET, SOCK\_STREAM, 0);  struct sockaddr\_in data\_addr;  CreateSocketAddr\_in(&data\_addr, ipp->ip, ipp->port);  if (ConnectServer(data\_s, &data\_addr) == 0) {  //QMessageBox::critical(0, "critical message", "Connect fail!", QMessageBox::Default, 0, 0);  closesocket(data\_s);  return 0;  }  delete ipp;  return data\_s;    } |

##### 3.1.2.9 FlushBuffer接口

接口实现：清空buffer，在之前的函数中作为被调用函数出现（buffer包括接收和发送两种）

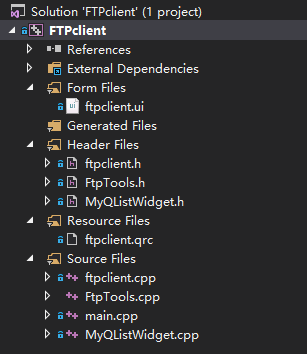
|  |  |  |  |
| --- | --- | --- | --- |
| **参数序号** | **参数名称** | **参数类型** | **参数说明** |
| 1 | s | SOCKET | Socket 套接字 |

代码如下：

|  |
| --- |
| void FtpTools::FlushBuffer(SOCKET s)  {  fd\_set recvfd;  struct timeval tv = { 3, 0 };  FD\_ZERO(&recvfd);  FD\_SET(s, &recvfd);  int result = select(0, &recvfd, NULL, NULL, &tv);  char readbuffer[500];  if (result > 0) {  while (true)  {  int ret = recv(s, readbuffer, 500, 0);  if (ret < 500)  {  break;  }  }  }  } |

### 3.2 FTPClient主体程序

#### 3.2.1 工程结构



其中，ftpclient.ui是QtDesigner根据我们的界面设计生成的xml文件，编译时uic工具会将ui文件的内容转换成C++代码，在工程目录同一级目录的 build- 目录下自动生成 ui\_类名.h 文件。

ftpclient.cpp与ftpclient.h实现了界面中各部件的功能。

FtpTools.cpp和FtpTools.h封装了WINSOCK2.H中有关ftp协议和socket通信的相关函数与数据结构。

MyQListWidget.cpp和MyQListWidget.h实现了继承自QListWidget的类的功能，用于展示文件列表。

#### 3.2.2 界面展示

本次实验中使用Qt设计了一个简单的图形化界面，如下所示。



标签后的四个单行文本编辑框均是类QLineEdit下的对象，依次命名为ip\_text、username\_text、password\_text、dir\_text，相应地用于获取域名、用户名、密码和从ftp服务器下载的文件的保存位置。

下面的大文本框是继承于QListWidget的MyQListWidget类的对象files，用于展示文件列表。这里也写了一个函数可以拖拽文件进行上传。

右下方三个按钮是QPushButton类的对象query\_button、download\_button和quit\_button。

#### 3.2.3 功能实现

##### 3.2.3.1登入及查询文件列表

单击“**查询**”按钮时，调用on\_query\_button\_clicked（）函数，客户端会根据ip\_text中的ip地址建立控制socket，尝试与服务器连接，若连接出错会报相应错误。之后获取服务器端的文件列表，将其展示在files框体中。

|  |  |
| --- | --- |
|  |  |
| 信号来源 | query\_button |
| 信号名 | clicked（） |
| 槽函数 | on\_query\_button\_clicked（） |
| 触发条件 | “**查询**”按钮被单击 |
| 调用接口（FTPTools） | CreateSocketAddr\_in()  ConnectServer()  LogIn()  ListFile() |

|  |
| --- |
| void FTPclient::on\_query\_button\_clicked() {  if (ftp.WSA() == 0) {  QMessageBox::critical(0, "critical message", "WSA fail!", QMessageBox::Default, 0, 0);  return;  }  //set control\_sock  control\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  struct sockaddr\_in server;  string server\_address = ui.ip\_text->text().toStdString();  ftp.CreateSocketAddr\_in(&server,server\_address,21);  //connect ftpserver  if (ftp.ConnectServer(control\_socket,&server)==0) {  QMessageBox::critical(0, "critical message", "Connect fail!", QMessageBox::Default, 0, 0);  closesocket(control\_socket);  return;  }    //log in  int ret = ftp.LogIn(control\_socket, ui.username\_text->text().toStdString(), ui.password\_text->text().toStdString());  switch(ret) {  case -2:  QMessageBox::critical(0, "critical message", "Username is wrong!", QMessageBox::Default, 0, 0);  closesocket(control\_socket);  return;  case -1:  QMessageBox::critical(0, "critical message", "Password is wrong!", QMessageBox::Default, 0, 0);  closesocket(control\_socket);  return;  case 0:  QMessageBox::critical(0, "critical message", "PASV failed!", QMessageBox::Default, 0, 0);  closesocket(control\_socket);  return;  case 1:  QMessageBox::information(0, "infomation", "Log in success!", QMessageBox::Default, 0, 0);  }  //get filelist  string filelist;  ftp.ListFile(control\_socket,filelist);  int filenum = count(filelist.begin(), filelist.end(), '\r');  ui.files->setViewMode(QListWidget::IconMode);  ui.files->setResizeMode(QListWidget::Adjust);  //ui.files->setMovement(QListWidget::Static);  string temp;  for (int i = 0; i < filenum; i++) {  QListWidgetItem \*item = new QListWidgetItem(ui.files);  QStyle::StandardPixmap sp = (QStyle::StandardPixmap)(25);  item->setData(Qt::DecorationRole, qApp->style()->standardPixmap(sp).scaled(QSize(16, 16), Qt::KeepAspectRatio, Qt::SmoothTransformation));  temp = filelist.substr(0, filelist.find("\r"));  filelist.erase(0, filelist.find("\n") + 1);  item->setData(Qt::DisplayRole, temp.c\_str());  }  ui.files->control\_socket = control\_socket;  ui.files->setDragDropMode(QAbstractItemView::DragDrop);  } |

##### 3.2.3.2下载文件

单击“**下载**”按钮时，调用on\_download\_button\_clicked()函数，客户端会下载文件列表中被选中的文件。

|  |  |
| --- | --- |
|  |  |
| 信号来源 | download\_button |
| 信号名 | clicked（） |
| 槽函数 | on\_download\_button\_clicked（） |
| 触发条件 | “**下载**”按钮被单击 |
| 调用接口（FTPTools） | DownloadFile() |

|  |
| --- |
| void FTPclient:: on\_download\_button\_clicked()  {  string filename = ui.files->currentItem()->text().toStdString();    ftp.DownloadFile(control\_socket, filename, ui.dir\_text->text().toStdString());  } |

##### 3.2.3.3登出

单击登出按钮时调用该函数，关闭控制socket，清空files中的文件列表，关闭files的拖动模式，清空密码文本框中内容，将下载按钮的enable值设为false。

|  |  |
| --- | --- |
|  |  |
| 信号来源 | quit\_button |
| 信号名 | clicked（） |
| 槽函数 | on\_quit\_button\_clicked（） |
| 触发条件 | “**登出**”按钮被单击 |
| 调用接口（FTPTools） | CloseSocket() |

|  |
| --- |
| void FTPclient::on\_quit\_button\_clicked()  {  string file = "dawtraw/awraw/awraw/dd.text";  string filename = file.substr(file.find\_last\_of('/')+1);  ftp.CloseSocket(control\_socket);  ui.files->clear();  ui.files->setDragDropMode(QAbstractItemView::NoDragDrop);  ui.password\_text->clear();  ui.download\_button->setEnabled(false);  } |

##### 3.2.3.4选中文件列表中的文件

单击files框中的文件图标时调用该函数，如果下载按钮的enable值是false，改变为true。

|  |  |
| --- | --- |
|  |  |
| 信号来源 | files |
| 信号名 | itemSelectionChanged() |
| 槽函数 | on\_files\_itemSelectionChanged() |
| 触发条件 | 文件列表中选中的文件变更 |
| 调用接口（FTPTools） | 无 |

|  |
| --- |
| void FTPclient::on\_files\_itemSelectionChanged()  {  if (ui.download\_button->isEnabled() == false) {  ui.download\_button->setEnabled(true);  }  } |

##### 3.2.3.5上传文件

files框体是我们自己定义的继承于QListWidget的MyQListWidget类的对象。它支持将客户端本地的文件拖拽至框体中（DnD）进行上传的操作。

|  |  |
| --- | --- |
|  |  |
| 信号来源 | files |
| 信号名 | QDropEvent() |
| 槽函数 | dropEvent() |
| 触发条件 | 将文件拖入文件展示列表中并释放 |
| 调用接口（FTPTools） | UploadFile() |

|  |
| --- |
| void FTPclient::dropEvent(QDropEvent \*event)  {  QString str = event->mimeData()->text();  string temp = str.toStdString().substr(str.toStdString().find\_last\_of('/')+1);  FtpTools ftp;  ftp.UploadFile(control\_socket,str.toStdString());  if (!str.isEmpty())  {  QListWidgetItem \*item = new QListWidgetItem(this);  QStyle::StandardPixmap sp = (QStyle::StandardPixmap)(25);  item->setData(Qt::DecorationRole, qApp->style()->standardPixmap(sp).scaled(QSize(16, 16), Qt::KeepAspectRatio, Qt::SmoothTransformation));  item->setData(Qt::DisplayRole, temp.c\_str());      event->accept();  }  } |

## 4.结论

### 4.1 实验收获

在这几天编写FTP服务器端和客户端程序时，总是遇到这样那样的错误和困难，现在已经解决大部分，几点收获记录如下： 

1.学会了许多FTP命令，与服务器通信的方式。基本掌握了Socket的工作原理和编程方式。

2. 明白了主动模式和被动模式的根本区别：   
主动模式：服务器向客户端敲门（请求打开端口），然后客户端开门（打开端口）；   
被动模式：客户端向服务器敲门（请求打开端口），然后服务器开门（打开端口）。   
  
3. Socket的recv是阻塞的，如果一直没有recv到信息的话，会使程序保持挂起状态，导致死机。解决方案是使用select()函数判断recv缓冲区是否为空，然后再去读取，这一点在FlushBuffer()內有体现。

4.Socket的recv缓冲区不会自动清空，为了保证下次发送指令后读取到的是正确的当次返回，需要在发送指令前先清空recv缓冲区，保证下次到来的数据直接被读取。  
  
5. 明白了一种协议标准的必要性。 通过在客户端和服务器端两端观察交互结果，可知完成FTP全都是建立在标准之上，本实验使用的FileZilla 0.9.41 beta作为一款互联网常用软件能够与本实验客户端无阻通讯全都建立在协议标准之上。

6. 学会使用Qt designer 进行界面设计，同时加强了C++编程技巧以及对VS2017的使用。

7. 掌握了图形化界面DnD机制的使用技巧，通过拖拽实现了更为方便的界面操控。

### 4.2 未来工作

1.完成对FTP服务端的设计，通过对服务端和客户端共同的设计，突破服务端的指令限制，完成一些更加人性化的设计，包括低成本重命名，在线修改等功能。

2.由于时间限制本次实验未完成的断点续传，同步文件等功能，可以继续实现，使客户端更加符合使用需求。

3．界面设计进一步完善，包括实现进度条显示下载上传进度，超时退出提醒等功能，同时对于错误判断的覆盖继续完善。

**参考文献**

[1] 《简单理解Socket》<https://www.cnblogs.com/dolphinX/p/3460545.html>；

[2] Qt维基百科界面 <https://zh.wikipedia.org/zh/Qt>

[3]**《使用 Socket 通信实现 FTP 客户端程序》**<https://www.ibm.com/developerworks/cn/linux/l-cn-socketftp/index.htm>

[4] 《网络编程实用教程》（叶树华等编著，人民邮电出版社）

本实验全部源代码放于 <https://github.com/17pigfarmer/FTPclient>。