

计算机网络上机作业

作业 2：编写 SMTP 服务器并观察通信过程

实验报告

November 10, 2019

学号：1711425

姓名：曹元议

专业：计算机科学与技术

Contents

1 实验要求	1
2 开发环境	1
3 界面设计	1
4 实验原理	3
4.1 SMTP 协议	3
4.2 MIME 邮件报文格式	4
4.3 Base64	6
4.4 程序实现	7
5 实现思路和代码解释	7
5.1 与客户端的交互	8
5.2 处理邮件内容	22
6 程序演示	29
6.1 对客户端的设置	29
6.2 不添加附件	30
6.3 把图片添加到附件	31
6.4 把图片嵌入正文	32
6.5 发送文本附件	33
6.6 发送其他附件	34
6.7 发送多个附件	36
6.8 捕获数据包并查看通信过程	39

1 实验要求

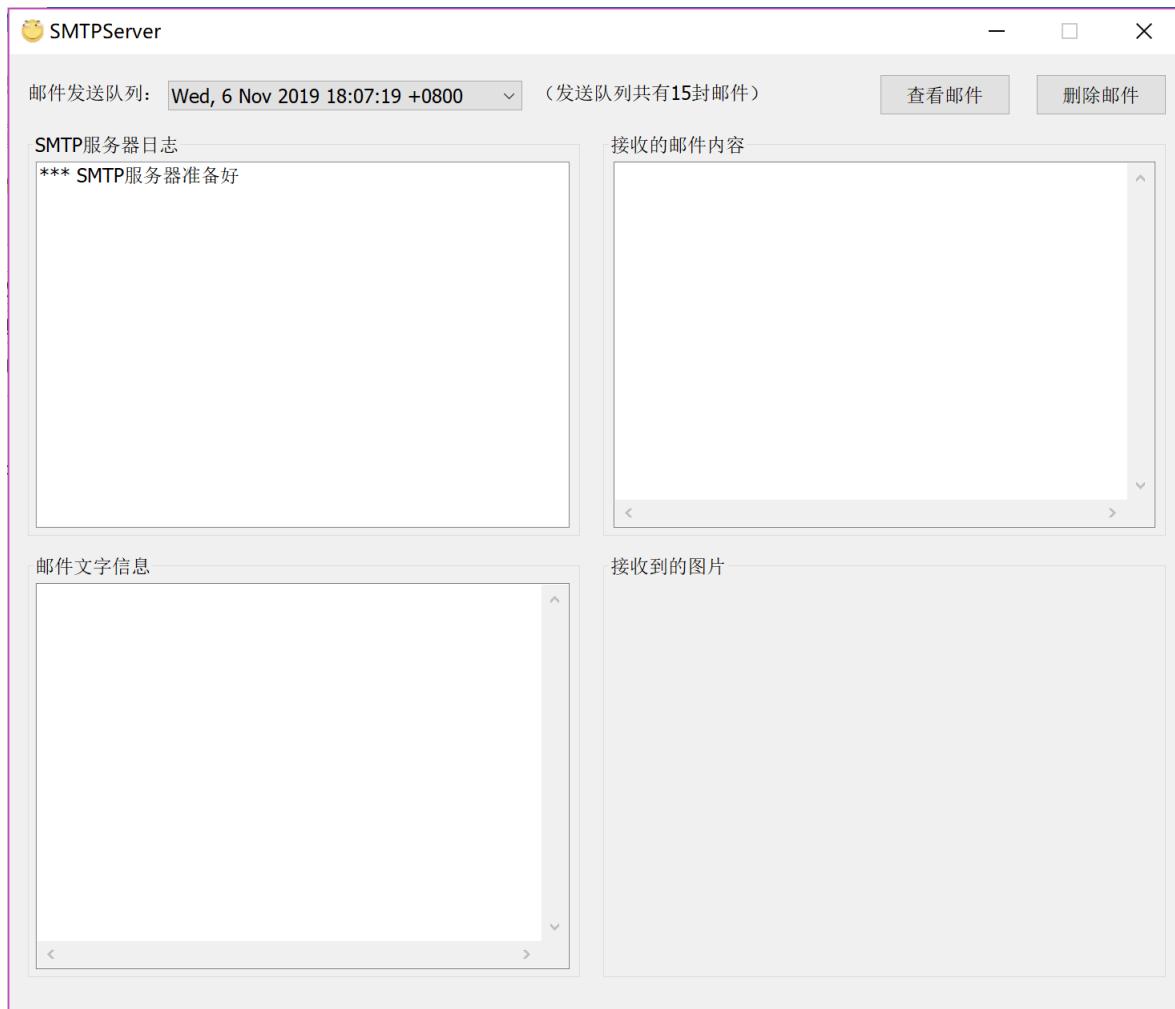
1. 响应客户 SMTP 命令，并将命令的交互过程和收到的邮件显示到屏幕上（注：可正常显示邮件内容和附件，附件为图片或文档）；
2. 支持单用户（注：客户端为本计算机邮箱）；
3. 不存储和转发收到的邮件；
4. 不做错误处理

2 开发环境

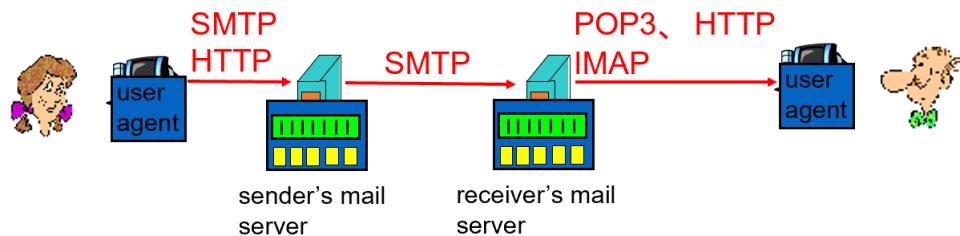
1. Windows 10 64 位专业版
2. Visual Studio 2015
3. Outlook 2016

3 界面设计

控件的排列方式也是为了看起来协调，没有其他目的。左上角的框显示 SMTP 与客户端的交互过程。右上角的框显示本次收到的邮件（即客户端发送完 DATA 命令服务器响应之后，客户端发送 QUIT 命令前客户端传送的内容原文）。左下角框为解析右上角框的内容（显示发送方、接收方、日期、标题、正文、附件信息），附件信息的附件格式任意。右下角框显示一张接收到的图片（如果有多个图片只显示程序识别出的第一张图片），支持各种图片格式。除了左上角的交互日志框不清空以外，下一次接收邮件会清空其余三个框，重新显示下一个邮件内容。

**Figure 1:** 界面设计

界面最上方是本次存储的接收到的邮件列表，右上方的按键可以查看或删除队列中的某一封邮件。这里描述成队列是参考了这张图中发送方服务器的发送队列：

**Figure 2:** 发送和接收邮件报文的过程

4 实验原理

4.1 SMTP 协议

SMTP 是一个应用层协议，用于邮件的发送，下层使用可靠的 TCP 协议实现。邮件服务器之间的通信和客户代理向发送服务器发送邮件都可以使用 SMTP 协议。

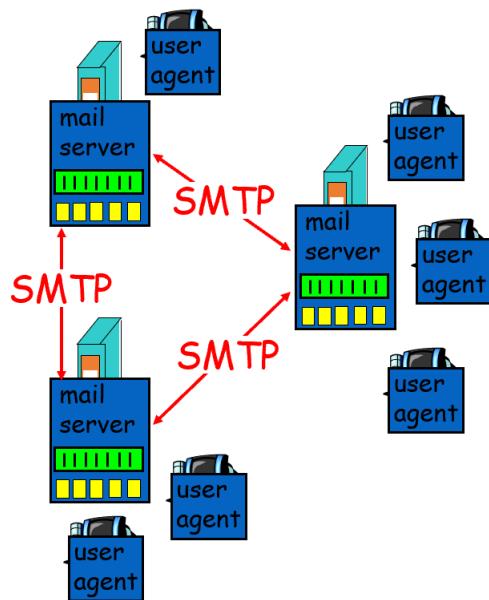


Figure 3: 电子邮件系统

下面展示了 SMTP 协议主要的命令以及响应。

命令	含义
HELO 或 EHLO < 主机名 >	与服务器确认，通知其客户端使用的机器名称
AUTH	使用 AUTH LOGIN 与服务器进行登录验证
MAIL FROM: < 发件人邮箱地址 >	指定发件人信息
RCPT TO: < 收件人邮箱地址 >	指定收件人信息
DATA	发送邮件数据（在此之前一定要先指定发件人和收件人信息）
RSET	重置会话，当前传输被取消
QUIT	请求断开与服务器的连接

Table 1: SMTP 主要命令

响应码	含义
220	服务器准备就绪
221	服务关闭了传输通道
235	验证通过
250	请求命令成功完成
353	可以发送邮件数据（与 DATA 命令结合）
500	无法辨识的命令
501	命令参数错误
502	命令未实现
503	命令序列错误

Table 2: SMTP 主要响应码

SMTP 的工作过程大致如下（这里的客户端可以是客户代理也可以是一个邮件服务器）：

1. 首先，客户端请求与服务器的 25 端口建立一个 TCP 连接。一旦连接建立，SMTP 服务器和客户端就开始相互通报自己的域名，同时确认对方的域名。
2. 然后，客户端利用 MAIL、RCPT 和 DATA 命令将邮件的发件人邮箱地址、收件人邮箱地址和邮件的具体内容发送给 SMTP 服务器。SMTP 服务器进行相应的响应并接收邮件。
3. 最后，客户端发送 QUIT 命令，服务器在处理命令后进行响应，随后关闭本次的 TCP 连接。

每条 SMTP 的交互内容的最后一定有一个回车换行。

4.2 MIME 邮件报文格式

以下是发送的邮件的大致格式，也是本程序需要解析的内容。

```

From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched

Happy birthday to you
Happy birthday to you
Happy birthday dear <b>Carolyn</b>
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
access-type="anon-ftp";
site="bicycle.abcd.com";
directory="pub";
name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--

```

Figure 4: 邮件报文格式

由于 SMTP 协议的历史原因，报文信息只能用 7 位 ASCII 字符传输。因此在邮件报文中使用了 MIME 协议，它主要解决了多媒体等二进制信息利用电子邮件传输的问题。

邮件内容的主要类型如下：

Type	Subtype	Description
Text	Plain	Unformatted text
	Enriched	Text including simple formatting commands
Image	Gif	Still picture in GIF format
	Jpeg	Still picture in JPEG format
Audio	Basic	Audible sound
Video	Mpeg	Movie in MPEG format
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in PostScript
Message	Rfc822	A MIME RFC 822 message
	Partial	Message has been split for transmission
	External-body	Message itself must be fetched over the net
Multipart	Mixed	Independent parts in the specified order
	Alternative	Same message in different formats
	Parallel	Parts must be viewed simultaneously
	Digest	Each part is a complete RFC 822 message

Figure 5: 内容的类型

本程序解析的类型有：Text（文本）、Image（图片）、Application（附件）。

邮件内容的编码主要有：quoted-printable（7bit）和 Base64。quoted-printable 主要是用 7 位 ASCII 字符表示的，如果表示非 ASCII 字符则需要用 = 和该字节对应的 16 进制码（大写）表示。例如字符值 230（对应二进制值 1110 0110），因此可表示为=E6。quoted-printable 主要用于邮件文本。由于时间关系，本程序未实现 quoted-printable 对非 ASCII 字符的处理。

4.3 Base64

Base64 是网络上最常见的用于传输 8Bit 字节码的编码方式之一，它是一种基于 64 个可打印字符来表示二进制数据的方法。也是本程序要求实现解码的部分。

Base64 编码的基本思想是将每 3 个字节（共 24 位）作为一个整体将其划分为 4 组，每组六位。然后将每组 6 位的值作为索引，将其映射为对应可打印 ASCII 字符。因此，Base64 将 3 个字节转换成了 4 个可打印字符。

为了进行 Base64 编码，需要将原始数据分为多个组，每组 3 个字节。如果原始数据的字节总数不是 3 的整数倍，那么最后一组有可能仅剩有一个或两个字节。如果最后只剩 1 个字节，则在后面补 4 个比特的 0，形成 12 位二进制数值，再将其分成 2 个 6 位组。将这两个 6 位数组映射为两个可打印的 ASCII 字符，而后在后面填充两个字符”=”，形成 4 个字符；如果最后只剩两个字节，则在后面补 2 个比特的 0，形成 18 位二进制数值，再将其分成 3 个 6 位组。将这三个 6 位组映射成三个可打印的 ASCII 字符，而后在后面填充 1 个字符”=”，形成 4 个字符。因此在末尾补的 = 的数量直接反应了 Base64 编码时在末尾补充的 0 字节数。

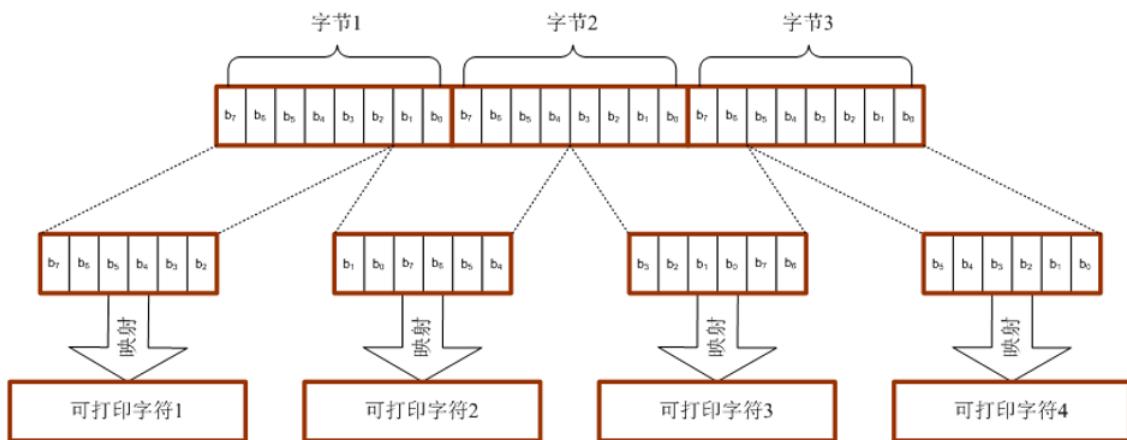


Figure 6: Base64 编码原理图

因此，Base64 的解码就是编码的逆过程。首先将这些可打印字符从头开始映射到 6 位二进制的 Base64 编码（这些二进制编码的总长度是 24 的倍数），再以 8 个 bit 一组得

到原来的字符。最后要根据结尾的 = 的数量决定在结尾去掉多少个冗余 0 字节。

4.4 程序实现

在程序实现上，这次依然可以使用**CAsyncSocket**类实现。

CAsyncSocket类主要的函数及其功能重复列举如下：

函数	功能
Create()	创建和初始化一个套接字
Send() 或 SendTo()	发送数据
Receive() 或 ReceiveFrom()	接收数据
Connect()	请求连接服务器
Listen()	侦听连接请求
Accept()	接收连接请求
Bind()	将 IP 地址、端口号与套接字绑定
Close()	关闭套接字
AsyncSelect()	选择接收消息的模式

Table 3: CAsyncSocket 类的主要的函数及其功能

如上所述，由于 SMTP 的底层使用 TCP，需要建立连接。由于这次只需要实现服务器端，因此上述的函数除了**Connect()**以外都需要使用。使用**Send()**和**Receive()**而不使用**SendTo()**和**ReceiveFrom()**是因为 TCP 需要建立连接，发送和接收数据都是在这个已经建立好的连接上的，对方的 IP 地址和端口号已经在建立连接（**Connect()**）的时候就已经指定好了，所以无需指定对方的 IP 地址和端口号，所以使用不指定对方 IP 地址和端口号的**Send()**和**Receive()**。服务器端需要使用**Bind()**，因为服务器的 IP 和端口号一般是固定的（SMTP 发送服务器所在的端口一般在 25）。因此实现一个 SMTP 服务器的基本思想就是先在 25 端口建立一个 TCP 服务器，然后这个服务器与邮件客户端的交互的只需要在用**Send()**函数发送内容的时候使用上述的 SMTP 的格式即可（末尾要加回车换行）。

5 实现思路和代码解释

本节仅展示主要的涉及实验原理的代码，其余代码在项目文件中。

使用**CAsyncSocket**类实现 TCP 客户端和服务器通信的流程图如下，下述的与客户端交互的代码实现也是基于这个流程：

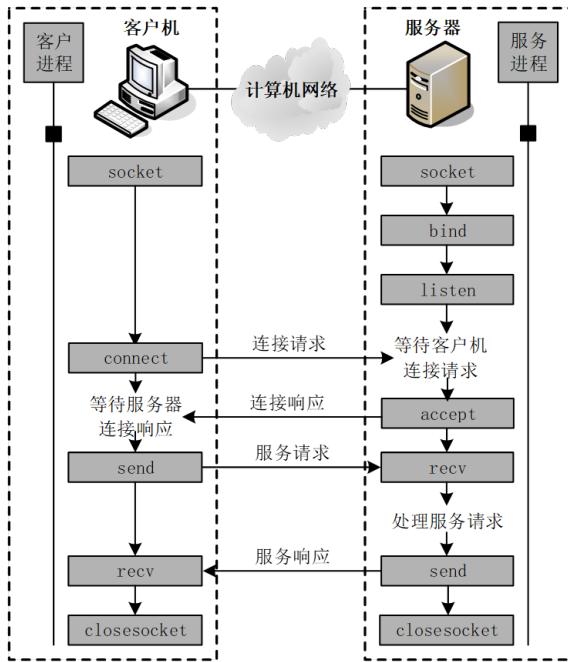


Figure 7: 流程图

上图可概括为：客户端进程在向服务器请求之前先要创建套接字（`Create()`），然后使用`Connect()`函数向指定的服务器进程发出连接请求，得到服务器端的连接响应之后使用`Send()`函数向服务器发送字节流，`Receive()`函数接收服务器端的响应字节流，最后应使用`Close()`函数关闭套接字。对于一次通信，服务器端需要建立两个套接字：一个用于监听请求，另一个用于与客户端通信（相当于一个子进程或者线程）。由于服务器端进程是总在一个固定的 IP 地址和端口号在线的，所以必须是服务器进程创建之后不久服务器端就要创建监听套接字，另外还应使用`Bind()`函数把服务器端进程的 IP、端口号与监听套接字绑定，然后使用`Listen()`函数监听客户端的连接请求，监听到连接请求后需要新建一个专门用于与客户端通信的套接字并使用`Accept()`函数对连接请求作出响应，然后使用这个新的套接字并使用`Send()`函数向客户端发送响应字节流，`Receive()`函数接收客户端的字节流，最后与客户端断开连接需要关闭这个新建的套接字。服务器继续监听连接请求。最后服务器进程在退出的时候应关闭监听套接字。

5.1 与客户端的交互

源代码和项目文件在文件夹SMTPServer中。

主体的实现框架是一个 TCP 服务器。

首先从`CAsyncSocket`类派生出一个子类`CServerSocket`专门用于监听客户端的连接请求，并重写这个子类的虚函数`CServerSocket::OnAccept()`作为监听到请求时服

务器该做的动作。然后在对话框类CUDPServerDlg中增加一个新的CServerSocket的指针成员变量m_pSocket，表明CServerSocket类的对象将在堆上创建。这里选择在CSMTPServerDlg的构造函数创建新的对象，在析构函数中销毁该对象。

由于服务器是总在线的，并且其 IP 和端口号一般是固定的，要保证服务器进程刚创建不久就要创建监听套接字，在进程退出时关闭监听套接字，因此选择在对话框初始化的时候就创建监听套接字并将服务器 IP 地址和端口号与监听套接字绑定，对话框对象被销毁时再关闭监听套接字，以营造服务器总是在线并在同一个地方守候的样子。

以下是对CSMTPServerDlg::OnInitDialog()函数的修改：

```
1 BOOL CUDPServerDlg::OnInitDialog()
2 {
3     CDialogEx::OnInitDialog();
4
5     // 将“关于...”菜单项添加到系统菜单中。
6
7     .....
8
9     // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动
10    // 执行此操作
11    SetIcon(m_hIcon, TRUE);      // 设置大图标
12    SetIcon(m_hIcon, FALSE);    // 设置小图标
13
14    // TODO: 在此添加额外的初始化代码
15    m_picture.GetClientRect(&m_picrect); // 获取图片区域大小，为打印图片作准备
16    // 从本地读取发送队列
17    string folderPath = "queue";
18    if (GetFileAttributesA(folderPath.c_str()) == INVALID_FILE_ATTRIBUTES)
19    {
20        CString dir = char2CString((char *)folderPath.c_str());
21        bool flag = CreateDirectory(dir, NULL); // 如果没有创建文件夹就新建一个
22    }
23    vector<string> files;
24    getfilebytime(folderPath, files); // 获取文件夹中所有文件并按时间由早到晚
25    // 排列
26
27    // 从文件中读取邮件内容
28    for (int i = 0; i < files.size(); i++)
29    {
30        ifstream infile(files[i].c_str(), ios::in | ios::binary);
31        MailInfo *pinfo = new MailInfo;
32        string data;
33        int len = 0; // 此处是根据所读的长度来决定下一个字符串读多长
```

```
33     infile.read((char *)&len, sizeof(int));
34     pinfo->date = new char[len + 1];
35     infile.read(pinfo->date, len);
36     pinfo->date[len] = '\0'; //从这里读出的字符串没有'\0'需手动添加
37     infile.read((char *)&len, sizeof(int));
38     pinfo->from = new char[len + 1];
39     infile.read(pinfo->from, len);
40     pinfo->from[len] = '\0';
41     infile.read((char *)&len, sizeof(int));
42     pinfo->to = new char[len + 1];
43     infile.read(pinfo->to, len);
44     pinfo->to[len] = '\0';
45     infile.read((char *)&len, sizeof(int));
46     pinfo->mailstr = new char[len + 1];
47     infile.read(pinfo->mailstr, len);
48     pinfo->mailstr[len] = '\0';
49     info.push_back(*pinfo);
50     infile.close();
51     m_queue.InsertString(m_queue.GetCount(), char2CString(pinfo->date));
52 }
53
54 //如果没有任何邮件，则禁用两个按钮
55 if (!files.size())
56 {
57     m_lookmail.EnableWindow(FALSE);
58     m_delemail.EnableWindow(FALSE);
59 }
60 else
61 {
62     m_queue.SetCurSel(0);
63 }
64
65 CString temp1, temp2;
66 temp2.Format(L"%d", files.size());
67 ((CStatic*)GetDlgItem(IDC_TIPS))->GetWindowTextW(temp1);
68 ((CStatic*)GetDlgItem(IDC_TIPS))->SetWindowTextW(temp1 + temp2 + L"封邮
件"); //显示队列中邮件个数
69
70
71 //指定好服务器的IP和端口，采用本地IP 127.0.0.1和SMTP的默认端口25
72 CString ip = L"127.0.0.1";
73 int port = 25;
```

```
74 //创建监听套接字
75 if (!m_pSocket->Create(port, SOCK_STREAM, FD_ACCEPT, ip)) //FD_ACCEPT为设
    置的事件为接收连接的事件（OnAccept）
76 {
77     MessageBox(L"Create socket error!", L"SMTPServer", MB_OK | MB_ICONERROR
    );
78     return FALSE;
79 }
80 //将IP和端口与监听套接字绑定
81 m_pSocket->Bind(port, ip);
82 //监听，等待连接请求
83 //nConnectionBacklog: 挂起连接的队列可以增长到的最大长度。有效范围是从1到
    5（默认）。
84 if (!m_pSocket->Listen())
85 {
86     //如果不是被阻塞，则错误
87     if (GetLastError() != WSAEWOULDBLOCK)
88     {
89         CString error;
90         int errorcode = GetLastError();
91         error.Format(L"Socket failed to listen: %d", errorcode);
92         LPVOID lpMsgBuf;
93         FormatMessage(
94             FORMAT_MESSAGE_ALLOCATE_BUFFER |
95             FORMAT_MESSAGE_FROM_SYSTEM |
96             FORMAT_MESSAGE_IGNORE_INSERTS,
97             NULL,
98             errorcode,
99             MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
100            (LPTSTR)&lpMsgBuf,
101            0,
102            NULL
103        );
104         error = error + L"\n" + (LPCTSTR)lpMsgBuf;
105         AfxMessageBox(error);
106         //m_pSocket->Close();
107         return FALSE;
108     }
109 }
110 else
111 {
112     m_loglist.InsertString(m_loglist.GetCount(), L"*** SMTP服务器准备好");
```

```

113 }
114 return TRUE; // 除非将焦点设置到控件，否则返回 TRUE
115 }
```

其中，函数调用Create(port, SOCK_STREAM, FD_READ, ip)的第一个参数是服务器的端口号。第二个参数SOCK_STREAM表示创建流式套接字，因为TCP是以字节流的方式传送数据的。第三个参数FD_ACCEPT表示一个事件，会触发对CServerSocket类的虚函数OnAccept()的调用，这个函数表示服务器可以接收客户端连接请求并在收到连接请求之后应该做的动作（响应客户端连接请求，打印日志等），MFC大量使用消息机制。第四个参数是服务器的IP地址。

对连接请求作出响应的CServerSocket::OnAccept()函数如下：

```

1 void CServerSocket::OnAccept(int nErrorCode)
2 {
3     // TODO: 在此添加专用代码和/或调用基类
4     CSMTSPServerDlg *pDlg = (CSMTSPServerDlg*)AfxGetApp()->GetMainWnd(); // 获取
      主窗口句柄
5     pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), L"
     $$$$$$$$$$$$$$$$$$$$");
6     pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), L"*** 收到连接请
      求");
7     m_pSocket = new CCClientSocket; // 新建用于收发信息的套接字
8     if (!Accept(*m_pSocket)) // 接收客户端连接请求
9     {
10         if (GetLastError() != WSAEWOULDBLOCK)
11         {
12             CString error;
13             int errorcode = GetLastError();
14             error.Format(L"Socket failed to accept: %d", errorcode);
15             LPVOID lpMsgBuf;
16             FormatMessage(
17                 FORMAT_MESSAGE_ALLOCATE_BUFFER |
18                 FORMAT_MESSAGE_FROM_SYSTEM |
19                 FORMAT_MESSAGE_IGNORE_INSERTS,
20                 NULL,
21                 errorcode,
22                 MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
23                 (LPTSTR)&lpMsgBuf,
24                 0,
25                 NULL
26             );
27             error = error + L"\n" + (LPCTSTR)lpMsgBuf;
```

```

28     AfxMessageBox(error);
29     delete m_pSocket;
30     CString temp;
31     temp.Format(L"*** 建立连接失败，错误码%d", errorcode);
32     pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), temp);
33     return;
34 }
35 }
36 else
37 {
38     pDlg->m_loglist.InsertString(pDlg->m_loglist.GetCount(), L"*** 建立连接");
39     // 设置新的套接字需要接受的消息类型 (send)，用于向客户端发出问候消息
40     m_pSocket->AsyncSelect(FD_WRITE);
41 }
42 CAsyncSocket::OnAccept(nErrorCode);
43 }
```

其中，函数`Accept(*m_pSocket)`为服务器的接收客户端的连接请求。参数表示新建的专用于与客户端收发信息的套接字（后两个没有表示出来的参数为接收连接套接字的地址及其长度，由于不需要这个信息，所以这两个参数默认为 0），类型为`CCClientSocket`，也是派生自`CAsyncSocket`类。最后的`AsyncSelect(FD_WRITE)`指定这个新套接字接收消息的类型，为`FD_WRITE`，当收到这个消息时调用虚函数`OnSend()`向客户端发出问候消息（响应码为 220）。

以下是`CCClientSocket::OnSend()`函数的定义：

```

1 void CCClientSocket::OnSend(int nErrorCode)
2 {
3     // TODO: 在此添加专用代码和/或调用基类
4     CSMTSPServerDlg *pDlg = (CSMTSPServerDlg*)AfxGetApp()->GetMainWnd();
5     pDlg->greetClient(this);
6     CAsyncSocket::OnSend(nErrorCode);
7 }
8
9 void CSMTSPServerDlg::greetClient(CCClientSocket *pSocket)
10 {
11     // 此函数由CCClientSocket::OnSend()调用
12     if (!start) // 如果还没有进入到对话状态，就向客户端发出问候消息
13     {
14         sprintf(sendBuffer, "220 127.0.0.1 SMTP Mail Server\r\n");
15         start = true;
16     }
}
```

```
17     else
18         return;
19     int ret = pSocket->Send(sendBuffer, strlen(sendBuffer)); //发送消息缓冲区
20     长度只需是发送的字符串的有效长度即可，否则会发出很多无关的消息，无法与客
21     户端进行通信
22     if (ret == SOCKET_ERROR)
23     {
24         //WSAEWOULDBLOCK：非阻塞模式下，请求的操作被阻塞，需等待再次调用
25         if (GetLastError() != WSAEWOULDBLOCK)
26         {
27             CString error;
28             int errorcode = GetLastError();
29             error.Format(L"Socket failed to send: %d", errorcode);
30             LPVOID lpMsgBuf;
31             FormatMessage(
32                 FORMAT_MESSAGE_ALLOCATE_BUFFER |
33                 FORMAT_MESSAGE_FROM_SYSTEM |
34                 FORMAT_MESSAGE_IGNORE_INSERTS,
35                 NULL,
36                 errorcode,
37                 MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
38                 (LPTSTR)&lpMsgBuf,
39                 0,
40                 NULL
41             );
42             error = error + L"\n" + (LPCTSTR)lpMsgBuf;
43             AfxMessageBox(error);
44             pSocket->Close();
45         }
46     }
47     else
48     {
49         CString temp = char2CString(sendBuffer);
50         temp = L"S: " + temp;
51         m_loglist.InsertString(m_loglist.GetCount(), temp);
52         pSocket->AsyncSelect(FD_READ); //设置接收消息类型为receive，准备与客户
53         端会话或接收邮件数据
54     }
55 }
```

为了方便，把这个函数的主体放在CSMTPServerDlg::greetClient()中实现了，这个套接字通过参数参入，就可以在CSMTPServerDlg中操作套接字了。Send()函数传

入的第二个参数发送消息缓冲区的大小只需是发送的字符串的有效长度即可，否则会发出很多无关的消息，无法与客户端进行通信。最后通过**AsyncSelect()**函数设置接收消息的类型为**FD_READ**，这样之后**OnReceive()**虚函数被调用，表示接收客户端的消息并应答。**OnSend()**函数的使命就结束了。以下是**CClientSocket::OnReceive()**函数的定义：

```
1 void CClientSocket::OnReceive(int nErrorCode)
2 {
3     // TODO: 在此添加专用代码和/或调用基类
4     CSMTSPServerDlg *pDlg = (CSMTSPServerDlg*)AfxGetApp()->GetMainWnd();
5     pDlg->recvClient(this);
6     if (pDlg->quit)
7         Close();
8     CAsyncSocket::OnReceive(nErrorCode);
9 }
10
11 void CSMTSPServerDlg::recvClient(CClientSocket *pSocket)
12 {
13     //接收客户端消息并与其进行交互，有由函数CClientSocket::OnReceive()函数调用
14     quit = false;
15     memset(recvBuffer, 0, sizeof(recvBuffer));
16     int ret = pSocket->Receive(recvBuffer, sizeof(recvBuffer) - 1);
17     if (ret == SOCKET_ERROR)
18     {
19         //WSAEWOULDBLOCK: 非阻塞模式下，请求的操作被阻塞，需等待再次调用
20         if (GetLastError() != WSAEWOULDBLOCK)
21         {
22             CString error;
23             int errorcode = GetLastError();
24             error.Format(L"Socket failed to receive: %d", errorcode);
25             LPVOID lpMsgBuf;
26             FormatMessage(
27                 FORMAT_MESSAGE_ALLOCATE_BUFFER |
28                 FORMAT_MESSAGE_FROM_SYSTEM |
29                 FORMAT_MESSAGE_IGNORE_INSERTS,
30                 NULL,
31                 errorcode,
32                 MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
33                 (LPTSTR)&lpMsgBuf,
34                 0,
35                 NULL
```

```
36     );
37     error = error + L"\n" + (LPCTSTR)lpMsgBuf;
38     AfxMessageBox(error);
39     pSocket->Close();
40 }
41 }
42 else
43 {
44     if (isinputdata) //如果客户端正在发送邮件数据
45     {
46         delethis = false;
47         recvBuffer[ret] = '\0'; //terminate the string
48         CString temp = char2CString(recvBuffer), temp2;
49         m_mailrecv.GetWindowTextW(temp2);
50         temp2 += temp;
51         m_mailrecv.SetWindowTextW(temp2);
52         if (temp.Find(L"\r\n.\r\n") >= 0) //如果遇到单独的.和回车换行则停止接收邮件数据，返回接收成功
53     {
54         isinputdata = false;
55         memset(sendBuffer, 0, sizeof(sendBuffer));
56         sprintf(sendBuffer, "250 Message acceptd for delivery\r\n");
57         ret = pSocket->Send(sendBuffer, strlen(sendBuffer));
58         if (ret == SOCKET_ERROR)
59     {
60         //WSAEWOULDBLOCK: 非阻塞模式下，请求的操作被阻塞，需等待再次调用
61         if (GetLastError() != WSAEWOULDBLOCK)
62     {
63         CString error;
64         int errorcode = GetLastError();
65         error.Format(L"Socket failed to send: %d", errorcode);
66         LPVOID lpMsgBuf;
67         FormatMessage(
68             FORMAT_MESSAGE_ALLOCATE_BUFFER |
69             FORMAT_MESSAGE_FROM_SYSTEM |
70             FORMAT_MESSAGE_IGNORE_INSERTS,
71             NULL,
72             errorcode,
73             MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
74             (LPTSTR)&lpMsgBuf,
75             0,
76             NULL
```

```
77         );
78     error = error + L"\n" + (LPCTSTR)lpMsgBuf;
79     AfxMessageBox(error);
80     pSocket->Close();
81 }
82 }
83 else
84 {
85     CString temp = char2CString(sendBuffer);
86     temp = L"S: " + temp;
87     m_loglist.InsertString(m_loglist.GetCount(), temp);
88     pSocket->AsyncSelect(FD_READ); //设置事件为接收(OnReceive)
89     AfxBeginThread(processMail, NULL, THREAD_PRIORITY_NORMAL); //处理
邮件内容
90 }
91 }
92 }
93 else //进入命令处理
94 {
95     CString tempp;
96     CString deal = char2CString(recvBuffer);
97     tempp = L"R: " + deal;
98     m_loglist.InsertString(m_loglist.GetCount(), tempp);
99     CString deal2 = deal.MakeLower(); //把接收到的内容转为小写再比较
100    deal = char2CString(recvBuffer);
101    if (!deal2.Find(L"hello") || !deal2.Find(L"ehlo"))
102    {
103        regex reg("(hello|ehlo) [\\S]+\\r\\n");
104        string test(CString2char(deal2));
105        memset(sendBuffer, 0, sizeof(sendBuffer));
106        if (!regex_match(test, reg)) //如果不匹配正则表达式，则命令的语法错
误
107    {
108        sprintf(sendBuffer, "501 Syntax error\\r\\n");
109    }
110    else
111    {
112        sprintf(sendBuffer, "250 Hello, pleased to meet you\\r\\n");
113    }
114    }
115    else if (!deal2.Find(L"mail from"))
116    {
```

```
117     regex reg("mail from:[ ]?<\\w+([\\-\\+\\.]\\w+)*@\\w+([\\-\\.]\\w+)*(&lt;\\w+([\\-\\.]\\w+)*?)>\\r\\n");
118     string test(CString2char(deal2));
119     memset(sendBuffer, 0, sizeof(sendBuffer));
120     if (!regex_match(test, reg))
121     {
122         sprintf(sendBuffer, "501 Syntax error\\r\\n");
123     }
124     else
125     {
126         CString temp = deal.Mid(deal.Find(L"<") + 1);
127         temp = temp.Left(temp.Find(L">"));
128         from = CString2char(temp);
129         sprintf(sendBuffer, "250 %s... Sender ok\\r\\n", from);
130     }
131 }
132 else if (!deal2.Find(L"rcpt to"))
133 {
134     regex reg("rcpt to:[ ]?<\\w+([\\-\\+\\.]\\w+)*@\\w+([\\-\\.]\\w+)*(&lt;\\w+([\\-\\.]\\w+)*?)>\\r\\n");
135     string test(CString2char(deal2));
136     memset(sendBuffer, 0, sizeof(sendBuffer));
137     if (!regex_match(test, reg))
138     {
139         sprintf(sendBuffer, "501 Syntax error\\r\\n");
140     }
141     else
142     {
143         CString temp = deal.Mid(deal.Find(L"<") + 1);
144         temp = temp.Left(temp.Find(L">"));
145         to = CString2char(temp);
146         sprintf(sendBuffer, "250 %s... Recipient ok\\r\\n", to);
147     }
148 }
149 else if (!deal2.Find(L"data"))
150 {
151     if (!from || !to) //如果还未指定发送方和接收方，则返回命令序列错误
152     {
153         memset(sendBuffer, 0, sizeof(sendBuffer));
154         sprintf(sendBuffer, "503 Wrong command order\\r\\n");
155     }
156     else
```

```
157     {
158         memset(sendBuffer, 0, sizeof(sendBuffer));
159         if (deal2.GetLength() == 6)
160     {
161         sprintf(sendBuffer, "354 Enter mail, end with \".\" on a line
by itself\r\n");
162         isinputdata = true; //进入接收数据模式
163         hasimage = false;
164         m_mailrecv.SetWindowTextW(L"");
件内容
165         m_mailtext.SetWindowTextW(L"");
166         m_lookmail.EnableWindow(FALSE); //禁用两个按钮
167         m_delemail.EnableWindow(FALSE);
168     }
169     else
170     {
171         sprintf(sendBuffer, "501 Syntax error\r\n");
172     }
173 }
174 }
175 else if (!deal2.Find(L"quit"))
176 {
177     memset(sendBuffer, 0, sizeof(sendBuffer));
178     memset(sendBuffer, 0, sizeof(sendBuffer));
179     if (deal2.GetLength() == 6)
180     {
181         sprintf(sendBuffer, "221 Quit, goodbye\r\n");
182         start = false;
183         quit = true; //关闭负责通信的套接字
184     }
185     else
186     {
187         sprintf(sendBuffer, "501 Syntax error\r\n");
188     }
189 }
190 else if (!deal2.Find(L"rset"))
191 {
192     memset(sendBuffer, 0, sizeof(sendBuffer));
193     memset(sendBuffer, 0, sizeof(sendBuffer));
194     if (deal2.GetLength() == 6)
195     {
196         sprintf(sendBuffer, "250 OK\r\n");
```

```
197     }
198     else
199     {
200         sprintf(sendBuffer, "501 Syntax error\r\n");
201     }
202 }
203 else if (!deal2.Compare(L"\r\n") == 0)
204 {
205     memset(sendBuffer, 0, sizeof(sendBuffer));
206     sprintf(sendBuffer, "250 Choose a command\r\n");
207 }
208 else //如果不能匹配上述的任何一个命令，则返回命令未定义
209 {
210     memset(sendBuffer, 0, sizeof(sendBuffer));
211     sprintf(sendBuffer, "502 Bad command\r\n");
212 }
213 ret = pSocket->Send(sendBuffer, strlen(sendBuffer)); //发送邮件的缓冲
区大小应和发送内容有效长度一致
214 if (ret == SOCKET_ERROR)
215 {
216     //WSAEWOULDBLOCK: 非阻塞模式下，请求的操作被阻塞，需等待再次调用
217     if (GetLastError() != WSAEWOULDBLOCK)
218     {
219         CString error;
220         int errorcode = GetLastError();
221         error.Format(L"Socket failed to send: %d", errorcode);
222         LPVOID lpMsgBuf;
223         FormatMessage(
224             FORMAT_MESSAGE_ALLOCATE_BUFFER |
225             FORMAT_MESSAGE_FROM_SYSTEM |
226             FORMAT_MESSAGE_IGNORE_INSERTS,
227             NULL,
228             errorcode,
229             MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), //Default language
230             (LPTSTR)&lpMsgBuf,
231             0,
232             NULL
233         );
234         error = error + L"\n" + (LPCTSTR)lpMsgBuf;
235         AfxMessageBox(error);
236         pSocket->Close();
237 }
```

```

238     }
239     else
240     {
241         CString temp = char2CString(sendBuffer);
242         temp = L"S: " + temp;
243         m_loglist.InsertString(m_loglist.GetCount(), temp);
244         pSocket->AsyncSelect(FD_READ); // 设置接收消息类型，以免之后不再调用
245         OnReceive();
246     }
247 }
248 }
```

还是一样，把这个函数的主体放在`CSMTPServerDlg::recvClient()`中实现了。其中，`Receive()`的第一个参数是接收消息缓冲区，第二个参数是缓冲区的大小，设置成`sizeof(recvBuffer)-1`的原因是由于很多邮件消息的长度大于设置的缓冲区的长度，而之后需要根据消息字符串的有效长度（根据函数返回值`ret`确定）在末尾加'\0'作为字符串结束，如果不减1则在加入结束符之后重写了`CSMTPServerDlg`的其他成员变量。后两个参数是引用，该函数返回以后会得到客户端的IP地址和端口号。该函数的返回值`ret`为收到消息的字节数，若为-1则接收错误，需要作相应的错误处理。最后通过`AsyncSelect()`函数设置接收消息的类型为`FD_READ`，这样之后`OnReceive()`虚函数被调用，表示继续接收客户端的消息并应答，否则`OnReceive()`可能不再被调用。

在调用完`Receive()`之后，如不出现任何错误，则会进入对接收消息的处理。首先用布尔变量`isinputdata`来判断是否正在接收邮件信息。如果是，则等到客户端发来的消息出现\r\n.\r\n再发出已成功接收邮件内容的响应，并把这个变量置为`false`以继续接收客户端的命令，与此同时开启处理邮件内容的线程；否则客户端发送的是命令，需要对命令作出回应。这里根据不同的命令设置不同的响应，如果命令和语法均正确，则返回正确的响应码（250、354、221等）；如果命令参数不符合语法规则（跟设置的正则表达式不匹配或者无参命令后面跟随参数）则返回参数错误的响应（501）；如果没有匹配的命令，则返回命令未定义响应（502）。如果客户端在未指定发件方和收件方的邮箱地址就发送 DATA 命令，则返回命令序列错误的响应（503）。如果客户端发来QUIT命令，则设置布尔变量`quit`的值为`true`关闭这个负责通信的套接字（相当于断开连接）。

为处理邮件内容单独开设工作者线程的原因主要是这个过程往往时间比较久，这样就不会阻塞主界面线程从而导致主界面无法操作（例如无法滚动查看日志及接收到的邮件报文内容等）。次要的是客户端也可以继续和服务器进行通信。该线程会调用`processMail()`函数，注意这个不是`CSMTPServerDlg`的成员函数，是`AfxBeginThread()`可

以调用的普通函数。

5.2 处理邮件内容

注：这里的代码比较冗长，所以主要讲思路。由于我不知道作业要求更新了（是真的不知道 10 月 30 号发的通知的意思是重新上传了作业要求的意思，以为只是重复通知，就没有去看），需要实现对文本附件的处理，无需考虑多个客户端以及存储邮件的问题。但是当时周三给助教检查完以后同学提醒我了才知道。考虑到我周五下午有课并且对这一部分修改的成本不大，因此在这里的代码和当时给助教检查的有所不同。非常抱歉。因此我以后在更加注意的同时也希望如果是更新了作业要求一定要在通知里面说明清楚是更新了作业要求，否则可能会产生歧义以为是重复通知。后来我问了一部分同学他们也不知道这个。辛苦各位老师和助教学长学姐了。

基本的思路是先按照Content-Type所对应的类型从整个收到的邮件内容中提取出所在的子串的所有位置，在根据这些位置截取到子串。然后对每一个子串用相应的正则表达式匹配（使用 C++ 标准库regex），提取出编码、文件名、文件内容等信息。再根据提取出的编码信息决定是否要进行 base64 解码。如果是附件，会保存到本地。这样就可以从本地加载图片并显示在右下角。文件名和文件内容会显示在左下角的框中。这些匹配的代码就只展示附件解析的部分，因为解析的各个分支都大同小异。

由于很多附件被编码成 base64，故需要对其进行解码。base64 的解码函数如下（原先只是为解码图片设计的，但是其实对所有 base64 编码的附件以及文本内容都可以复用）：

```

1 char *CSMTPServerDlg::base64_decode(char *src)
2 {
3     char base64[] =
4     {
5         'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
6         'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
7         'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
8         'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
9         '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '/'
10    }; //字符所在的数组索引就是其对应的base64编码
11    unordered_map<char, int> base64map; //unordered_map的底层是哈希表实现的,
12    //查询速度快
13    int len = strlen(src), i, equalcount = 0;
14    for (i = 0; i < 64; i++)
15    {
16        base64map[base64[i]] = i; //制作base64字符到值的映射, 方便查找
17    }

```

```
17 string binstr = "";
18 //将字符转为对应的6位二进制字符串，并拼接
19 for (i = 0; i < len; i++)
20 {
21     if (src[i] != '=')
22     {
23         //生成base64字符对应的值的6位二进制字符串
24         char c = base64map[src[i]];
25         char bin[9] = { 0 };
26         itoa(c, bin, 2); //转为二进制字符串
27         char tarbin[7] = { 0 };
28         sprintf(tarbin, "%06s", bin); //控制6位
29         binstr.append(tarbin); //拼接
30     }
31     else
32     {
33         binstr.append("000000"); //如果遇到结尾的等号则直接在二进制字符串后面补上0
34         equalcount++; //等号个数加1
35     }
36 }
37 picbuflen = binstr.length() / 8; //图片的字节大小应是二进制字符串长度的1/8
38 char *stream = new char[picbuflen + 1];
39 int count = 0;
40 //将上面生成的二进制串转为正常的内容，每8个bit（一个Byte）存入一个字符
41 for (i = 0; i < binstr.length(); i += 8) //由于要生成1个字节的字符，所以循环的步长为8
42 {
43     int j;
44     char c = 0;
45     for (j = 0; j < 8; j++)
46     {
47         char temp;
48         temp = (binstr[i + j] - 48);
49         temp <= (8 - 1 - j);
50         temp = temp & 0xff;
51         c += temp;
52         c = c & 0xff;
53     }
54     stream[count++] = c;
55 }
```

```

56     stream[count] = '\0';
57     picbuflen -= equalcount; //因为要原样保存附件，所以要用统计的等号数目修正
      图片大小
58             //不修正也不影响显示，但是最后可能会有冗余0字节
59     return stream;
60 }

```

以下只展示对附件的匹配：

```

1  UINT processMail(LPVOID hWnd)
2 {
3     CSMTSPServerDlg *pDlg = (CSMTSPServerDlg*)AfxGetApp()->GetMainWnd(); //获取
      主窗口句柄
4     pDlg->hasimage = false; //清空右下角区域 同时也是判断有没有图片显示的标志
5     pDlg->firstpicfile = NULL; //第一个图片
6     pDlg->Invalidate(true);
7     pDlg->m_lookmail.EnableWindow(FALSE);
8     pDlg->m_delemail.EnableWindow(FALSE);
9     pDlg->m_queue.EnableWindow(FALSE);
10    CString mail;
11    if (pDlg->islookfile) //如果正在查看队列中的邮件，就从队列中获取待解析的
      邮件内容
12    {
13        mail = char2CString(pDlg->mailstr);
14        pDlg->m_mailrecv.SetWindowTextW(mail);
15    }
16    else //如果是正在接收邮件，则从窗口中获取待解析的邮件内容
17    {
18        pDlg->m_mailrecv.GetWindowTextW(mail);
19        pDlg->mailstr = CString2char(mail);
20    }
21
22    .....
23
24    //如果可能有图片（有可能是其他附件），则在图片显示框中显示Loading.....
25    if (mail.Find(L"Content-Type: image") >= 0 || mail.Find(L"Content-Type:
      application/octet-stream") >= 0)
26    {
27        pDlg->dealimage = true;
28        pDlg->Invalidate(true);
29        pDlg->m_picture.UpdateWindow();
30    }
31

```

```
32 pDlg->m_mailtext.SetWindowText(L"From: "+char2CString(pDlg->from)+  
33 L"\r\nTo: "+char2CString(pDlg->to)+L"\r\n"); //在邮件文字信息框显示发件  
34 方和收件方  
35 .....  
36  
37 vector<int> txt, txt2, img, attach, attach2;  
38.findall(mail, L"Content-Type: text/plain", txt); //找出相应的所有子串的  
39 索引  
40.findall(mail, L"Content-Type: text/enriched", txt2);  
41.findall(mail, L"Content-Type: image", img);  
42.findall(mail, L"Content-Type: application/octet-stream", attach);  
43.findall(mail, L"Content-Type: application/x-msdownload", attach2);  
44 .....  
45  
46 //如果是一个附件（也有可能是文本或图片）  
47 for (int j = 0; j < attach.size(); j++)  
48 {  
49     CString judge;  
50     if (j == attach.size() - 1)  
51     {  
52         judge = mail.Mid(attach[j]);  
53     }  
54     else  
55     {  
56         judge = mail.Mid(attach[j], attach[j + 1] - attach[j]);  
57     }  
58     regex regpic("Content-Type: [\\w\\-\\/]*;\\s*name=\"(\\S*\\.\\S*)\"\\s*  
Content-Transfer-Encoding: base64\\r\\n(?:Content-ID: <.+>|Content-  
Disposition: attachment;\\r\\n\\tfilename=\".*\")\\r\\n\\r\\n([a-zA-Z0-  
9-/=\\+\\r\\n]+)");  
59     regex regfiletxt("Content-Type: application/octet-stream;\\s*name=\"(\\  
S*\\.\\S*)\"\\s*Content-Transfer-Encoding: (?:7bit|quoted-printable)\\s*  
Content-Disposition: attachment;\\s*filename=\"(?:\\S*\\.\\S*)\"\\r\\n\\r\\n\\r\\n([\\S\\s]*-----");  
60     smatch result1, result2;  
61     string test(CString2char(judge));  
62     regex_search(test, result1, regpic); //搜索在正则表达式中匹配的子串，并  
63     regex_search(test, result2, regfiletxt);  
64     bool matchfilename = false, matchpic = false, picmatch = false;
```

```
65 for (int i = 0; i < result1.size(); i++)
66 { //如果匹配，0号字符串一定是被匹配的字符串的最长的那个（最长匹配原则）
67     if (matchfilename && matchpic) //确保只匹配第一个出现的附件
68     {
69         picmatch = true;
70         break;
71     }
72     regex filenamereg("\\S*\\.\\S*"), picdetailreg("[a-zA-Z0-9/=\\+\\r\\n]+");
73     string a = result1[i].str();
74     if (regex_match(a, filenamereg)) //regex_match()是完全匹配，不是跟某个子串匹配
75     {
76         pDlg->attachname = new char[a.length() + 1];
77         ::strcpy(pDlg->attachname, (char*)a.c_str());
78         matchfilename = true;
79     }
80     else if (regex_match(a, picdetailreg))
81     {
82         pDlg->base64src = new char[a.length() + 1];
83         ::strcpy(pDlg->base64src, (char*)a.c_str());
84         matchpic = true;
85     }
86 }
87 if (matchpic)
88 {
89     CString t;
90     pDlg->m_mailtext.GetWindowTextW(t);
91     pDlg->m_mailtext.SetWindowTextW(t + L"\r\nAttachment Filename: \r\n"
+ char2CString(pDlg->attachname));
92     string base64str = pDlg->base64src;
93     int pos;
94     int count = 0;
95     //去除base64编码中的回车换行
96     while ((pos = base64str.find("\r\n")) >= 0)
97     {
98         base64str = base64str.erase(pos, 2);
99         count++;
100    }
101    delete[] pDlg->base64src;
102    pDlg->base64src = new char[base64str.length() + 1];
103    ::strcpy(pDlg->base64src, (char*)base64str.c_str());
```

```
104     pDlg->picbuf = pDlg->base64_decode(pDlg->base64src); //base64解码
105     ofstream outFile(pDlg->attachname, ios::out | ios::binary);
106     outFile.write(pDlg->picbuf, pDlg->picbuflen); //将附件保存到本地
107     outFile.close();
108     CImage ima;
109     ima.Load(char2CString(pDlg->attachname));
110     if (!ima.IsNull()) //尝试当做图片加载，如果不能加载，则不是图片
111     {
112         if (!pDlg->hasimage) //如果可以加载，就看有没有显示过图片，没有就
113             {
114                 pDlg->firstpicfile = pDlg->attachname;
115                 pDlg->dealimage = false;
116                 pDlg->Invalidate(true); //发送WM_PAINT消息加载图片
117                 pDlg->m_picture.UpdateWindow(); //跳过消息队列
118                 pDlg->hasimage = true;
119             }
120     }
121     else //二进制文件不是图片就把该二进制文件开头可打印的字符（如果有）显
122         {
123             CString t;
124             pDlg->m_mailtext.GetWindowTextW(t);
125             pDlg->m_mailtext.SetWindowTextW(t + L"\r\nContent: \r\n" +
126             char2CString(pDlg->picbuf));
127         }
128     else //附件确定是一个文本文件
129     {
130         for (int i = 1; i < result2.size(); i++)
131         {
132             regex filen("\S*\.\S*");
133             string a = result2[i].str();
134             if (regex_match(a, filen))
135             {
136                 pDlg->attachname = new char[a.length() + 1];
137                 ::strcpy(pDlg->attachname, (char*)a.c_str());
138             }
139             else
140             {
141                 pDlg->base64src = new char[a.length() + 1];
```

```
142         ::strcpy(pDlg->base64src, (char*)a.c_str());
143     }
144 }
145 ofstream outFile(pDlg->attachname, ios::out | ios::binary);
146 outFile.write(pDlg->base64src, strlen(pDlg->base64src)); //将附件保存
到本地
147 outFile.close();
148 CString temp;
149 temp = char2CString(pDlg->base64src);
150 temp = temp.Left(temp.GetLength() - 2); //去掉结尾回车换行
151 pDlg->base64src = CString2char(temp);
152 CString t;
153 pDlg->m_mailtext.GetWindowTextW(t);
154 pDlg->m_mailtext.SetWindowTextW(t + L"\r\nAttachment Filename: \r\n"
+ char2CString(pDlg->attachname));
155 pDlg->m_mailtext.GetWindowTextW(t);
156 pDlg->m_mailtext.SetWindowTextW(t + L"\r\nContent: \r\n" +
char2CString(pDlg->base64src));
157
158 }
159 }
160
161 if (!pDlg->hasimage) //如果不显示图片，则清空右下角的Loading.....
162 {
163     pDlg->dealimage = false;
164     pDlg->Invalidate(true);
165 }
166
167 if (!pDlg->islookfile) //如果是刚接收的邮件，则把它存在队列中
168 {
169     pDlg->stock();
170 }
171 else
172     pDlg->islookfile = false;
173 pDlg->clearInfo(); //清除一些变量信息，为下次做准备
174 return 0;
175 }
```

以下的部分由于篇幅所限不展示，请自行查阅代码文件 **SMTPServerDlg.cpp**。

加载图片的函数 **CSMTPServerDlg::loadImage()** 使用了 **CImage** 类，它可以加载多种格式的图片，但是如果加载 gif 动图就只会加载一帧。因此对 gif 的解析使用了第三方类 **CPictureEx** 类。中心思想是如果图片大小小于右下角框的大小，就原样显示到这

个框中，否则按比例缩放图片到小于右下角框的大小以内再显示。这个函数在出现附件的时候也可以在右下角框中显示Loading.....，表示正在解析邮件内容，因为速度太慢了。

把对右下角区域的更改统一放在了函数CSMTPServerDlg::OnPaint()中，由函数调用Invalidate(true)发出WM_PAINT消息并对OnPaint()进行调用，防止由于窗口调用OnPaint()进行重绘而看不到图片。还有一个清除右下角区域的函数CSMTPServerDlg::clearImage()，就是把这个区域用窗口背景色进行重绘，以达到清除图片的效果。

除此之外，对邮件内容的保存到本地的发送队列CSMTPServerDlg::stock()的中心思想是保存重要的部分到二进制文件中：邮件原文、收件方和发件方、日期，并对这些部分的长度进行保存，以便可以正确读取文件，读取了文件以后这些信息由vector维护，并把日期由早到晚显示到上面的框中。其他不重要的部分直接调用processMail()现场解析（通过读取vector中相应元素的内容）。

最后还有清除现场环境的函数CSMTPServerDlg::clearInfo()，就是清除一些变量（一些变量置为0值，例如base64解码缓冲区picbuf、邮件正文content等），以便接受下一个邮件，覆盖本次显示的内容。当然，在recvClient()中在进入接收邮件数据模式中首先清空右上角框和左下角框，进入processMail()之后先清空右下角区域，也是为接收下一个邮件作准备。

此外，还有一些其他用作工具的函数，例如char2CString()、CString2char()是实现CString和char*的相互转换，为了防止乱码，中心思想是多字节字符和宽字符的相互转换。其他函数就不在这里介绍了。

6 程序演示

6.1 对客户端的设置

在Outlook 2016中，任选一个邮箱（这里使用163邮箱），使用POP3登录帐户。点击左上角的“文件”，再点击“帐户设置”，再点击“服务器设置”，在如下界面中，设置发送服务器的地址为127.0.0.1，端口号不变，并不勾选任何选项。如下图所示：



Figure 8: 客户端

由于 Outlook 会发送测试邮件，因此应保持本邮件服务器程序处于开启状态。

6.2 不添加附件



Figure 9: 客户端

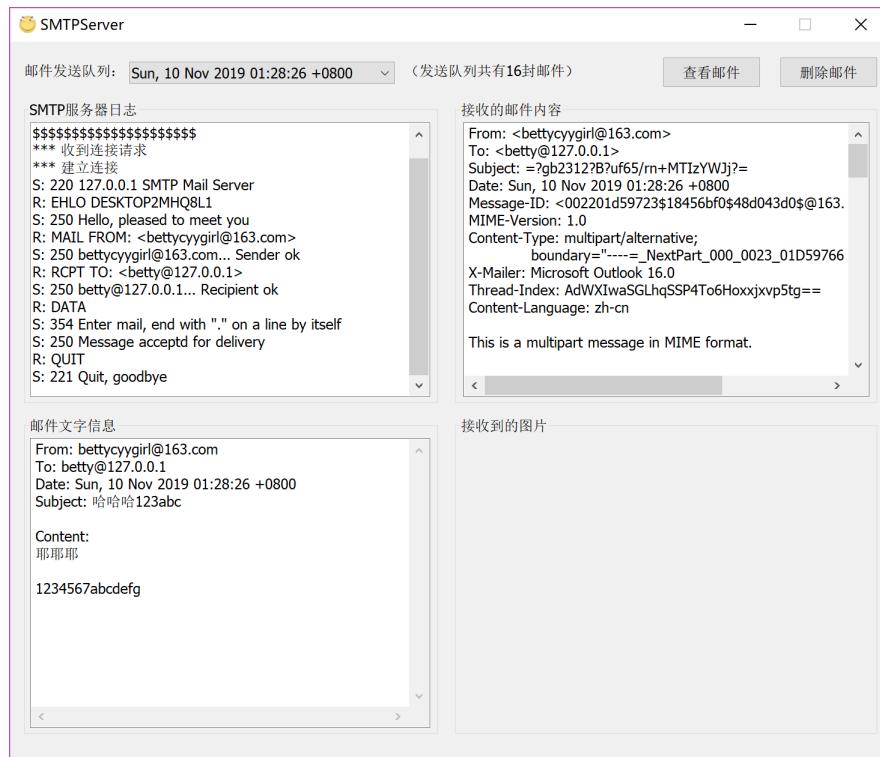


Figure 10: 服务器

6.3 把图片添加到附件



Figure 11: 客户端

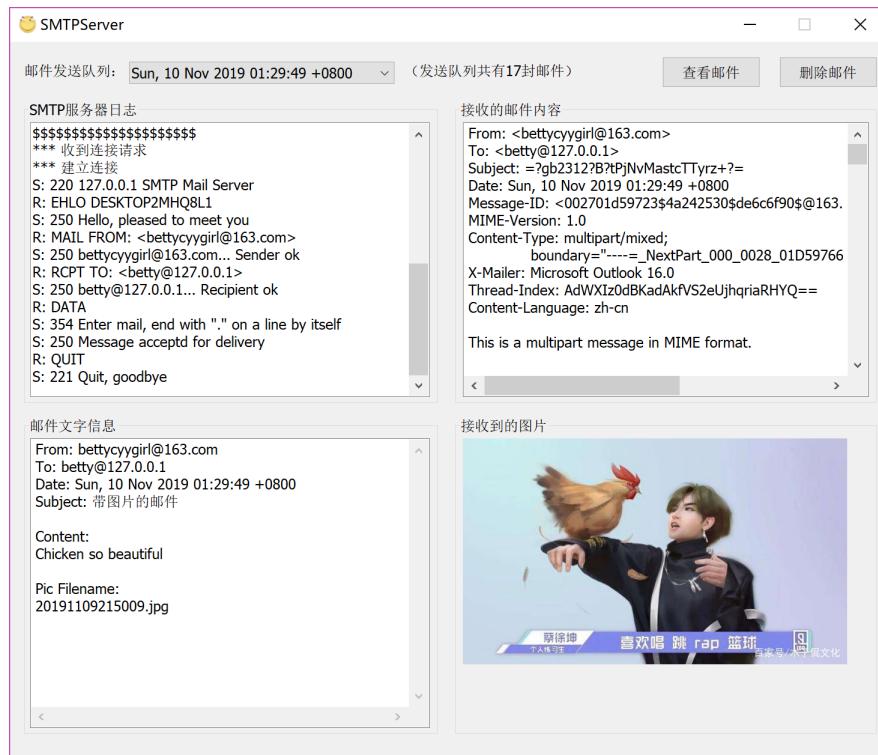


Figure 12: 服务器

6.4 把图片嵌入正文



Figure 13: 客户端

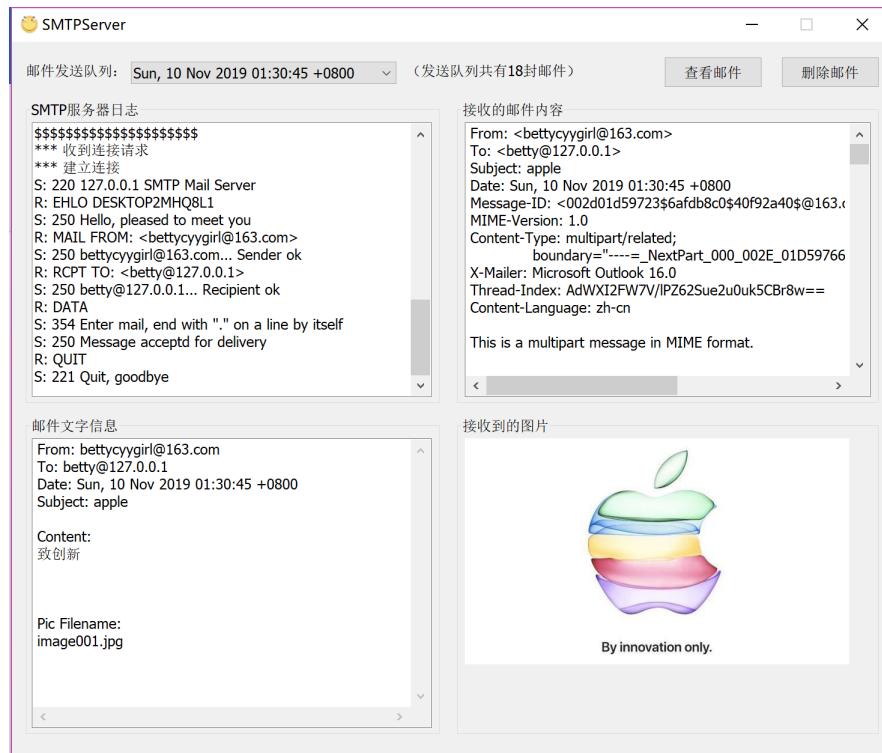


Figure 14: 服务器

6.5 发送文本附件



Figure 15: 客户端

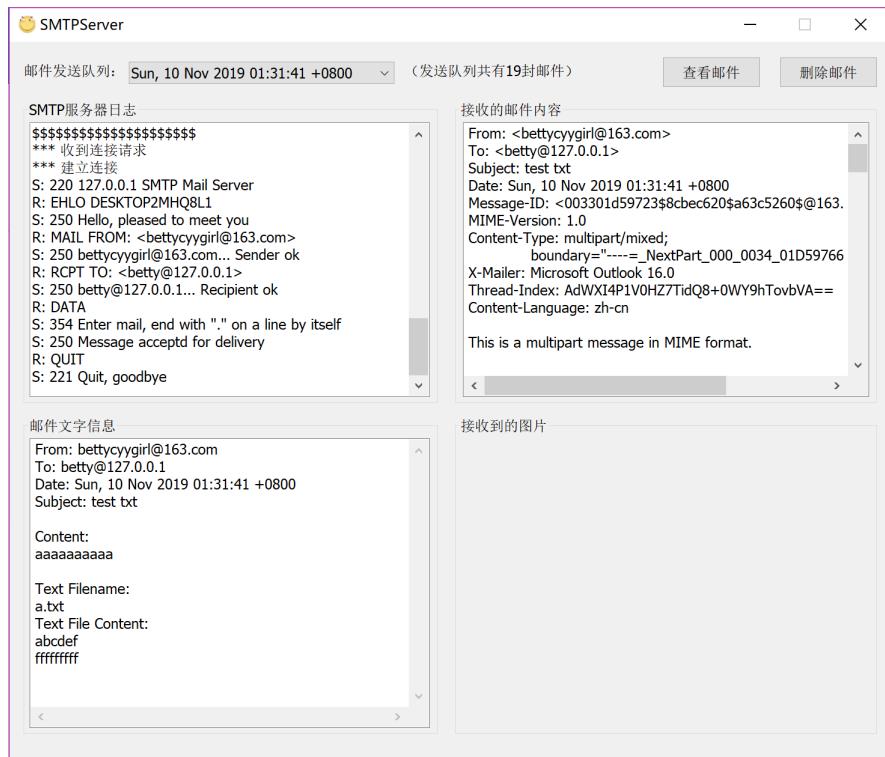


Figure 16: 服务器

6.6 发送其他附件

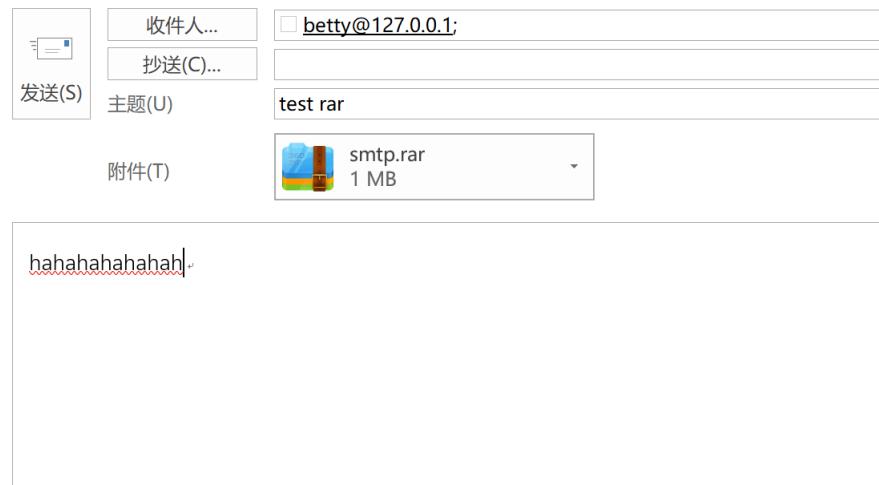


Figure 17: 客戶端

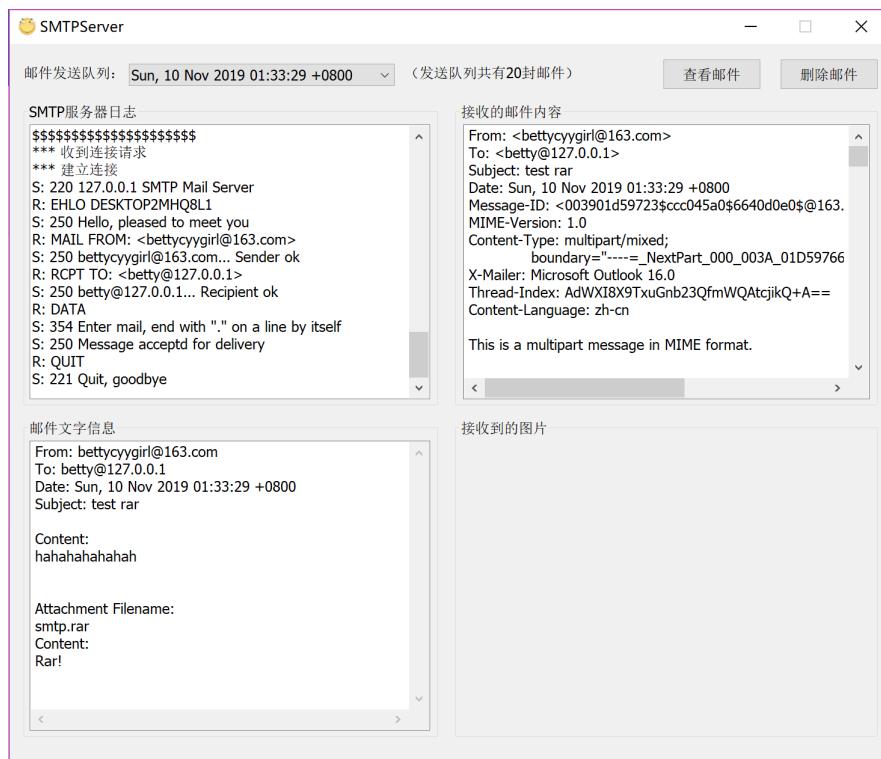


Figure 18: 服务器

6.7 发送多个附件



Figure 19: 客户端

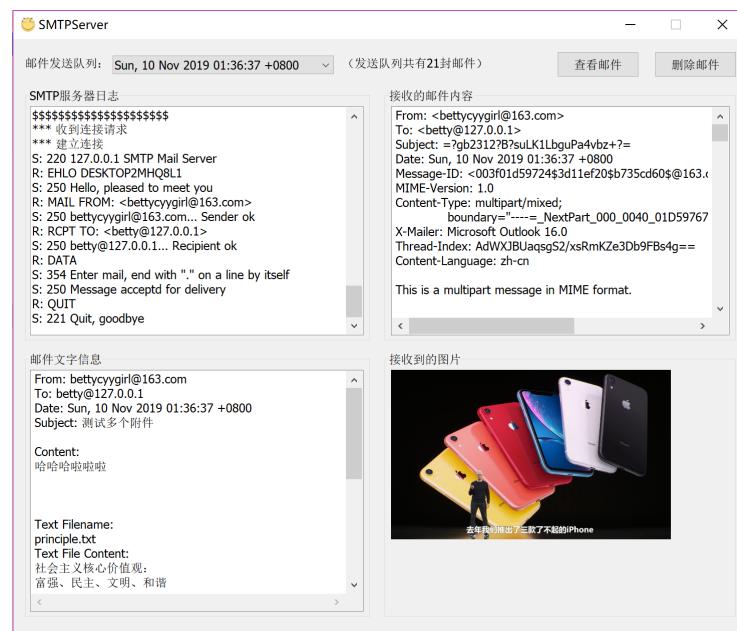


Figure 20: 服务器

6 程序演示

作业 2：编写 SMTP 服务器并观察通信过程

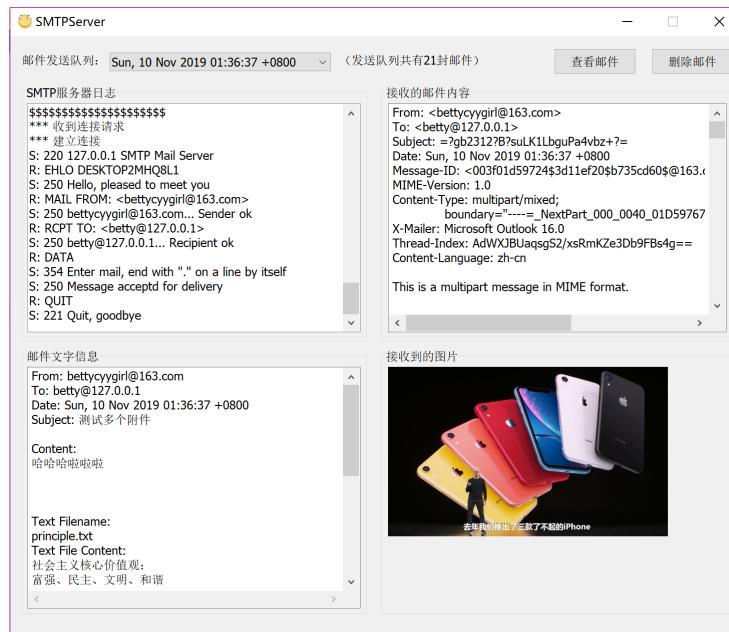


Figure 21: 服务器

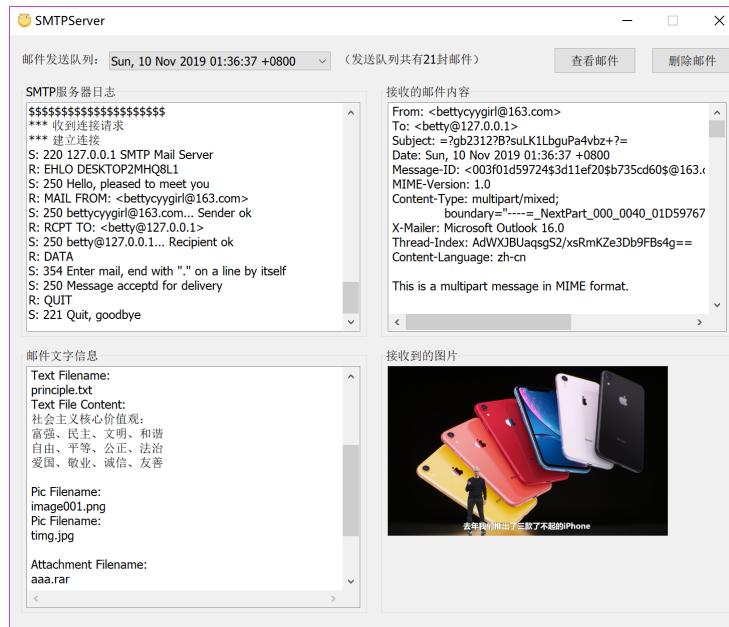
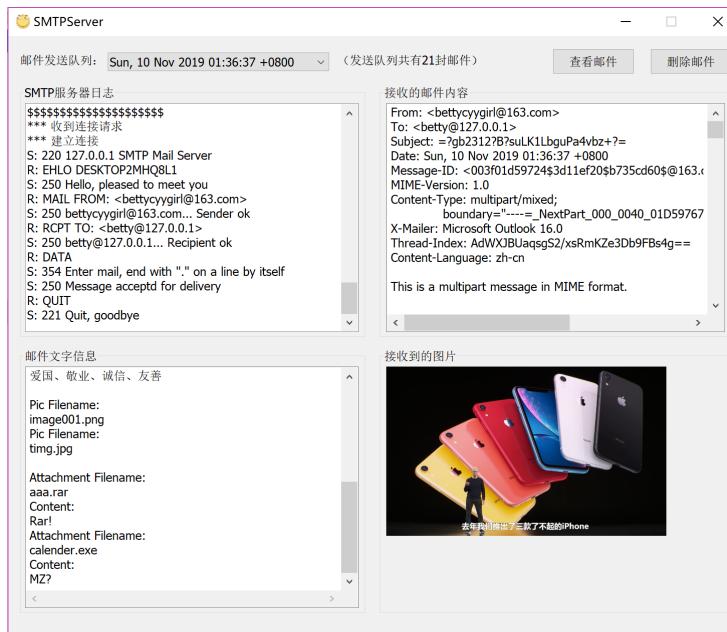
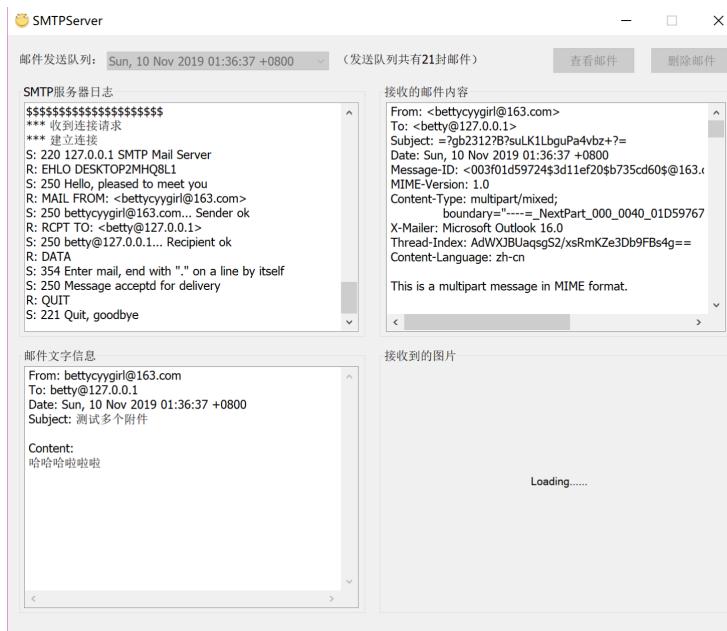


Figure 22: 服务器

作业 2：编写 SMTP 服务器并观察通信过程

**Figure 23:** 服务器

如果点击一下“查看邮件”，如果加载的速度较慢，则会在右下角出现一个 Loading.....，表明未加载完成。

**Figure 24:** 服务器

6.8 捕获数据包并查看通信过程

利用 Wireshark + Npcap 可以捕捉本地回环接口的数据包。如果服务器成功响应则会看到发送和接收的数据包信息。以下记录了一次完整的通信过程（以上述的第一次通信为例）：

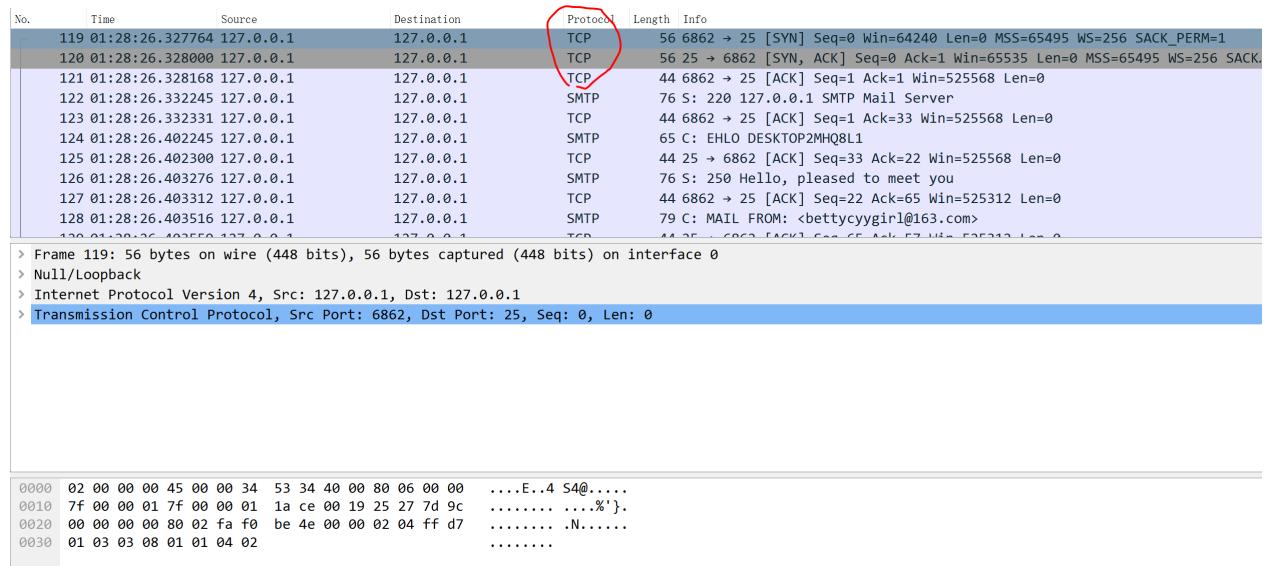


Figure 25: 通过三次握手建立客户端与服务器的 TCP 连接

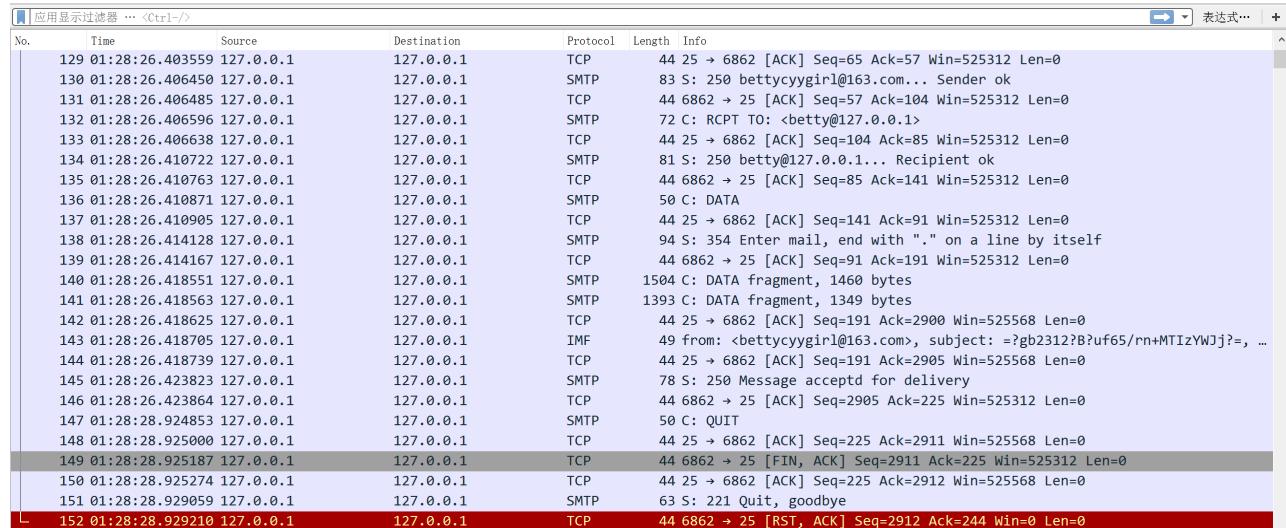


Figure 26: 通信过程

可以看到，日志中命令的部分和捕获到的包是一致的。此外也可以看出在发出一次消息后对方在响应之前至少先要发一个 ACK 确认，然后再发出应答消息。而且比较长的数据会分多次发（例如在发邮件数据的时候，涉及到 IP 数据报的分片）。