SMTP 邮件服务器实现并观察通信过程实验报告

李伟　1711350　2017 级计算机一班

摘要

我们经常使用邮件进行通信，通过网络课我们了解到邮件应用是基于 TCP 协议基础上进行数据通信的，通信过程也比较简单，就是基于三次握手建立 TCP 连接，然后进行数据通信的 SMTP 协议，因此可以尝试利用 MFC 和 CAsyncSocket 网络编程实现一个简单的 SMTP 服务器能够接收客户端应用程序发送来的邮件报文并进行解析呈现。本次邮件服务器实现的功能包括接收用户的邮件报文（课多次接收，记录历史数据），解析报文的邮件正文、图片附件、txt 文档附件，能够实现浏览历史邮件接收记录，同时还可以多个客户端同时发送，服务器也能够并发响应（会存在一定的卡顿）。

关键字： SMTP 协议　邮件服务器　并发响应　网络编程　邮件协议解析

目录

一、 实验任务 & 目的

1.1 实验目的：

编写一个简单的 SMTP 服务器，观察电子邮件应用查程序与 SMTP 服务器的命令交互过程。

1.2 实验要求：

- 实现的服务器应能够响应用户的 SMTP 命令，将命令的交互过程显示到屏幕上
- 支持单用户
- 不保存和转发收到的邮件
- 不做错误处理
- 在屏幕上显示接收到的邮件内容，包括对部分附件和邮件正文的解析
- 支持中文字符解析和显示

1.3 程序界面设计样例：

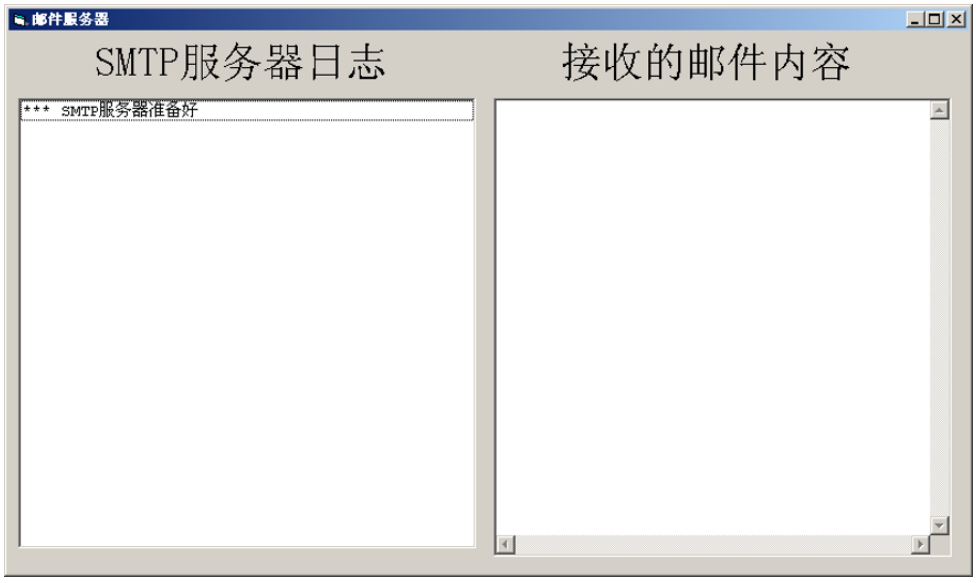

图 1　邮件服务器运行界面样例

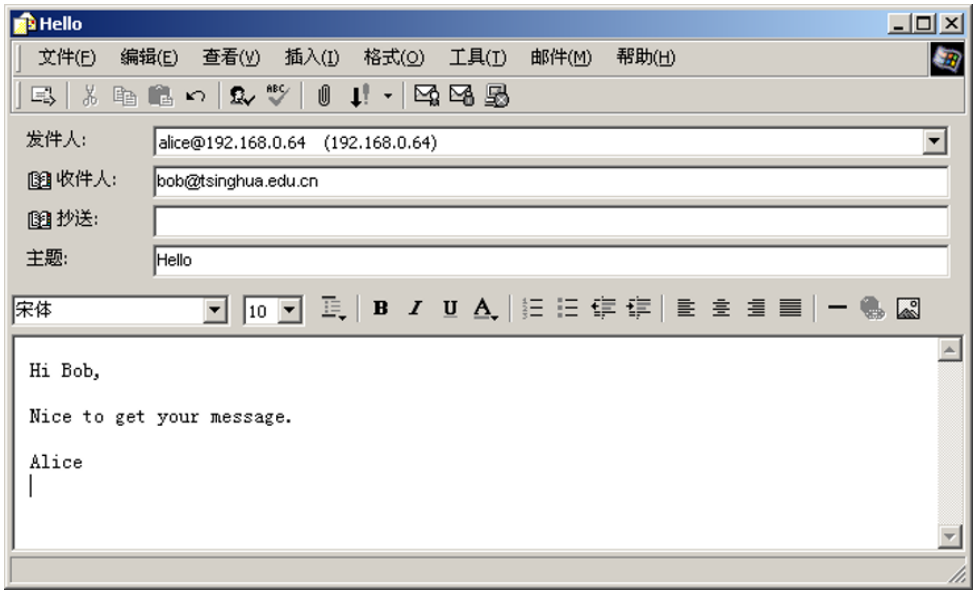

图 2　发送邮件样例

图 3　接收界面显示样例

二、 实验环境

- 系统环境：window10 专业版
- 邮件客户端：window10 自带的邮件客户端程序
- WireShark：Version
- IDE：Visual Studio 2015
- 调试环境：MFC release X86
- 编程语言：c++

三、 实验原理

3.1 SMTP 服务器工作原理

通过 WireShark 可以查看本机邮件收发过程中的数据通信过程，通过网络编程重现这个交互过程便能够实现一个 SMTP 服务器。SMTP 服务器的工作原理如下：

- SMTP 服务器绑定 25 号端口，并持续监听客户端的连接请求，客户端使用 TCP 协议向服务器端发送连接请求
- 服务器收到连接请求，回送 220 响应给客户端
- 客户端发送 HELO 命令，并声明自己的网络域
- 服务器端发送 250 响应并报送自己地址
- 客户端发送 MAIL FROM 命令，说明发件人，服务器确认后回送 250 响应
- 客户端发送 RECP TO 命令，说明收件人，服务器确认后回送 250 响应
- 客户端发送 DATA 命令说明准备开始发送数据，服务器回送 354 响应，统一开始接收数据
- 客户端发送数据结尾出现 <CR><LF>.<CR><LF>，说明数据发送结束，服务器确认后回送 250 命令，说明正确接收
- 客户端发送 QUIT 命令，说明需要拆除连接，服务器确认后回送 221 响应

3.2 MIME 邮件报文协议格式

邮件常用的报文协议为 MIME 协议，具体的格式如下所示：

域名	含义	添加者
Received	传输路径	各级邮件服务器
Return-Path	回复地址	目标邮件服务器
Delivered-To	发送地址	目标邮件服务器
Reply-To	回复地址	邮件的创建者
From	发件人地址	邮件的创建者
To	收件人地址	邮件的创建者
Cc	抄送地址	邮件的创建者
Bcc	暗送地址	邮件的创建者
Date	日期和时间	邮件的创建者
Subject	主题	邮件的创建者
Message-ID	消息 ID	邮件的创建者
MIME-Version	MIME 版本	邮件的创建者
Content-Type	内容的类型	邮件的创建者
Content-Transfer-Encoding	内容的传输编码方式	邮件的创建者

四、 实现步骤

4.1 数据交互分析

通过 163 邮箱发送邮件可以通过 WireShark 分析数据报文交互如下所示：

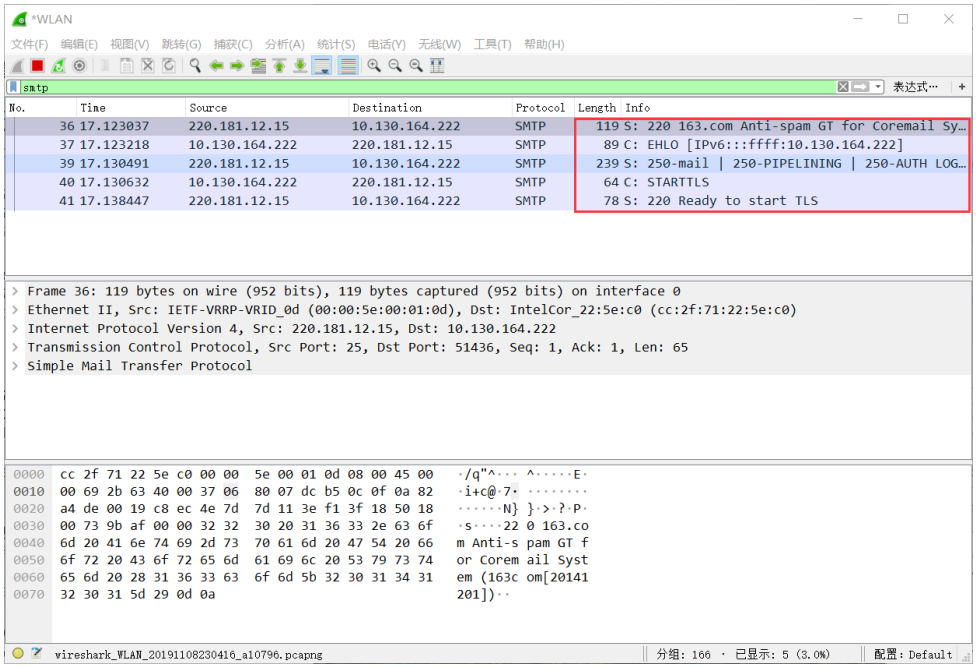

图 4　邮件发送 SMTP 交互过程

通过上面的截图可以清晰的看到，客户端通过三次握手和 SMTP 服务器端建立起数据通信连接。

4.2 邮件客户端设置

要想将本机设置为 SMTP 服务器，并将发出的邮件能够发送到本机的服务器端口，需要设置客户端的 SMTP 服务器地址设置。本实验使用的是系统自带邮件系统，设置界面如下图所示（192.168.56.1 为本机 IP 地址）：

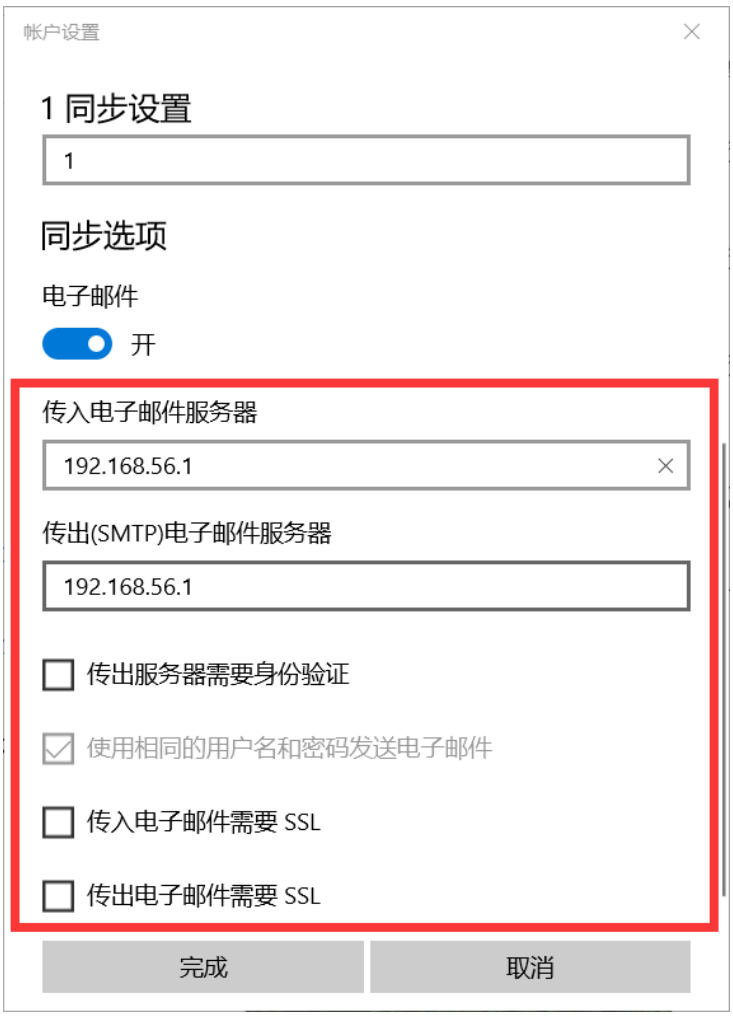

图 5　邮件客户端设置

4.3 SMTP 服务器实现

通过实验原理中对于邮件服务器响应的分析可知，SMTP 服务器实现的重点在于利用 TCP 协议编写好与客户端之间的交互命令和响应操作，需要实现对于客户端 HELO、DATA、MAIL FROM 、RCPT TO、QUIT、RESET、NOOP 命令的响应，同时编写好建立连接的函数。建立连接的代码如下：

```
void SMTP_Socket::OnAccept(int nErrorCode)
{
   // TODO: 在此添加专用代码和或调用基类/
   //获取主窗口句柄
   CMail_ServerDlg* Dlg = (CMail_ServerDlg*)AfxGetApp()->m_pMainWnd;
   //记录日志
   Dlg->log_list_ctrl.InsertString(Dlg->log_list_ctrl.GetCount(), ”收到连接请求***”);
```

```
    //创建负责连接的套接字
    SMTP_Socket * pSocket = new SMTP_Socket();
    if (Accept(*pSocket))
    {
        char* t = "220 Simple Mail Server Ready for Mail\r\n";
        pSocket->Send(t, strlen(t));
        pSocket->AsyncSelect(FD_READ);    //调用接收函数接收回应
        Dlg->log_list_ctrl.InsertString(Dlg->log_list_ctrl.GetCount(),
            "建立连
            接***");
        //记录日志
        Dlg->log_list_ctrl.InsertString(
            Dlg->log_list_ctrl.GetCount(),
            "S:220 Simple Mail Server Ready for Mai");
    }
    else
    {
        Dlg->log_list_ctrl.InsertString(Dlg->log_list_ctrl.GetCount(),
            "连接错
            误***");
    }
    CAsyncSocket::OnAccept(nErrorCode);
}
```

在 DLG 类中创建好了套接字之后，绑定 25 号端口，设置其为监听状态，之后通过调用 ACCEPT 函数接收客户端的连接请求，并创建一个新的套接字对象负责该链接的数据收发，实现并发的数据通信。数据通信的具体实现在重载的 OnReceive 函数中，具体的实现代码见附录 A 中的 SMTP 客户端交互实现核心代码，在接收完数据之后，将接收到的报文传到一个缓冲区中，并将最新报文显示到报文展示区中并发送消息触发报文解析函数对报文进行解析并呈现。

在实现完毕数据交互的核心功能之后开始进行报文解析工作，报文解析的核心在于 base64 编码的解码工作，这部分的代码在 base64.cpp 文件中，核心解码代码如下所示：

```
int get_index(char a) { //返回对应的数字
    if (a <= '9'&&a >= '0')
        return 52 + a - '0';
```

```
  else if (a <= 'Z'&&a >= 'A')
    return a - 'A';
  else if (a <= 'z'&&a >= 'a')
    return a - 'a' + 26;
  else if (a == '/')
    return 63;
  else if (a == '+')
    return 62;
  return 0;
}
char *base64_decode(const char *input, char *output)
{

  output[0] = '\0';
  if (input == NULL || output == NULL)
    return output;
  int input_len = strlen(input);
  if (input_len < 4 || input_len % 4 != 0)
    return output;

  // 0xFC -> 11111100
  // 0x03 -> 00000011
  // 0xF0 -> 11110000
  // 0x0F -> 00001111
  // 0xC0 -> 11000000
  char *p = (char*)input;
  char *p_out = output;
  char *p_end = (char*)input + input_len;
  for (; p < p_end; p += 4) {
    *(p_out++) = ((get_index(p[0]) << 2) & 0xFC) |
      ((get_index(p[1]) >> 4) & 0x03);
    *(p_out++) = ((get_index(p[1]) << 4) & 0xF0) |
      ((get_index(p[2]) >> 2) & 0x0F);
    *(p_out++) = ((get_index(p[2]) << 6) & 0xC0) |
      (get_index(p[3]));
```

```
    }

    if (*(input + input_len - 2) == '=') {
        *(p_out - 2) = '\0';
    }
    else if (*(input + input_len - 1) == '=') {
        *(p_out - 1) = '\0';
    }
    return output;
}
```

通过上述的代码，将报文中的 base64 编码段每 4 个字符作为一个单元处理成 3 个字符，之后相应的进行呈现即可，相关的业务代码在附录 B 中呈现，包括图片的展示。其中对于中文字符的显示使用了一个将 utf-8 编码字符串转为 ANSII 编码字符串的函数实现。

4.4 前端界面与交互设计

前端界面设计主要分为日志显示区、报文显示区、邮件内容解析展示区三个主要模块。具体页面效果如下：当程序启动之后，程序会尝试在 25 号端口创建套接字负责监听，如果成功则在日志显示区记录成功日志，否则则记录创建失败信息。

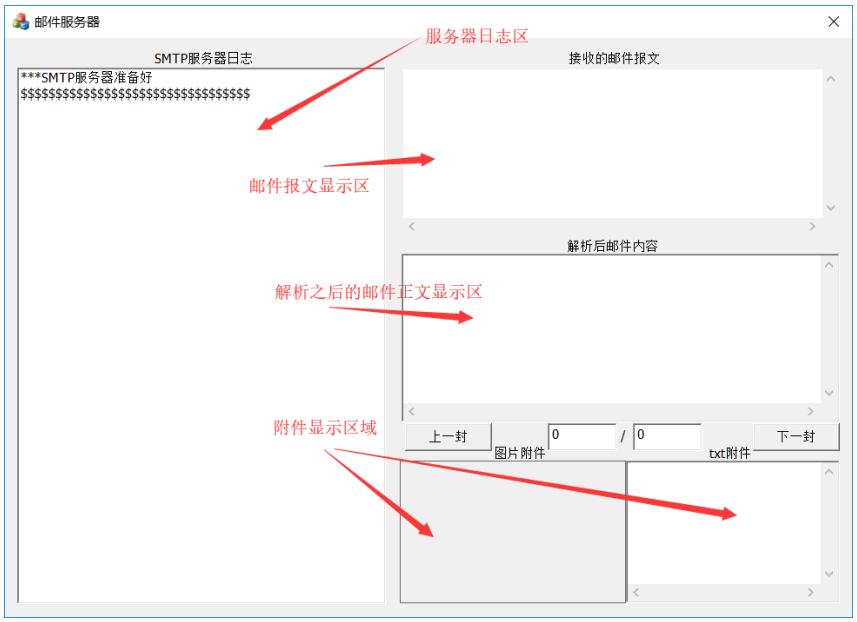

图 6　界面显示效果

五、 程序运行效果

本程序能够接收多个的邮件发送请求，并进行响应，在 OnReceive 函数中注释掉了一个 Sleep 函数，解除注释后可以测试程序的并发响应功能。其他的功能演示如下面的诸图所示：

图 7　发送邮件 1 界面

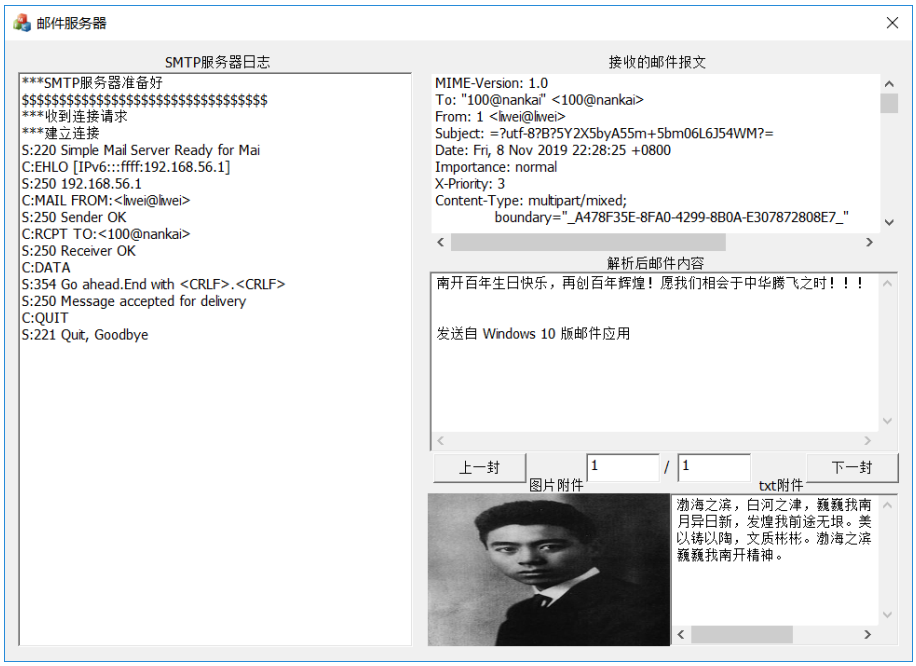

图 8　接收第 1 封邮件界面

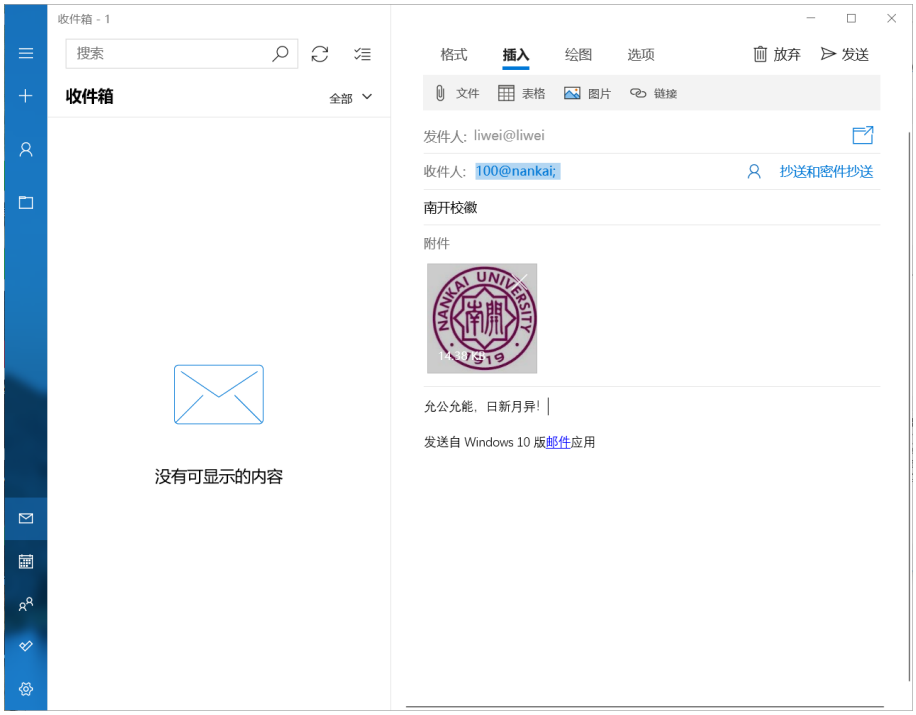

图 9　发送邮件 2 界面

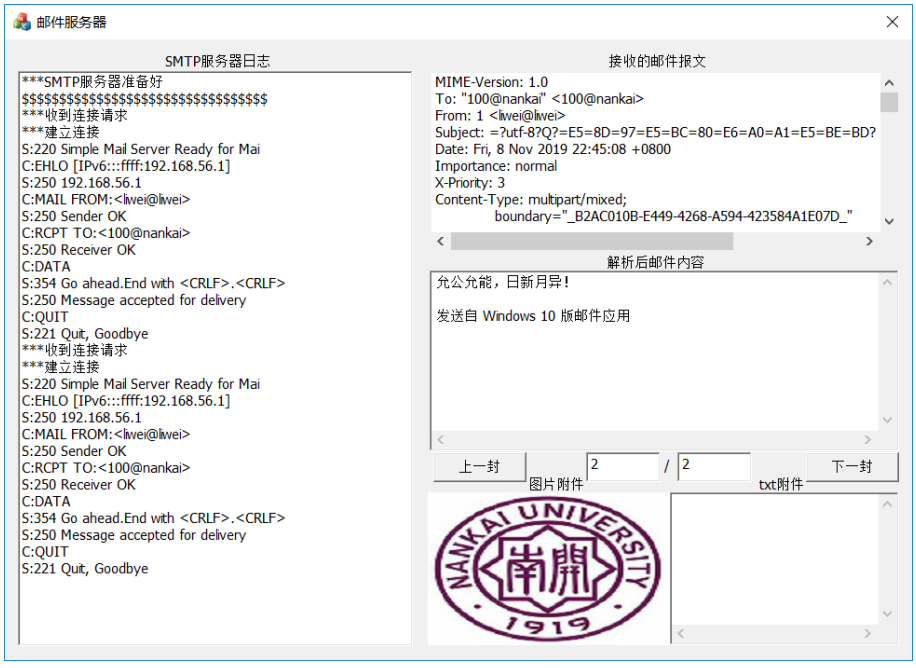

图 10　接收第 2 封邮件界面

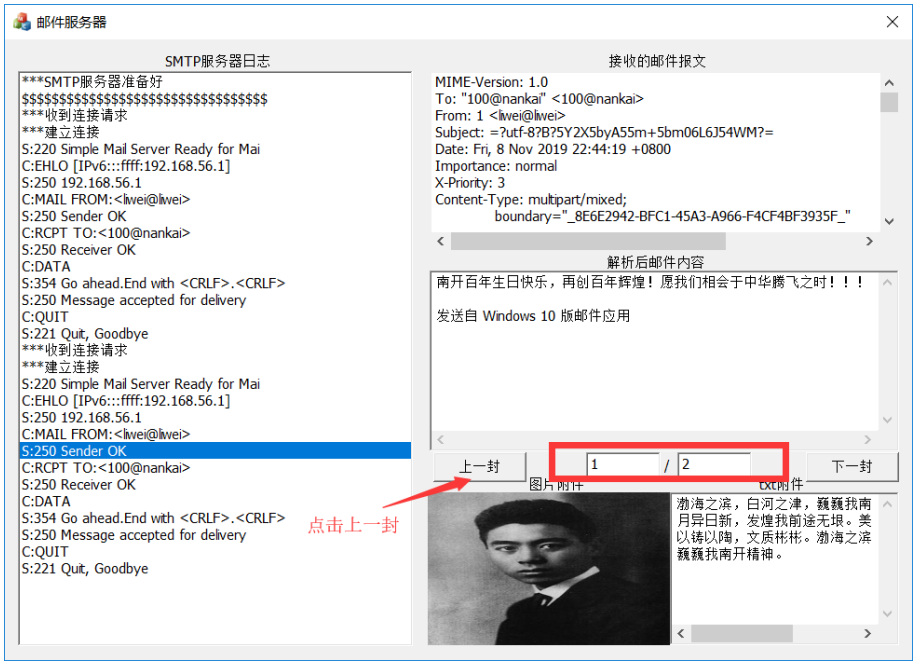

图 11　切换查看历史收到的邮件操作演示

六、 总结与思考

通过这次的 SMTP 服务器实现实验让我更深刻的理解了邮件收发的过程以及 TCP 网络编程的具体实现，同时通过学习邮件编码知识，让我对于编码在网络通信中的意义和作用有了更深刻的理解，同时通过对附件和图片的解析过程，对于数据包和各种类型的文件在网络中的传输本质有了更加清晰的认识。

在这一次的实验过程中，遇到了一些问题，但是也做出了一些探索和思考，比如在实现 base64 解码的过程中总是出现问题，后来查询资料后采用了移位合并，每 4 个字符作为单元处理的方式，从而更完美的解决了解码问题，除此之外在将中文字符的 utf-8 字符转为 MFC 能够显示的 ASCII 编码字符上遇到了很多问题，在查找网络资料之后，找到了两个函数能够实现编码格式的转换，最终实现了中文字符的显示功能。

附录 A　SMTP 客户端交互实现核心代码

```
void SMTP_Socket::OnReceive(int nErrorCode)
{
   // TODO: 在此添加专用代码和或调用基类/
   //获取主界面句柄
   CMail_ServerDlg* Dlg = (CMail_ServerDlg*)AfxGetApp()->m_pMainWnd;
   //接收报文
   int len = Receive(lbuf, sizeof(lbuf),0);
   if (len == -1)  //错误处理
   {
      Dlg->log_list_ctrl.InsertString(Dlg->log_list_ctrl.GetCount(),
         ”接收消息错
         误***”);
      return;
   }
   else if (len == 0)
   {
      Dlg->log_list_ctrl.InsertString(Dlg->log_list_ctrl.GetCount(),
         ”接收报文为
         空***”);
      return;
   }
   //转换格式
   CString recv_info =(CString)lbuf;
   //如果没有开始接收数据，输出接收到的命令并进行响应
   if (!begin_data_recv)
   {
      Dlg->log_list_ctrl.InsertString(Dlg->log_list_ctrl.GetCount(),
         ”C:” + recv_info.Left(len));
      if (recv_info.Left(4) == ”EHLO”)
      {
         //发送回应
         char *ret= ”250 192.168.56.1\r\n”;
         Send(ret, strlen(ret));
         //触发接收报文响应函数
```

```
        AsyncSelect(FD_READ);
        //记录日志
        Dlg->log_list_ctrl.InsertString(
          Dlg->log_list_ctrl.GetCount(),
          "S:" + (CString)ret);
        return;
      }
      else if (recv_info.Left(4)== "NOOP")
      {
        char *ret = "250 OK No Operation\r\n";
        Send(ret, strlen(ret));
        //触发接收报文响应函数
        AsyncSelect(FD_READ);
        //记录日志
        Dlg->log_list_ctrl.InsertString(
          Dlg->log_list_ctrl.GetCount(),
          "S:" + (CString)ret);
        return;
      }
      else if (recv_info.Left(10) == "MAIL FROM:")
      {
        char *ret = "250 Sender OK\r\n";
        Send(ret, strlen(ret));
        //触发接收报文响应函数
        AsyncSelect(FD_READ);
        //记录日志
        Dlg->log_list_ctrl.InsertString(
          Dlg->log_list_ctrl.GetCount(),
          "S:" + (CString)ret);
        return;
      }
      else if ( recv_info.Left(8) == "RCPT TO:")
      {
        char *ret = "250 Receiver OK\r\n";
        Send(ret, strlen(ret));
```

```
        //触发接收报文响应函数
        AsyncSelect(FD_READ);
        //记录日志
        Dlg->log_list_ctrl.InsertString(
            Dlg->log_list_ctrl.GetCount(),
            "S:" + (CString)ret);
        return;
    }
    else if (recv_info.Left(4) == "QUIT"||recv_info.Left(4)==
        "RSET")
    {

        char *ret = "221 Quit, Goodbye\r\n";
        Send(ret, strlen(ret));
        //记录日志
        Dlg->log_list_ctrl.InsertString(
            Dlg->log_list_ctrl.GetCount(),
            "S:" + (CString)ret);
        //关闭连接
        this->Close();
        return;
    }
    else if (recv_info.Left(4) == "DATA")
    {
        if (!begin_data_recv)
        {
            //开始接收数据，设置标志位
            begin_data_recv = true;
            //清空当前文本区域内的文本空间
            Dlg->mail_text_ctrl.SetWindowTextA("");
        }
        //发送响应
        char *ret = "354 Go ahead.End with <CRLF>.<CRLF>\r\n";
        Send(ret, strlen(ret));
        //触发接收报文响应函数
```

```
99          AsyncSelect(FD_READ);
100         //记录日志
101         Dlg->log_list_ctrl.InsertString(
102            Dlg->log_list_ctrl.GetCount(),
103            "S:" + (CString)ret);
104         return;
105      }
106      else
107      {
108         char *ret = "500 order is wrong\r\n";
109         Send(ret, strlen(ret));
110         //触发接收报文响应函数
111         AsyncSelect(FD_READ);
112         //记录日志
113         Dlg->log_list_ctrl.InsertString(
114            Dlg->log_list_ctrl.GetCount(),
115            "S:" + (CString)ret);
116         //关闭连接
117         this->Close();
118         return;
119      }
120   }else   //接收数据报文并显示在邮件显示区
121   {
122      //接收多个连接，将相应的数据先独立存好
123      recv_post += recv_info.Left(len);
124
125      if (recv_info.Find("\r\n.\r\n") != -1)
126      {
127         //结束接收数据
128         begin_data_recv = false;
129         //将本次连接接收的报文输出到显示报文的文本框中
130         Dlg->mail_text_ctrl.SetWindowTextA(recv_post);
131
132         //延迟发送确认收到消息，用于测试程序的并发通信//
133         //srand((unsigned)time(NULL));
```

```
        //if((rand()%15)<3) 随机延迟//
        // Sleep(10000);

        //发送数据接收结束的响应
        char *ret = "250 Message accepted for delivery\r\n";
        Send(ret, strlen(ret));
        //记录日志
        Dlg->log_list_ctrl.InsertString(
          Dlg->log_list_ctrl.GetCount(),
          "S:" + (CString)ret);
        //将报文传入全局的报文缓存队列
        mail_t* cur_mail = new mail_t;
        Dlg->mail_text_ctrl.GetWindowTextA(cur_mail->mail_post);
        Dlg->mail_list.AddTail(cur_mail);
        //更新接收邮件数目
        CString cur;
        cur.Format("%d", Dlg->mail_list.GetCount());
        Dlg->cur_ctrl.SetWindowTextA(cur);
        Dlg->total_ctrl.SetWindowTextA(cur);
        Dlg->UpdateData(true);
        //发送显示邮件内容消息，让主线程解析邮件，进行内容显示
        Dlg->PostMessage(WM_DISPLAYMAIL,0,0);
      }
      //触发接收报文响应函数
      AsyncSelect(FD_READ);
    }
    //memset(lbuf, 0, sizeof(lbuf) / sizeof(char));
    CAsyncSocket::OnReceive(nErrorCode);
}
```

附录 B　解析邮件报文处理函数

```
// 解析邮件报文的消息处理函数
```

```
afx_msg LRESULT CMail_ServerDlg::OnDisplayMail(WPARAM wParam,
    LPARAM lParam)
{
   if (mail_text_ctrl.GetLineCount() == 0)
      return afx_msg LRESULT();  //报文显示区域为空，不做解析处理
   //解析结果将新的内容填充之前需要先清空原文件内容
   mail_display_ctrl.SetWindowTextA("");
   txt_ctrl.SetWindowTextA("");
   //设置空间的图片为空
   ((CStatic*)GetDlgItem(IDC_picture))->SetBitmap(NULL);
   UpdateData(true);
   //获取报文文本
   CString mail_post;
   mail_text_ctrl.GetWindowTextA(mail_post);
   //获取对应的编码文本
   int start_base64_code = 0;
   while (1) {
      start_base64_code = mail_post.Find("Content-Transfer-Encoding:
         base64", start_base64_code);
      if (start_base64_code == -1)
      {
         break;
      }
      int text_type_begin = mail_post.Find("\r\n",
         start_base64_code);
      int code_begin = mail_post.Find("\r\n\r\n",
         text_type_begin+2);//找到编码开始位
         置base64
      int code_end = mail_post.Find("\r\n\r\n", code_begin+4);//找到图
         片附件编码结束位置
      CString code= mail_post.Mid(code_begin + 4, code_end -
         code_begin - 4); // 截断编
         码;
      while (1) {
         int pos = code.Find("\r\n");
         if (pos == -1) break;
```

```
        CString a = code.Mid(0, pos);
        CString b = code.Mid(pos + 2, code.GetLength()-2);
        code = a + b;
    }
    CString type = mail_post.Mid(text_type_begin + 2, code_begin -
        text_type_begin - 2);
    if (type.Find("attachment") != -1)  //处理附件
    {
        int p1 = type.Find('"', 0);
        int p2 = type.Find('"', p1 + 1);
        CFile file;
        CString file_name = type.Mid(p1 + 1, p2 - p1 - 1);
        //log_list_ctrl.InsertString(log_list_ctrl.GetCount()打开附件:
            ,""+file_name);
        file.Open(file_name, CFile::modeWrite | CFile::modeCreate|
            CFile::typeBinary);
        char *output = new char[code.GetLength() * 3 / 4];
        base64_decode(code, output);
        file.Write(output, code.GetLength() * 3 / 4);
        file.SeekToEnd();
        file.Close();
        if (type.Find("txt") != -1)
        {
            txt_ctrl.SetWindowTextA(output);
        }
        else if(type.Find("jpg")!=-1||
            type.Find("bmp") != -1||
            type.Find("PNG") != -1||
            type.Find("png") != -1||
            type.Find("BMP") != -1||
            type.Find("JPG") != -1) {

            CImage image;
            image.Load(file_name);

```

```
62          //以下两个矩形主要作用是，获取对话框上面的Picture 的和，
               Controlwidthheight
63          //并设置到图片矩形，根据图片矩形对图片进行处理，
               rectPicturerectPicture
64          //最后绘制图片到对话框上Picture 上面Control
65          CRect rectControl;                  //控件矩形对象
66          CRect rectPicture;                  //图片矩形对象
67
68
69          int x = image.GetWidth();
70          int y = image.GetHeight();
71          //Picture 的为ControlIDIDC_picture
72          CWnd *pWnd = GetDlgItem(IDC_picture);
73          pWnd->GetClientRect(rectControl);
74
75
76          CDC *pDc = GetDlgItem(IDC_picture)->GetDC();
77          SetStretchBltMode(pDc->m_hDC, STRETCH_HALFTONE);
78          //重新设置图片的大小为空间的规格
79          rectPicture = CRect(rectControl.TopLeft(),
               CSize((int)rectControl.Width(),
               (int)rectControl.Height()));
80          //设置空间的图片为空
81          ((CStatic*)GetDlgItem(IDC_picture))->SetBitmap(NULL);
82
83          //以下两种方法都可绘制图片，这里是直接在控件的上面贴图，能够实现
               相同的功能
84          image.StretchBlt(pDc->m_hDC, rectPicture, SRCCOPY); //将图
               片绘制到控件表示的矩形区域Picture
85          //image.Draw(pDc->m_hDC, rectPicture); 将图片绘制
               到      //控件表示的矩形区域Picture
86
87          image.Destroy();
88          pWnd->ReleaseDC(pDc);
89          UpdateData(true);
90      }
91  }
```

```
92      else if (type.Find(”charset=”) != -1) //处理正文文本
93      {
94         char *output = new char[code.GetLength() * 3 / 4];
95         output = base64_decode(code, output);
96         mail_display_ctrl.SetWindowTextA(UTF8toANSI(CString(output)));
              //显示编
              码
97      }
98      start_base64_code = code_end;
99   }
100  return afx_msg LRESULT();
101 }
```