



南開大學
Nankai University

南开大学

计算机学院

计算机网络实验报告

实验三 ——— 简单 FTP 协议实现

学号 : 1712991

姓名 : 周辰霏

年级 : 2017 级

专业 : 计算机科学与技术

2019 年 12 月 9 日

摘要

借助 QT 提供的 QUdpSocket 类实现了具有简单功能的 FTP 协议的服务器和客户端, 实现了多线程下的可靠传输文件, 包含断点续传、超时重传, 同时还辅以验证身份、展示服务器文件目录、新建文件目录、删除文件等功能。在代码实现的过程中更深的体会 UDP、TCP 以及 FTP 的核心内容。

关键字: FTP 协议; 多线程; C/S 编程; QT

目录

一、 实验目的	1
二、 实验环境	1
三、 FTP 工作流程 & 程序结构	1
(一) 工作流程	1
(二) 程序结构	2
四、 程序演示	8
五、 实验遇到问题及解决	13
六、 结论	14

一、实验目的

实现 FTP 协议

- 下层使用 UDP 协议(即使用数据报套接字完成本次程序)

- 完成客户端和服务器端程序

- 实现可靠的文件传输：能可靠下载文件，能同时下载文件

自定义附加功能

- 多线程实现

- 可靠传输文件上传或下载，断点续传

- 验证身份、新建文件夹、显示目录、删除文件、进入到新的文件夹、注册新用户

二、实验环境

- 系统环境:MacOS 10.15

- IDE:Qt Creator 4.9

- 编程语言:C++

三、FTP 工作流程 & 程序结构

(一) 工作流程

整个 FTP 协议服务器客户端是通过 UDP 协议仿照 TCP 协议实现的具有可靠文件传输和多用户功能，最终实现了一个具备握手连接、验证身份、断开连接、列目录、下载/上传文件、删除文件、新建目录以及断点续传等基本功能的简单 FTP 协议下的服务器和客户端。

下图是 FTP 协议总体工作流程的简化表示，可以看出基本是仿照 TCP 实现的，所以这之中也包含了断点续传和命令重传等可靠性传输的保证。

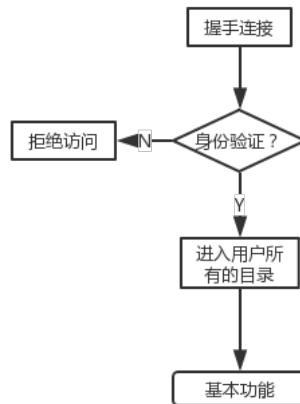


图 1: 总体工作流程

而当身份验证通过可以执行基本功能后，又分为两种情况：上传下载为一类(有文件传输需要可靠传输)，其它基础功能为一类(只需包含命令不达则重传)

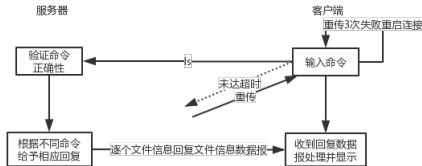


图 2: 其它基础功能,ls 为例

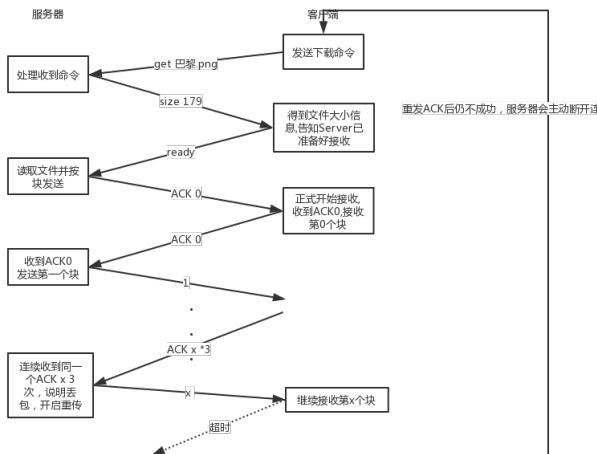


图 3: 上传下载功能,get 为例

(二) 程序结构

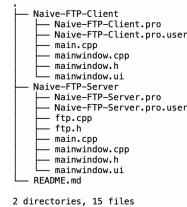


图 4: 程序文件

ui 文件包含了程序的 GUI 界面;pro 文件描述了整个 QT 程序，在其中加入 network 组件以便处理网络编程;main.cpp 中将 mainWindow 呈现出来;本次核心代码在 ftp.h 和 ftp.cpp 中实现，其中包含了全部 ftp 协议的命令处理，相应客户端内容在客户端 mainWindow.h 和 mainWindow.cpp 中，此外客户端 mainWindow 还包含客户端界面的处理;服务器 mainWindow 实现了对多用户分配不同线程以实现同时下载等多用户功能;最终实现了具备基础功能的 FTP 服务器与客户端。

1. Server 中 QUpdSocket 创建以及为每一个用户分配线程 在 mainWindow.h 以及 mainWindow.cpp 中

```

Initial
1 socket = new QUdpSocket();
2 bool success = socket->bind((ui->port->text()).toInt()); // 监听到指定端口
3 if(success)
4     ui->log->append("(Server) listening on "+ui->port->text());
5 else

```

```

6     QMessageBox::warning(this, tr("Warning"), tr("Listen to port failed!"));
7     connect(socket, &QUdpSocket::readyRead, this, &MainWindow::onNewDatagrams);
8 // 将 readyRead(即 socket 收到新数据报) 信号与 onNewDatagrams 连接
9 // onNewDatagrams 槽
10 void MainWindow::onNewDatagrams()
11 {
12 // 一旦有新的客户端发起连接数据报文就新建一个线程并且将 command 分到对应的线程
13     while(socket->hasPendingDatagrams()) {
14         auto datagram = socket->receiveDatagram(65535); // ≤65535 的数据报
15         const QString clientName = datagram.senderAddress().toString()
16         + "+" + QString::number(datagram.senderPort()); // 线程名
17         const auto dataTrimed = datagram.data().trimmed();
18         if(dataTrimed == "hello") // 收到的是建立请求的数据报
19         {
20             if(Clients.contains(clientName)) // 如果已有该线程，stop
21             {
22                 Clients[clientName]->stop();
23                 // Clients 定义 QMap<QString, FTP*> Clients;
24             }
25             Clients[clientName] = new FTP(datagram.senderAddress(),
26                                         static_cast<quint16>(datagram.senderPort()), socket);
27             auto thread = new QThread;
28             connect(thread, &QThread::started, Clients[clientName], &FTP::run);
29             connect(Clients[clientName], &FTP::finished, thread, &QThread::quit);
30             connect(Clients[clientName], &FTP::log, this, [=](QString log){
31                 ui->log->append("(" + clientName + ") " + log);
32             });
33             connect(thread, &QThread::finished, Clients[clientName],
34                     &FTP::deleteLater); // 下一次才删除，避免 delete 以后又调用出错_(:3)
35             Clients[clientName]->moveToThread(thread);
36             thread->start();
37         }
38     else { // 其他命令
39         if(Clients.contains(clientName))
40             // 将其转给 onCommand 处理命令队列内容
41             QMetaObject::invokeMethod(Clients[clientName], [=](){
42                 Clients[clientName]->onCommand(new QByteArray(datagram.data()));
43             }, Qt::QueuedConnection);
44     }
45     else
46         socket->writeDatagram("hello\n", datagram.senderAddress(),
47                               static_cast<quint16>(datagram.senderPort()));
48     }
49 }
50 }
```

2. 接收并处理请求 在 Server 中通过 OnCommand 处理从客户端收到的命令, Client 通过 On-Data 处理接收到的数据包在此仅列出 Client 部分, 二者原理相同

OnData——Client

```

1 void MainWindow::onData() { // 处理收到的数据报们
2     while (Socket->hasPendingDatagrams())
3     {
4         auto datagram = Socket->receiveDatagram(65535);
5         if (datagram.senderAddress().toIPv4Address() != host.toIPv4Address())
6             {// 未知来源数据报
7                 continue;
8             }
9         if(waitingConnectReply) // 等待服务器回复握手过程
10        {
11            if (datagram.senderPort() != port) // 端口不一致
12                continue;
13            auto data = datagram.data();
14            auto code = data.split(' ')[0];
15            if(code != "200") // 不是 success 的 200 码
16                return;
17            waitingConnectReply = false;
18            isConnected = true; // 连接状态
19            QMessageBox::about(this, tr("Tip"), tr("Connect To Server Success!"));
20            ui->textBrowser->append
21                ("connect to FTP server "+host.toString()+" succeded!");
22            accept();
23            // 接受这条回复数据报，将此次发送命令时设定的 timer 杀掉，无需重传命令
24        }
25    }
26    else
27    {
28        if (datagram.senderPort() != communicationPort) // 端口不一致
29            continue;
30        auto data = datagram.data();
31        if(data.simplified().mid(4,4)=="file"
32           || data.simplified().mid(4,3)=="dir") // 对于 ls 的回复报文处理
33        {
34            int begin=data.simplified().indexOf(" ");
35            int begin2=data.simplified().indexOf(" ",begin+1);
36            int begin3=data.simplified().indexOf("Byte",begin2+1);
37            ui->filelist->append(data.simplified().mid(4,4)+"
38                                "+"
39                                data.simplified().mid(begin,begin2-begin)+"
40                                "+"
41                                data.simplified().mid(begin2,begin3-begin2)+"
42                                "+data.simplified().mid(begin3+4));
43            // 显示在客户端窗口
44        }
45        if(lastTimerId != -1)
46            clearTimer(lastTimerId);
47            // 已经收到报文可以关掉这个计时器，去发送新的命令并开启下一个了

```

```

48
49     }
50     if(isSending)      // 发送之前
51     {
52         auto list = data.split(' ');
53         if(list.length() == 1 && list.at(0) == "ok")
54             {// 收到 Server 回复 ok 说明可以上传文件块
55                 isWritingData = true;
56                 isSending = false;
57                 onWriteData("ACK 0");
58             }
59         else {
60             isSending = false;
61             UnknownState(datagram);
62         }
63     }
64     else if(isReceiving) { // 接收之前发送 ready 告诉 Server 准备好
65         auto list = data.split(' ');
66         if(list.at(0) == "size")
67         {
68             bool ok = false;
69             qint64 size = list.at(1).toLongLong(&ok);
70             if(ok)
71             {
72                 totalBlock = size - 1;
73                 isWaitingData = true;
74                 isReceiving = false;
75                 sendCommand("ready");
76             }
77         }
78     }
79     else
80     {
81         isReceiving = false;
82         UnknownState(datagram);
83     }
84     else if(isWritingData) // 正在发送数据
85     {
86         onWriteData(data);
87     }
88     else if(isWaitingData) // 正在接收数据
89     {
90         onReceiveData(data);
91     }
92     else {
93         // 不显示过长的指令    比如数据传输的包
94         if(data.length() < 50)
95         {
96             ui->textBrowser->append(data.simplified());
97         }
98     }
99 }

```

```

96         }
97         accept(); // 清计时器，重试次数等 表示这次接收成功
98     }
99 }
100}
101}

```

3. 发送请求 在 Client 和 Server 中通过 sendCommand 以及 sendACK 发送数据报, 在此不赘述

sendCommand——Client

```

1 void MainWindow::on_pushButton_3_clicked() // 发送命令
2 {
3     if (!isConnected)
4         QMessageBox::warning(this, tr("Warning"),
5             tr("Please Connect First"));
6     if (!isAuth)
7         QMessageBox::warning(this, tr("Warning"),
8             tr("Please Auth Your Identity First"));
9     else
10    {
11        if (ui->Command->text() == "")
12            QMessageBox::warning(this, tr("Warning"),
13                tr("Please Enter Command"));
14        else if (ui->Command->text() == "quit") // 以 quit 为例, 其他见代码
15        {
16            QString str = "quit\n";
17            isConnected = false;
18            isAuth = false;
19            sendCommand(str.toLocal8Bit()); // SendCommand 发送数据报
20        }
21    }
22 }
23 void MainWindow::sendCommand(const QByteArray &array, bool retry) {
24     if (lastTimerId >= 0) // Server 也不计时命令是否传达
25     {
26         return;
27     }
28     if (retry) // Server 不需要重传命令就没有这一步
29     {
30         lastTimerId = this->startTimer(5000);
31         lastCommand = array;
32     }
33     Socket->writeDatagram(array, host, communicationPort);
34 }
35 // 如果超时, 计时器会调用处理事件, 重发最后一条命令
36 void MainWindow::timerEvent(QTimerEvent *event)
37 {
38     if (isWritingData || isWaitingData)

```

```

39     { // 超时以后还在接收或发送状态 即数据状态，由断点续传处理
40         onSendReceiveTimeout(event->timerId());
41         return;
42     }
43     if(lastTimerId == event->timerId()) // 重发命令
44     {
45         clearTimer(lastTimerId);
46         retryLastCommand(); // 重发最后一条命令
47     }
48 }
```

4. 上传下载文件 在 Client 和 Server 中通过 onReceiveData 来接收文件, 上传文件通过 onWriteData, 二者类似, 只举一例说明

onReceiveData——Client

```

1 void MainWindow::onReceiveData(const QByteArray &data)
2 { // 发送 get 命令以后接收文件
3     int begin = data.indexOf(" ");
4     bool ok = false;
5     int block = data.left(begin).toInt(&ok);
6     qDebug() << "? got block" << block;
7     if(ok && block == lastAck && block <= totalBlock)
8     { // 当前 ack 等于期望的 ack 且接收到的 block 小于总数，继续接收
9         if(sendTimerId >= 0) // 清除上一个计时器
10            clearTimer(sendTimerId);
11         dataTry = 0; // 重试次数
12         lastAck++; // 上一个 ack+1
13         if(tempFile.isWritable()) // 打开临时文件准备写入数据
14         {
15             tempFile.write(data.mid(begin + 1));
16         }
17         accept(); // 清计时器，重试次数等 表示这次接收成功
18         if(block == totalBlock) // 块数和 size 发送的块数相等，接收完毕
19         {
20             isWaitingData = false;
21             tempFile.close();
22             ui->textBrowser->append("t: file transfer finished");
23             sendCommand(QByteArray("ACK ") + QByteArray::number(block + 1),
24                         false);
25         }
26     } else
27     {
28         expecting--;
29         if(expecting == 0) // 一次性最多的 block 用完了就重置
30         {
31             expecting = 20;
32             sendCommand(QByteArray("ACK ") + QByteArray::number(lastAck),
33                         false);
34 }
```

```

34
35
36     sendTimerId = startTimer(5000); // 重新计时
37
38     else // Ack 对应不上说明超时了
39     {
40         onSendReceiveTimeout(-block); // 接收文件超时处理，见下
41     }
42 }
\\ 接收发送文件超时处理
44 void MainWindow::onSendReceiveTimeout(int timerId)
45 {
46     if(sendTimerId >= 0)
47         clearTimer(sendTimerId);
48     if(isWaitingData) // 仿TCP，还在等待接收的状态
49     {
50         if(dataTry == 2)// 已经重试过3次，重启 Client 连接
51         {
52             ui->textBrowser->append("t: file transfer failed");
53             reject();
54             return;
55         }
56         expecting = 20;
57         accept(); // 重发ACK
58         dataTry++;
59         sendCommand(QByteArray("ACK") + QByteArray::number(lastAck));
60         return;
61     }
62     ui->textBrowser->append("t: Send or Receive Timeout");
63 }

```

5. 其它之后再进行一些程序的界面和交互的完善即可,如加入错误提示框以及错误显示在日志记录上,以及 Server 日志框上记录 Server 和 Client 交互的全过程,较为简单,代码略。

四、 程序演示

- 工作界面：实现了实验要求的全部界面及功能, 下文依次介绍

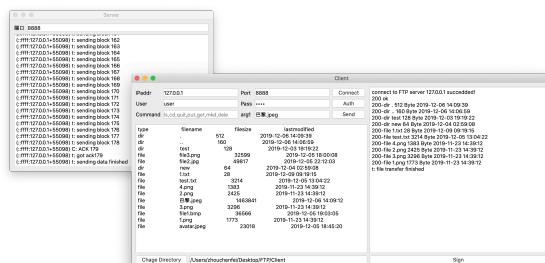


图 5: 工作界面

- 服务器可以切换监听端口：增加了服务器连接不成功,Socket 创建失败的提示

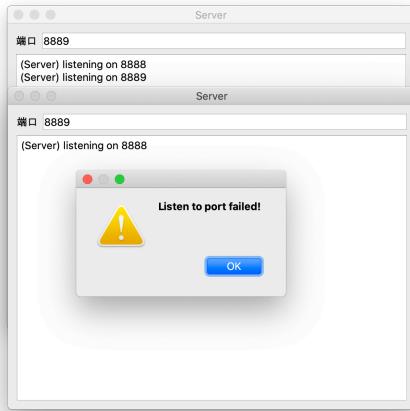


图 6: 服务器切换端口

- 客户端建立与 Server 之间的连接：

- 连接成功

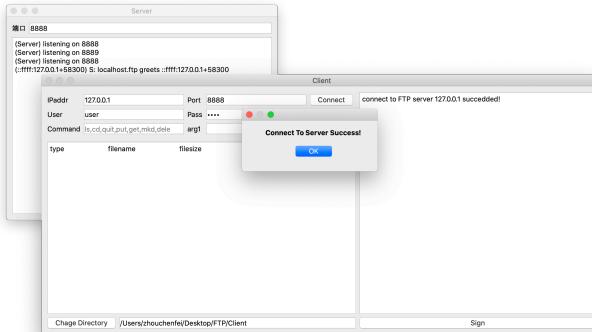


图 7: 与服务器建立连接成功

- 连接失败, 进行命令重传

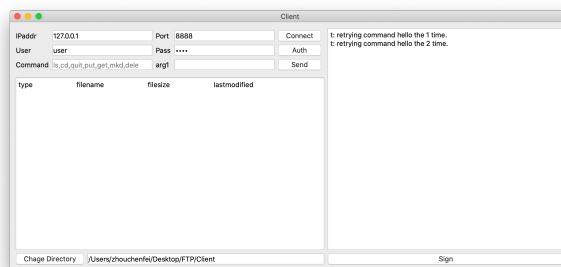


图 8: 与服务器建立连接失败——开始重传命令

- 验证身份: 图中展示了用户不存在、密码错误以及验证身份成功三种状态

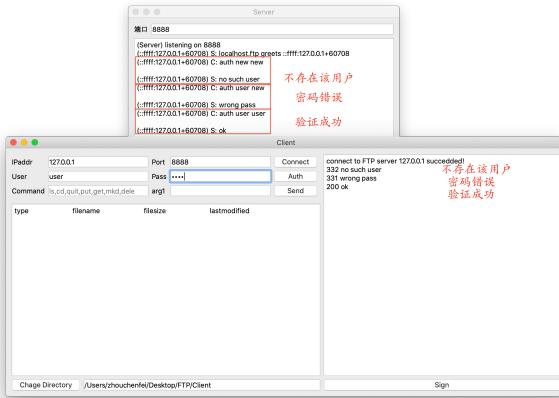


图 9: 验证身份

- 注册新用户: 在验证身份输入框输入用户名以及密码后按下“Sign”，图中展示了用户已存在、注册用户成功并通过注册用户登录成功三种状态

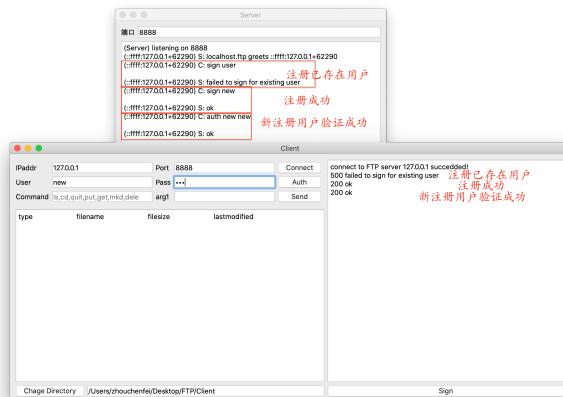


图 10: 注册用户

- 服务器文件目录展示 ls: 分别呈现: 类型 (文件夹或文件)、文件名、文件大小 (Byte)、上一次修改时间

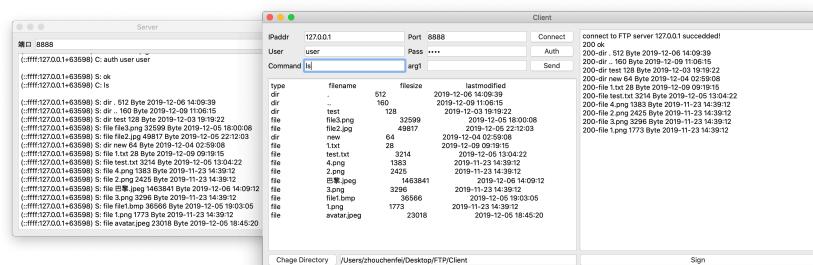


图 11: 文件目录展示

- 切换服务器目录 cd: 进入当前服务器目录下的新目录

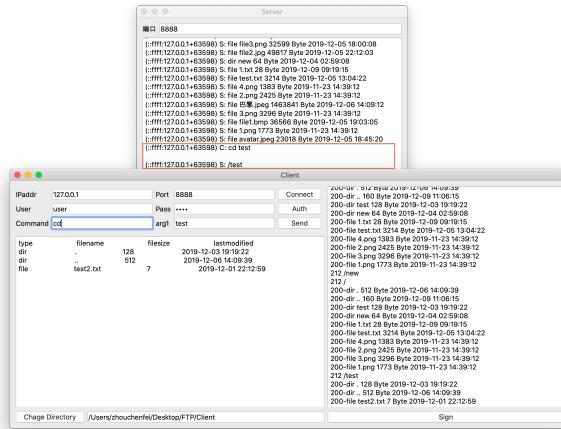


图 12: 切换服务器目录

- 新建文件夹 mkd: 在当前服务器目录新建文件目录

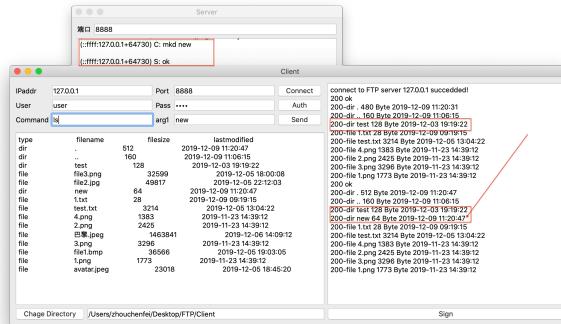


图 13: 新建文件夹

- 删除文件 delete: 在当前服务器目录删除文件

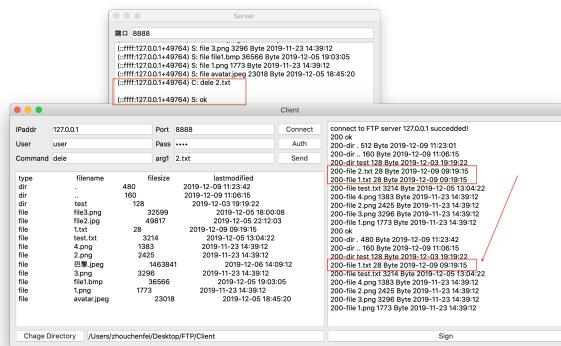


图 14: 删除文件

- 退出断开连接 quit: 退出并断开与服务器之间的连接



图 15: 退出登录

- 文件下载 get

- 同时下载文件: 可以看到图 16 中两个客户端的接收日志记录交替出现, 而最终也可以在指定客户端接收文件夹看到两个下载成功的文件(图 17)

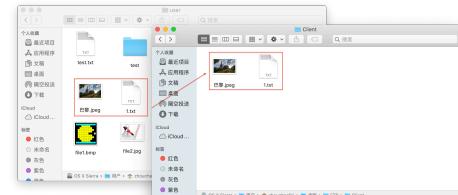
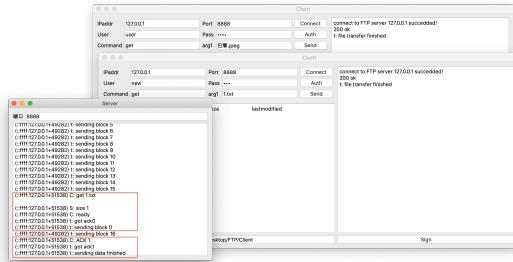


图 16: 同时下载文件过程

图 17: 下载成功示意图

- 丢包重传示意: 同时多个客户下载较大文件容易发生丢包需要重传。可以看到图中 Server 已经向客户端 58950 已经发送到第 79 个包, 但是由于收到了客户端回复的 ACK60, 所以又重新从第 60 个块开始发送, 这是一种可靠传输的机制, 即收到连续 ACK 则重传 ACK 指定的序号及其之后的所有块; 另一种情形是超时重传, 设定时间为 5s, 如果 5s 还未收到客户端发出的 ACK 信号, 也进行重传

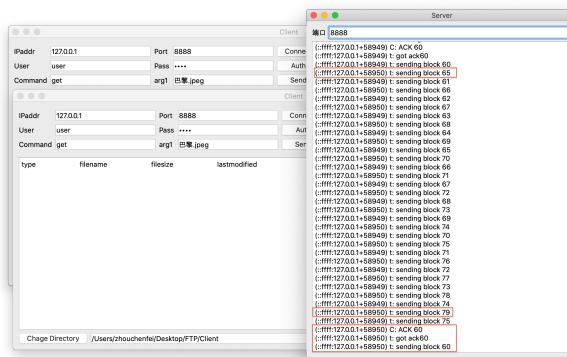


图 18: 可靠传输——丢包示意

• 文件上传 put: 文件上传与下载实现同理, 也是同样的可靠性传输机制, 所以就不赘述, 图中仅为简单的上传成功示意

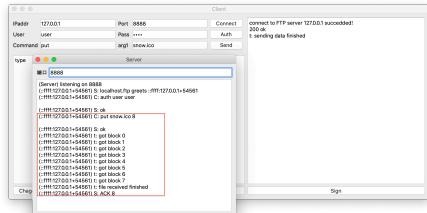


图 19: 上传文件示意

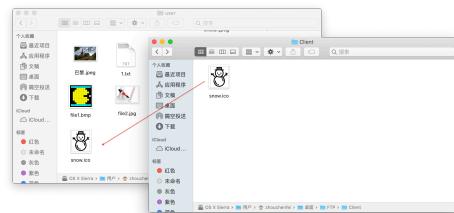


图 20: 上传成功示意图

五、 实验遇到问题及解决

- 起初只有文件大小小于一个块的文件可以正常传输,但这不符合我的预期呀,所以如下一点中解决
- 起初以为块的大小(数据类数据包)设定为小于等于 65507 即可(UDP 数据报数据部分长度),但是在编写的过程中,容易出现丢包且缓存根本写不上的问题,最后多番尝试选取了一个比较合适的大小——10000 字节,这样不容易丢包,而且缓存也更容易写入,不会导致超时这一问题;同时将一次写入缓存的块数相应调大,就可以很好的实现较大文件(10M 以下)的传输
- 同样是第一次上机作业出现过的传输数据报的转换问题——一开始并不可以识别包含中文的数据报,会报错。后来改成了 toLocal8Bit() 的转换方式就好了,顺便查询了下这几种转换字节流的方式的区别,toLatin1() 包含 ASCII,toUtf8() 包含 Utf-8 字符集,而 toLocal8Bit() 包含的是有 GBK-2312 在内的 Unicode,所以 toLocal8Bit() 可以正常表示中文
- 一开始并没有验证身份这一项,但是后来发现这是 FTP 不可或缺的一部分,所以取了个巧,即在 Server 根目录中创建与用户名相同的文件夹,如果用户名与密码一致,即可进入该文件夹,就表明验证成功,同理注册用户也是这么实现的,这样就避免了额外的不必要的数据库操作
- 最早的同时传输就是不同的登入以后分别像来源端口传输文件,这样不仅效率不高还非常容易丢包导致服务器主动断开连接,后来改用多线程实现,程序的层次变清晰了而且同时传输的文件之间也不会造成影响,很好的完成了实验要求
- 计时超时重传也是一开始没有的功能,是在 Client 需要重传命令的时候发现的,因为很有可能这个命令包根本没有送到服务器,所以不能通过验证 ACK 或其他的回复数据报的方式来重传命令,所以加入了计时器,并且在上传下载文件过程中也加入了计时器,最终就形成了比较完整的一个可靠传输

六、 结论

这是计算机网络最后一次上机作业,虽然要求很简单,但还是希望可以完成的比较好,可以涵盖一下多个层的知识和交互,虽然到后来也就还是只有传输层和应用层(其他我也涉及不到),但是最终仿照 TCP 用 UDP 实现了一个简单的 FTP 协议也是一件非常开心的事情,可以在本次作业上看出这一个学期从作业一到作业三的成长。整个作业的完成基本全是基于 Qt 官方类文档完成的,虽然中途挖坑无数,但是在加深了对 UDP 和 FTP 的理解的同时,代码实现的能力以及阅读文档能力也有了不少的长进。

参考文献

- [1] FTP Protocol[EB/OL].https://en.wikipedia.org/wiki/File_Transfer_Protocol
- [2] FTP 命令列表 [EB/OL].https://zh.wikipedia.org/wiki/FTP_命令列表
- [3] Qt 类官方文档 [EB/OL].<https://doc.qt.io/qt-5/qtcore-module.html>