南开大学
Nankai University

南　开　大　学

计 算 机 学 院

计算机网络实验报告

实验二————SMTP 邮箱服务器实现

学号：1712991

姓名：周辰霏

年级：2017 级

专业：计算机科学与技术

2019 年 11 月 1 日

摘要

通过 Wireshark 抓包分析邮件发送过程的数据交互细节, 深入理解邮件服务器的工作流程。再借助 CAsyncSocket 类实现一个简单的邮箱服务器, 更好的学习体会 SMTP 协议的内容, 并兼以习得 base64 编码相关知识。最终的服务器不仅有着完备的日志显示, 可以显示邮件的发件人收件人主题正文内容等, 同时进行多封邮件的传输、存储显示以及附件图片或 txt 文本的显示。

关键字：SMTP 协议；邮箱服务器；MFC；Wireshark 分析

目录

一、 实验目的

SMTP 和 POP3 协议是目前电子邮件应用系统中最重要的两个协议。深入理解 SMTP 和 POP3 协议的工作过程对理解整个电子邮件系统具有重要的意义。基于最基本的 SMTP 协议, 进行本次实验:

- 观察电子邮件应用程序与 SMTP 邮件服务器的命令交互过程
- 编程实现简化的 SMTP 邮箱服务器, 要求有以下功能:
  - 响应客户 SMTP 命令, 将命令的交互过程和收到的邮件显示到屏幕上
  - 支持单用户
  - 不保存和转发收到的邮件
  - 不作错误处理

二、 实验环境

- 系统环境:Windows 10 家庭中文版 (64 位)
- 测试用邮件客户端:Firefox 7.2.13
- WireShark:3.0.6
- IDE:Visual Studio 2019
- 调试环境:MFC Release x86
- 编程语言:C++

三、 实验原理

(一) 邮件服务器与客户端交互过程

利用 Winsock 实现本地 SMTP 服务器并实现与邮件客户端 (如 Firefox) 的交互

(二) SMTP 工作原理

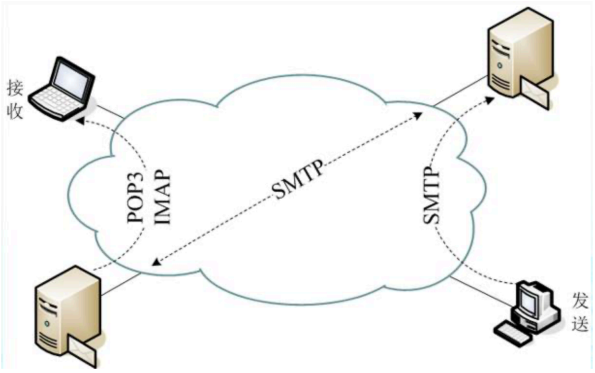

图 1: SMTP 工作流程

主机 1 作为发送主机, 目标为主机 2 上的邮箱账户, 整个邮件通过简化 SMTP 服务器发送的流程如下:

1. 主机 1 上的客户端发起连接请求, 使用 TCP 协议连接 SMTP 服务器 1 的 25 端口
2. 经过三次握手, 即此 SMTP 服务器 1 和主机 1 上的客户端连接完毕后, SMTP 服务器 1 会首先发送 220 ready 告知客户端服务器准备就绪, 客户端则会发送 “HELO” 或 “EHLO” 报文将自己的域地址告诉给 SMTP 服务器
3. 之后即是主机 1 上的客户端与 SMTP 服务器 1 之间命令和数据的交互, 程序命令响应表如下:

报文命令前几位	服务器响应	描述
HELO 或 EHLO	250 hello	客户端发送此命令与 SMTP 服务器建立连接, 将发送者域地址发送给 SMTP 服务器
MAIL FROM	250 mail from	客户端将邮件发送者的名称传送给 SMTP 服务器
RCPT TO	250 receiver	客户端将邮件接收者的名称传送给 SMTP 服务器
DATA	354 data	客户端将邮件报文内容传送给 SMTP 服务器 (此时并未接收到数据, 只是代表客户端即将发送数据, 当再次接受到客户端发送的信息时, 对数据进行处理并且回复)
	250 data	此时数据接收完毕
QUIT	221 succeed	SMTP 服务器关闭, 之后客户端将关闭对于 25 端口的 TCP 连接

表 1: 命令响应

4. 常用 SMTP 服务器响应返回值如下:

返回值	描述
211	系统状态或系统帮助应答
214	帮助信息
220<domain>	域服务就绪
221<domain>	服务关闭传输信道
250	请求的命令成功完成
354	可以发送邮件内容
500	语法错误, 命令不能识别 (包括命令行过长)
501	参数格式错误
502	命令未实现
503	错误的命令序列
504	命令参数不可实现
550	邮箱不可用
554	操作失败

表 2: 服务器响应返回值

5. 当 SMTP 服务器 1 回应 354 并且接收到数据后, 此时在 SMTP 服务器上对接收到的数据进行解码

6. 之后此 SMTP 与主机 2 上客户端预设的 SMTP 服务器 2 进行数据交互, 具体的邮件发送到主机 2 的客户端上则由 SMTP 服务器 2 负责发送交互

四、 SMTP 协议抓包分析————Wireshark

1. 首先通过 Foxmail 登录 QQ 邮箱, 无须更改任何初始设置
2. 之后编辑任意一封邮件, 不发送; 并打开 Wireshark, 将 WLAN 的 Filter 设置为 TCP only 之后开始捕获, 与此同时发送刚刚编辑好的邮件
3. 在提示邮件发送成功之后停止捕获, 可以看到 Wireshark 的结果如下

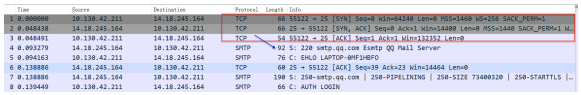

图 2: 连接成功

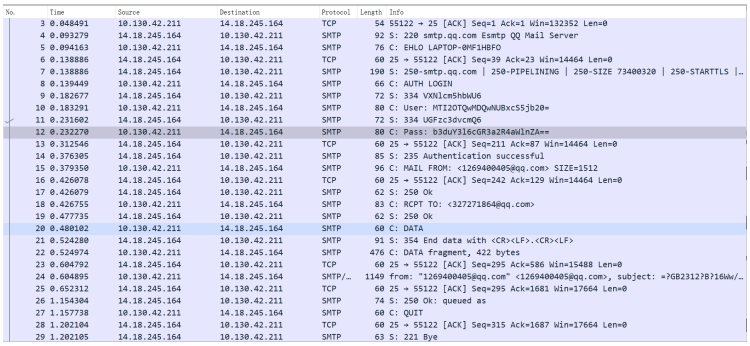

图 3: 客户端与 SMTP 服务器交互

图二中, 因为 SMTP 协议是基于 TCP 的, 所以先进行三次握手, 客户端与服务器建立 TCP 连接, 服务器返回连接信息, 表示是否连接成功————S：220 $smtp.qq.com\ Estmp\ QQ\ Mail\ Server$ (服务器发送 220 表示服务器已准备就绪)

图三呈现了 SMTP 服务器与邮箱客户端交互的全过程。

1. 客户端向服务器发送命令 $HELO$, 并加上本机的主机名 $LAPTOP-0MF1HBFO$ 服务器响应并回复 250 表示服务器可用。
2. 客户端向服务器发送用户登录命令 $AUTHLOGIN$, 服务器回复的两个 334 分别表示用户名和密码, 之后客户端向服务器发送编码后的用户名和密码 (SMTP 要求用户名和密码都通过 base64 编码后再发送, 不接受明文), 服务器分别回复 235 表示身份验证成功
3. 客户端先后向服务器发送 $MAILFROM$ 和 $RCPTTO$ 命令, 后面分别加上发件人和收件人的邮箱地址, 服务器回应 250 表示成功接受
4. 客户端向服务器发送命令 $DATA$, 表示将要向服务器发送邮件正文, 服务器回应 354 End data with <CR> <LF>.<CR><LF> 表示开始邮件输入, 以 “\r\n.\r\n” 结束
5. 客户端将邮件内容发送给服务器 (422bytes), 服务器回应 250 表示接收成功。此时邮件已成功发送到服务器, 客户端向服务器发送命令 QUIT, 断开与服务器的连接, 服务器回应 221 表示同意。双方断开连接, 通信过程结束。

五、 简化 SMTP 服务器实现

(一) 流程图 & 程序构成

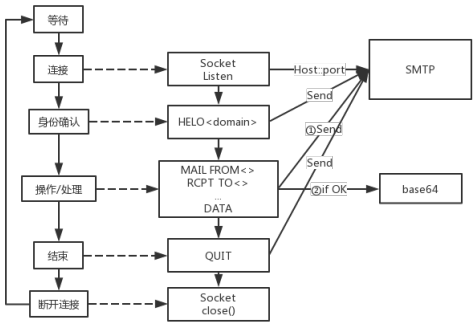

图 4: 流程图

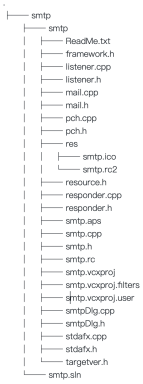

图 5: 程序构成

(二) 编程思路

1. 监听客户端

通过继承 CAsyncSocket 并重载 OnAccept() 实现, 这部分实现在 listen 类中

Listen

```
void listener::OnAccept(int nErrorCode)
{// 获取当前主窗口对象dlg便于操作
    CsmtpDlg *dlg = (CsmtpDlg*)AfxGetApp()->GetMainWnd();
    // 在日志框中输出表示OnAccept发生连接到客户端
    dlg->m_log.AddString(L"*** 收到连接请求");
    //responder类重载了OnSend()和OnReceive()
    responder *responderSocket = new responder();
    //每次新建一个Socket就往表里添加一个
    dlg->socket_array.Add(responderSocket);
        dlg->socket_num ++;//数目也增加
        responderSocket->num = dlg->socket_num;//编号对应
    if (Accept(*responderSocket))
    {
        dlg->m_log.AddString(L"*** 建立连接");
        responderSocket->AsyncSelect(FD_WRITE);
        //选择FD_WRITE触发OnSend()函数发送220Ready
    }
    else
        delete responderSocket;
    CAsyncSocket::OnAccept(nErrorCode);
}
```

有了 listen 类之后在对话框类 smtpDlg.cpp 中创建监听 Socket 与客户端相连接

Init

```
m_log.AddString(L"*** SMTP服务器准备好\n");
m_log.AddString(L"$$$$$$$$$$$$$$$$$$$$$$$$\n");
socket_num = 0;//初始化socket数
if (mylistener.m_hSocket == INVALID_SOCKET)
{
    //监听套接字,FD_ACCEPT,默认端口25 →OnAccept()
        BOOL bFlag = mylistener.Create(25, SOCK_STREAM, FD_ACCEPT,NULL);
        if (!bFlag)//创建失败
        {
                AfxMessageBox(L"Socket创建失败!");
                mylistener.Close();
        PostQuitMessage(0);
        return TRUE;
    }
}
```

2. SMTP 服务器与客户端交互

简化 SMTP 服务器与邮箱客户端的交互通过继承 CAsyncSocket 并重载 OnSend() 和 OnReceive() 实现,这部分实现在 responder 类中。其中 OnReceive() 负责从客户端接收数据并解析之后再通过 OnSend() 给出响应。以下是两个重载函数的大致框架:

responder

```
void responder::OnSend(int nErrorCode)
{//获取dlg
    CsmtpDlg* dlg = (CsmtpDlg*)AfxGetApp()->GetMainWnd();
    //以下是对客户端不同命令的对应响应
    if(smtp_count==0){}//220 ready具体实现见下文
    else if(smtp_count==1){}//250 ok HELO
    else if(smtp_count==2){}//250 mail from
    else if(smtp_count==3){}//250 receiver
    else if(smtp_count==4){}//354 data
    else if(smtp_count==5){}//250 data
    else if(smtp_count==6){}//221 succeed
    CAsyncSocket::OnSend(nErrorCode);
}

void responder::OnReceive(int nErrorCode)
{
    CsmtpDlg *dlg = (CsmtpDlg*)AfxGetApp()->GetMainWnd();
        memset(m_buffer, 0, sizeof(m_buffer));//缓冲清零接收多邮件
    length = Receive(m_buffer, sizeof(m_buffer), 0);
    //从客户端接收到的存到m_buffer,length为消息长度
        CString receiveStr(m_buffer);//将缓冲区char型数组转化为CString
        while (isData && !isEnd)//是数据但未结束,缓冲区不够
        {
                data += receiveStr;
```

```
                if (receiveStr.Right(5) != "\r\n.\r\n")
                        return;
                else
                        isEnd = true;
        }
        //消息接收    具体实现见后文
    if (receiveStr.Left(4) == "HELO" || receiveStr.Left(4) == "EHLO"){}
    if (receiveStr.Left(10) == "MAIL FROM:"){}
    if (receiveStr.Left(8) == "RCPT TO:"){}
    if (receiveStr.Left(4) == "DATA"){}
    if (isData&&isEnd){}
    if (receiveStr.Left(4) == "QUIT")}{}
    CAsyncSocket::OnReceive(nErrorCode);
}
```

根据上述框架进行相应不同情况的处理,由于实现思路一致,报告中仅以服务器响应“EHLO”和服务器接收数据完成后响应“250 data”为例说明:

responder

```
void responder::OnReceive(int nErrorCode)
{
    if (receiveStr.Left(4) == "HELO" || receiveStr.Left(4) == "EHLO")
        {//对客户端的HELO或EHLO进行响应
                dlg->m_log.AddString(L"C:" + receiveStr);//日志输出客户端命令
                smtp_count++;//步骤数+1进行下一步
                AsyncSelect(FD_WRITE);//触发OnSend()传送信息
        }
    else if (isData&&isEnd)
        {//新建mail并将其加入vector
                mail *nowMail = new mail();
                nowMail->decodeAll(data);//初步解码将各部分拆分
                dlg->mail_array.Add(nowMail);//将解码后的内容存入mail类
                dlg->mail_num = dlg->mail_array.GetSize() - 1;//当前邮件计数
                dlg->showNowMail();//在邮件内容框中输出邮件内容
                smtp_count++;//步骤数+1进行下一步
                AsyncSelect(FD_WRITE);//触发OnSend()传送信息
        }
}

void responder::OnSend(int nErrorCode)
{
    if(smtp_count==0)//第0步告诉客户端220准备好
        {
                char* ready = "220 ready\r\n";
                Send(ready, strlen(ready));//响应客户端
        dlg->m_log.AddString
        (L"S:220 Simple Mail Server Ready For Mail Service");
        //显示日志
                AsyncSelect(FD_READ);//触发OnReceive()
```

```
    }
    else if (smtp_count == 5)//第五步告知client数据接收完成
        {
                char* replyStr = "250 data\r\n";
                Send(replyStr, strlen(replyStr));
        dlg->m_log.AddString
        (L"S:250  Message accepted for delivery!");
                isData = false;//后续与isEnd结合处理断开连接
                AsyncSelect(FD_READ);  //触发OnReceive()
        }
}
```

3. mail 类对邮件操作

通过上述对 CAsyncSocket 重载已经完成了这次作业的大部分, 后续我采用新建立一个 mail 类来对服务器接收到的邮件进行后续的处理, 通过 mail 类我还实现了基本实验要求以外的多封邮件显示、多用户发送以及附件图片显示的功能。

在上述 Socket 的 Receive 进行到第五步的时候, 调用了 mail 类中的 decodeAll 函数对接收到的数据内容进行初步解析分隔并存到对应 mail 类的成员变量中,decodeAll 实现基于接收到的邮件内容的显示,decodeAll() 的实现如下 (篇幅有限, 代码有所省略):

decodeAll

```
void mail::decodeAll(CString data)
{//获取主窗口对象操控dlg
    CsmtpDlg* dlg = (CsmtpDlg*)AfxGetApp()->GetMainWnd();
    mailContent = data;
    //from    //find串得到的是F的位置,从0开始
    int pos_name = data.Find(L"From:");
    if (pos_name == -1)//未找到
        mailFrom = (L"发件邮箱获取失败");
    else
    {
        pos_name += 5;//从from:后开始找
        int posName = data.Find('<', pos_name);
        mailFrom = data.Mid(pos_name+2, posName - pos_name-4);
    }
    //to类似from省略
    //正文
    int pos_content = data.Find(L"Content-Type: text/plain;");
    if (pos_content == -1)
        AfxMessageBox(L"未找到邮件正文,请检查邮件");
    else
    {
        int linenum = 0;   //后面有几行
        while (linenum != 4)
        {
            if (data[pos_content++] == '\n')
                linenum++;
```

```
        }
        int posContent = pos_content;
        while (data.Mid(posContent, 6) != "------")//\r\n.\r\n后面是这个
            posContent++;
        posContent -= 4;//位置前推到没有多找的\r\n.\r\n处
        CString base64_text = data.Mid(pos_content, posContent - pos_content);
        base64_text.Remove('\r');//删掉不属于base64的\r\n
        base64_text.Remove('\n');
        int text_length = 0;//base64解码
        mail_text = CString(Text_DecodeBase64(base64_text,text_length));
    }
    //图片
        int pos_image= data.Find(L"filename");
        if (pos_image == -1)     //如果没有附件则返回
        {
                mail_filename = (L"无附件");
        }
        else
    {
        CsmtpDlg* dlg = (CsmtpDlg*)AfxGetApp()->GetMainWnd();
        CString  ImageContent; //存未解析的提取的image内容
        CString  TxtContent;//存未解析的附件文本内容
        data.Remove('\r');//同正文
        data.Remove('\n');
        pos_image = data.Find(L"filename");//删除\r\n后位置有所变化
        int pos_img1 = data.Find('"', pos_image + 7);//附件名在""中间
        int pos_img2 = data.Find('"', pos_img1 + 1);
        mail_filename = data.Mid(pos_img1 + 1, pos_img2 - pos_img1 - 1);
        dlg->m_filename.SetWindowTextW(mail_filename);//显示附件名
        dlg->UpdateData(false);//更新数据
        if (mail_filename.Right(3) == 'jpg' ||
        mail_filename.Right(3) == 'png' || mail_filename.Right(3) == 'bmp')
        {//筛选图片后缀名
            int pos_pic = data.Find(L"------=", pos_img2 + 1);//同正文
        //获取图片内容
            ImageContent = data.Mid(pos_img2 + 1, pos_pic - pos_img2 - 1);
            std::vector<char> bytes;
            //图片用字节流形式储存base64解码后存到bytes中
            Image_DecodeBase64(ImageContent, bytes);
            //将解码后的内容以二进制字节流形式赋给文件名方便后续CImage显示
            std::fstream
            of(mail_filename, std::ios_base::out | std::ios_base::binary);
            of.write(static_cast<const char*>(&bytes[0]), bytes.size());
            of.close();
        }
        else
        {//处理txt后缀
            int posTxtContent = pos_img2;
```

```
            while (data.Mid(posTxtContent, 6) != "------")
                posTxtContent++;
            posTxtContent -= 4;//跳过\r\n\r\n
            CString attached_txt=data.Mid(pos_img2+1,posTxtContent-pos_img2+3);
            attached_txt.Remove('\r');//删掉不属于base64的\r\n
            attached_txt.Remove('\n');
            int text_length = 0;
            mail_txt = "Attached: ";
            mail_txt += CString(Text_DecodeBase64(attached_txt, text_length));
        }
    }
}
```

在经过上述 decodeAll() 函数的解析后，再对文本以及图片分别进行 base64 解码即可得到最终显示在对话框上的内容

decodebase64

```
unsigned char* mail::Text_DecodeBase64(CString base64_code, int &code_length)
{
    unsigned char* base64
    =(unsigned char*)
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
    CString src;//存索引转换
    unsigned char *real_code; //最终str
    int length = base64_code.GetLength();
    unsigned char* p;
    src.GetBufferSetLength(length);
    for (int i = 0; i < length; i++)
        src.SetAt(i, base64_code[i]);
    while (length > 0 && src[length - 1] == '=')
    {
        src.SetAt(length - 1, 0);//=实际上是空字符用来占位的
        length--;
    }
    for (int i = 0; i < length; i++)
    {//strchr找出第一个匹配的
        p = (unsigned char*)strchr((const char*)base64, (int)src[i]);
        if (!p) break;
        src.SetAt(i, p - (unsigned char*)base64);
    }
    real_code = new unsigned char[length * 3 / 4 + 1];
    memset(real_code, 0, length * 3 / 4 + 1);//置0
    for (int i = 0, j = 0; i < length; i += 4, j += 3) {
        real_code[j] = (src[i] << 2) + ((src[i + 1] & 0x30) >> 4);
        //第一个6bit+第二个6bit的前两位
        real_code[j+1]=((src[i + 1]&0x0F)<< 4)+((src[i+2]&0x3C)>>2);
        //第二个6bit的后四位+第三个6bit的前四位
        real_code[j + 2] = ((src[i + 2] & 0x03) << 6) + src[i + 3];
        //第三个6bit的后两位+第四个6bit
```

```
33      }//图片base64解码与text基本一致，除了最后输出的不是字符串而是字节流
34      code_length = length * 3 / 4 + 1;
35      return real_code;
36  }
```

经过上述 base64 解码的转换，再通过 stmpDlg 控制将 mail 成员写入控件即可。此外还附加两个功能，其一是，多封邮件的显示和切换，也不再赘述，只需要根据当前邮件所属的编号显示对应 mail 的成员即可；其二是同时与多客户端和多封邮件交互的多线程操作，前文代码已有所展示，即借助一个 vector 记录 socket 情况，一旦触发了 OnAccept 函数，就新建一个 socket 并将这一个 socket 进行记录，而在 socket 执行完数据接收的任务 (即发送 221Quit) 以后，再将这一 socket 关闭并将其从 vector 中移除即可。

六、 程序演示

- 首先需要先将邮箱的发件服务器按照下图设置，否则本地简易服务器无法实现功能

图 6: 邮箱发件服务器初始设置

- 初始界面如下图所示，两个按钮可以分别向前向后切换邮件内容

图 7: 初始界面

- 使用 Foxmail 发送一封无附件邮件

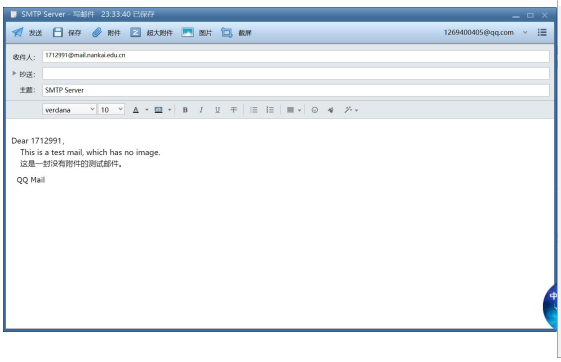

图 8: 邮件示意图

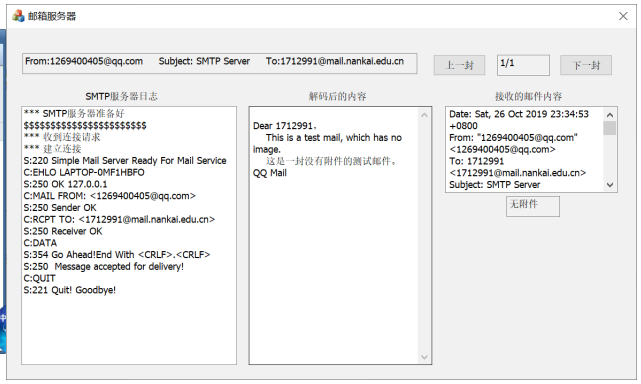

图 9: 服务器接收

- 使用 Foxmail 发送带附件邮件

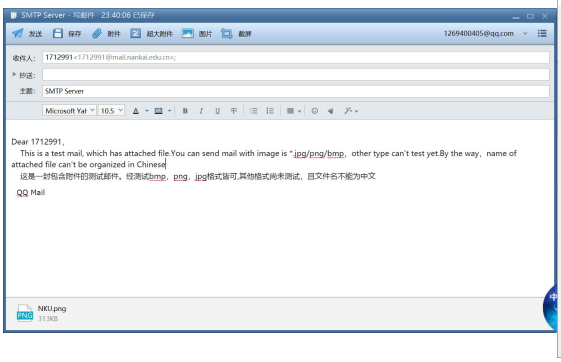

图 10: 邮件示意图

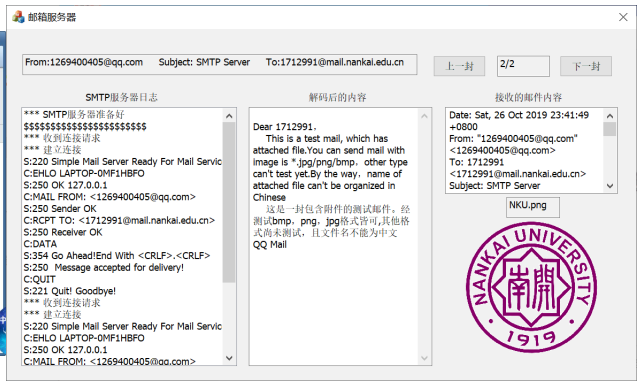

图 11: 服务器接收图片附件

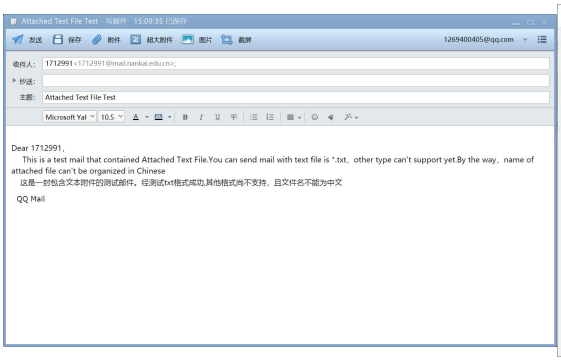

图 12: 邮件示意图

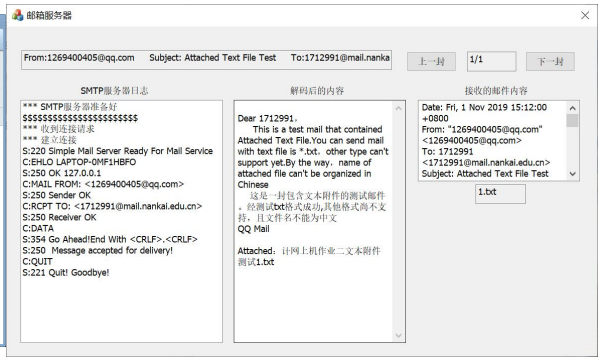

图 13: 服务器接收文本附件

- 此外也可以进行多用户切换

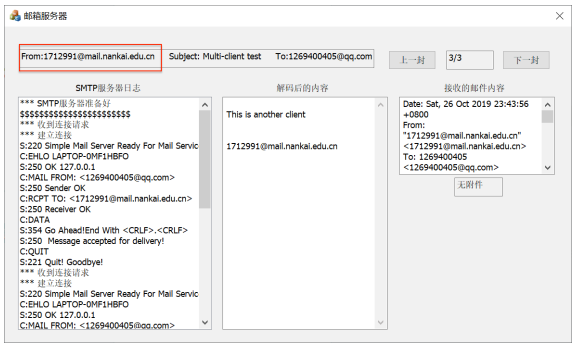

图 14: 多用户

- 同时发送多封邮件

图 15: SMTP 服务器日志

接收到的邮件如下, 也不会因为同时接收而使得内容混乱 (这里服务器的图片显示失真是缩放过小的原因, 打开接收到的文件查看是没有这一问题的)

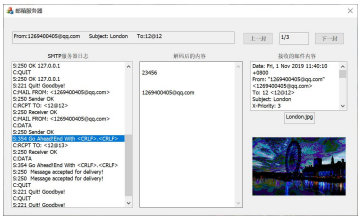

图 16: 邮件 1

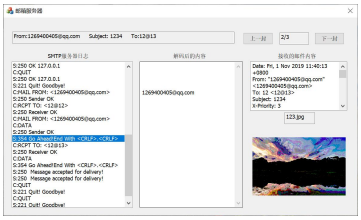

图 17: 邮件 2

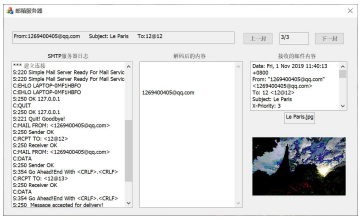

图 18: 邮件 3

七、 实验遇到问题及解决

- 最初的设定不能进行邮件翻页功能, 后来在此功能上创新的时候本来只是想使用各自变量的数组去存储, 但这样做的缺陷一是存储容易混乱, 操作复杂, 二是不移维护, 所以最后干脆推翻重写, 直接新建一个 mail 类结合 vector 用以维护多封邮件各自不同的成员变量, 操作简单, 且可以把原先随便放在 responder 中的解码函数也放到 mail 中, 合情合理

- 邮箱发件服务器设置为 127.0.0.1 是没有响应的, 只有设为 localhost 才行
- 开始的文本和图片 base64 解码是同一个解码函数, 后来发现这样没有没法很好的显示图片, 所以参照网上的代码, 图片的解码最后输出的是二进制的字节流, 而不是和原先文本一致的字符串, 输出字节流之后将其写入附加名中, 最后再通过 CImage 加载的 picture control 中即可
- 最早的正文结束判断用的是“\r\n.\r\n”, 这样导致的问题是如果正文中出现了类似故意输入的文字, 邮件内容显示就会出现偏差。所以在研究接收到的数据报文后, 选择使用“——”结合前文的正文终结符同时作为判断正文结尾的依据
- 这一次 MFC 的调试断点以及纯眼看 debug 无效, 由于未知太多, 加上判断邮件各部分内容起止的位置标记很难确定, 所以调试的时候我使用的是在各种需要查看截取邮件内容或者程序执行正确与否的地方借助一个 TextEdit 框显示字符串实现的, 通过这一举措, 很好的解决了我连蒙带猜决定邮件分块起止位置的难题
- 最早的 resonder 中的两个重载函数并没有执行步骤一说, 只是根据前一步的执行利用标记来跳转, 后来加上多封邮件之后的测试发现, 如果发送多封邮件就会出现邮箱服务器日志顺序颠倒的问题, 所以在 responder 类中加入了作为标记执行顺序的变量
- 起先没有加入多线程的操作, 所以同时发送的带大图片的邮件只能一封接着一封收, 可以看到进度条是一个完成之后才下一个的, 但是加入了一个简单的或许都算不得多么高深的并发操作, 就可以同时接收邮件了, 再测试中后两封邮件与日志相一致, 几乎是同时就接收成功的, 仍然收邮件速度较慢的原因大概是写字节流到 CImage 的速度原因, 后期改进也可以在此多线程操作
- 接收较大图片附件的时候, 可以看到在附件 PictureControl 显示的图片有失真的现象, 但是打开接收到的图片文件却没有这一问题, 初步判定是缩放过小等操作引起的偏差, 但是既然图片附件接收文件没有问题, 对实用性就不存在影响, 所以 PictureControl 并没有进行优化。
- 最终测试的时候使用过中文命名的图片, 由于会被转换成 GB2312 编码, 所以不能正常显示附件名以及图片, 一开始还以为是 base64 编码, 后来放到 base64 解码并搜索引擎才确认是 GB 包含中文的编码, 由于中文的解析以及字节流的写入还需要另写代码, 而这对于此次试验可以说是无关紧要, 所以解决方案是不使用中文命名的附件, 此外, 如果本身是分辨率较高的大图片文件, 显示在服务器程序上可能会有一定程度失真, 且上传速度较慢
- 最后的最后, 最早做好的版本只有图片附件, 后来看到完整版作业要求以后嫌麻烦没做 txt 附件的处理, 所以就想着给学姐解说一下 txt 的处理就不去做了, 结果学姐还是说那你就做完整一点再拿来给我看, 于是又花了五分钟左右在 mail 类加了一个 filename 后缀名的识别, 实现了这一功能

八、 总结

计网实验二学习了如何使用 CAsyncSocket 类编写一个简单 SMTP 邮件服务器程序, 更熟悉 Socket 编程以及加深了 TCP 建立连接和 SMTP 协议的理解。通过 Wireshark 对发送邮件的过程的抓包分析更直观的了解到了电子邮件客户端与服务器之间的交互过程及细节。之后再通过亲自编写这一比之第一次作业 更深入细致的程序, 加深对邮件服务系统的理解, 体会实现邮箱服务器不算太漫长艰难的过程, 加深对应用层 SMTP 和 TCP 协议的理解。