上机作业 2：编写 SMTP 服务器并观察通信过程

1711342 李纪

2019 年 11 月 8 日

摘要

这是我的**上机作业 2** 的实验报告，请老师查阅，谢谢老师。

程序主要特点：支持多国语言显示（包括但不限于中法德意俄英日韩等各国语言）。支持多格式图片（bmp, jpg, png, gif 等）。支持多附件接收并打开。

为了便于理解和确保正确性，附录的理论及标准介绍部分采用英文。

关键字：SMTP、Base64、MIME

目录

1 实验的目的

SMTP[1]和 POP3[2]是目前电子邮件应用系统中最重要的两个协议。深入了解 SMTP 和 POP3 的工作过程对理解整个电子邮件服务系统具有重要的意义。本实验要求编写一个简化的 SMTP 邮件服务器，通过观察电子邮件应用程序（如 Foxmail 等）与 SMTP 邮件服务器的交互过程 · 加深对整个邮件服务系统的理解。

2 实验基本要求

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。

3 实验方案和方法的选择

实验方案和方法参考《网络技术与应用》[4]。

3.1 编写简化的 SMTP 服务器

为了观察电子邮件应用程序（如 Foxmail）与 SMTP 邮件服务器的交互过程。可以使用 VC.NET 提供的 CAsyncSocket 类编写一个简化的 SMTP 服务器。为了简化程序的编写，该邮件服务器在同一时刻仅支持一个用户发送邮件。它既不保存收到的邮件，也不转发收到的邮件，甚至不作错误处理。简化的 SMTP 邮件服务器仅响应电子邮件应用程序发出的 SMTP 命令，并将命令的交互过程和收到的电子邮件显示到屏幕上。

SMTP 服务器在 TCP 的 25 端口守候，等待邮件应用程序发出命令。因此，在简化的 SMTP 邮件服务的编写过程中需要用到流式套接字。需要利用 CAsyncSocket 类编写流式套接字。

[1] **简单邮件传输协议**（英语：**S**imple **M**ail **T**ransfer **P**rotocol，缩写：**SMTP**）是一个在互联网上传输电子邮件的标准。首次推出由 RFC821(1982 年) 定义，最新且使用广泛的协议标准基于 RFC5321(2008 年)

[2] **邮局协议**（英语：**P**ost **O**ffice **P**rotocol，缩写：**POP**）是 TCP/IP 协议族中的一员，由 RFC 1939 定义。此协议主要用于支持使用客户端远程管理在服务器上的电子邮件。最新版本为 **POP3**，全名 "Post Office Protocol - Version 3"，而提供了 SSL 加密的 POP3 协议被称为 **POP3S**。

3.2 观察 SMTP 客户与服务器的交互过程

为了观察 SMTP 客户与服务器的交互过程，首先需要在网络的一台主机上（如 192.168.1.161）启动编写的 SMTP 服务器。然后，可以在网络的另一台主机上启动电子邮件应用程序（如 Foxmail），并创建一个 SMTP 服务器指向 192.168.1.161 的账号，由于仅关心 SMTP 客户与服务器的交互过程，因此该账号的账号名、POP3 服务器的地址等都可以任意填写。

一旦账号创建完成，就可以撰写一封电子邮件。如果编写的简化 SMTP 服务器程序运行正确，在该邮件发送后就可以在客户端看到“发送成功”的界面。

4 实验过程

先理解实验中涉及到的原理（比如 SMTP、Base64 以及 MIME），然后根据正确配置邮箱客户端（Foxmail）环境，之后开始写入源代码。

SMTP 的相关知识点在附录 A 中解释。

Base64 的相关知识点在附录 B 中解释。

MIME 的相关知识点在附录 C 中解释。

4.1 开发及测试环境

目前仅能保证程序在开发及测试环境中正常工作，**其他情况均不能保证程序能够稳定运行**。

测试环境如下：

- 操作系统：Microsoft Windows 10 专业版 (x64) (version: 17763)
- IDE：Visual Studio 2019 (version: 16.3.5)
- Windows SDK version: 10.0
- 平台工具集：Visual Studio 2019 (v142)
- C++ 语言标准：std:C++17

使用的邮箱客户端：**Foxmail 7.2 (build 13.365)**

5 关键部分源代码

（"..." 部分表示省略）

此部分分为 4 个小节，第一个小节展示函数和变量的声明，后三个小节每个小节解释一个具体功能的实现。每小节开头都有**代码摘要**，粗略地展示了功能的实现步骤。

代码解释大部分可以根据代码里的注释来解释说明，请老师查阅。为了方便解释与理解，代码之间的联系与具体实现请直接通过 Visual Studio 查看源代码。

5.1 头文件（函数和变量的声明）

代码摘要：函数和变量的声明。

因为很多变量都在头文件中声明，所以为了便于理解，首先贴上 Socket 类及对话框类头文件的代码。

```
class ListenServer : public CAsyncSocket
{
public:
  ListenServer();
  virtual ~ListenServer();
  virtual void OnAccept(int nErrorCode);
  virtual void OnClose(int nErrorCode);

  list<ServerSocket*> socketPtrList; // 多用户，多进程维护列表
};

class ServerSocket : public CAsyncSocket
{
public:
  ServerSocket();
  virtual ~ServerSocket();
  void CheckSymbol(char& c); // 将 base64 编码后的字符解码。
  CString base64_decode_picture(string attachmentName,string encoded_string)
    ; // base64 图片解码
  string base64_decode(string encoded_string); // base64 解码
  virtual void OnReceive(int nErrorCode);
  bool data = false; // 判断是否在传输数据的标志 0->否 1->是
  map<char, int> base64_map; // base64 解码表
  queue<CString> imageList; // 存储要打开的图片
};

class CSMTPDlg : public CDialogEx
{
...
```

```
29 public:
30   // 显示函数（一个能在编辑框里显示文本的万能函数）
31   void displayString(CEdit& editCtrl, CString& str);
32   // 附件名
33   CEdit m_attachmentName;
34   // 日志
35   CListBox m_log;
36   // 邮件报文
37   // CEditCtrl m_messege;
38   // 附件图片
39   CStatic m_picture;
40   // TCP 侦听 socket
41   ListenServer listen;
42   afx_msg void OnBnClickedStart();
43   afx_msg void OnBnClickedClear();
44   // 判断是否开始监听
45   bool isListen = false;
46   // 清除所有输出
47   CButton m_clear;
48   // 开始/关闭服务器
49   CButton m_start;
50   // 邮件报文
51   CEdit m_message;
52   // Mail Text
53   CEdit m_text;
54   // 如果附件中有文本文件，则在这里显示
55   CEdit m_attachmentText;
56   // 打开图片按钮，如果需要打开多张图，必须在收信前按下此按钮
57   CButton m_pictureButton;
58   // 是否需要打开图片的 flag
59   bool openPictures = false;
60   afx_msg void OnBnClickedButtonpicture();
61 };
```

Listing 1: 头文件代码

5.2 功能：服务器初始化

代码摘要：单击 Start 按钮，设置服务器的 TCP 监听端口，初始化服务器。

1. CSMTPDlg::OnBnClickedStart()
   - 更新监听状态
   - 更新日志
   - 监听 Socket 绑定 TCP 端口 25

```
void CSMTPDlg::OnBnClickedStart()
{
  // 如果不在监听状态
  if (!isListen)
  {
    // 更新日志
    m_log.InsertString(-1,(L"SMTP Server starts. \n"));

    // 绑定 TCP 端口 25
    BOOL bFlag = listen.Create(25, SOCK_STREAM, FD_ACCEPT);

    if (!bFlag)
    {
      // 如果创建 Socket 报错，执行下面的代码
      int error = listen.GetLastError();
      AfxMessageBox(_T("Listen Socket Create Failed. Error: " + error));
      listen.Close();
      return;
    }

    // 开始监听
    if (!listen.Listen())
    {
      // 如果 Socket 监听报错，执行下面的代码
      int error = listen.GetLastError();
      AfxMessageBox(_T("Listen Socket Listen Failed. Error: " + error));
      listen.Close();
      return;
    }
    else
    {
      // 更新日志
      m_log.InsertString(-1,(L"TCP Socket listening now. At Port: 25"));
    }

    // 更新监听状态
    isListen = true;
    // 更新控件显示
    m_start.SetWindowTextW(L"Shut down");
  }
  // 如果在监听状态
  else
  {
    // 关闭 Socket
    listen.Close();
    // 更新日志
    m_log.InsertString(-1,(_T("Server shut down.\r\n")));
    // 更新控件显示
    m_start.SetWindowTextW(L"Start");
    // 更新监听状态
    isListen = false;
```

```
  }
}
```

Listing 2: 服务器初始化

5.3 功能：结束所有维护的线程

代码摘要：关闭窗口的同时会触发 void ListenServer::OnClose(int nErrorCode) 函数，结束所有维护的线程。

1. void ListenServer::OnClose(int nErrorCode)
    - 结束所有维护的线程

```
void ListenServer::OnClose(int nErrorCode)
{
  // 清空 Socket 列表，释放空间
  while (!socketPtrList.empty())
  {
    ServerSocket* socket = socketPtrList.back();
    socket->Close();
    delete socket;
    socketPtrList.pop_back();
  }
  CAsyncSocket::OnClose(nErrorCode);
}
```

Listing 3: 结束所有维护的线程

5.4 功能：初始化 SMTP 服务线程

代码摘要：服务器的 TCP 监听端口收到客户端请求后，初始化服务器 SMTP 服务线程。

1. void ListenServer::OnAccept(int nErrorCode)
    - 创建一个新的服务器 SMTP Socket
    - 将 SMTP Socket 指针加入服务器指针列表
    - 准备接收客户端请求

```cpp
void ListenServer::OnAccept(int nErrorCode)
{
  CSMTPDlg* pdlg = (CSMTPDlg*)AfxGetApp()->GetMainWnd(); // 获取指向主对话框的指针
  ServerSocket* socket = new ServerSocket(); // 创建一个新的服务器 Socket

  if (Accept(*socket)) // Accepts a connection on the socket.
  {
    char buffer[100] = "220 ready\r\n";
    socket->Send(buffer, strlen(buffer));
    pdlg->m_log.InsertString(-1, (L"S: 220 Ready For Mail Service"));

    // 投递一个“读“的事件，准备接收
    socket->AsyncSelect(FD_READ);

    // 将指针加入服务器指针列表
    list<ServerSocket*>::iterator it = socketPtrList.begin();
    socketPtrList.insert(it, socket);
  }
  // 如果连接出错，输出错误代码
  else
  {
    int error = socket->GetLastError();
    AfxMessageBox(_T("Accept error: ") + error);
    // 关闭 Socket
    socket->Close();
    delete socket;
  }

  CAsyncSocket::OnAccept(nErrorCode);
}
```

Listing 4: 初始化 SMTP 服务线程

5.5 功能：根据 SMTP 协议通信

代码摘要：服务器回复客户端的过程。

1. void ServerSocket::OnReceive(int nErrorCode)
    - 根据 SMTP 协议通信

```cpp
void ServerSocket::OnReceive(int nErrorCode)
{
  CSMTPDlg* pdlg = (CSMTPDlg*)AfxGetApp()->GetMainWnd(); // 获取指向主对话框的指针
  UINT revLength = Receive(buffer, sizeof(buffer)); // 获取客户端的回复

```

```
6   buffer[revLength] = '\0'; // 获取到的数据的后一位换成字符串结束符 '\0', 不
      然消息显示会乱码
7   CString str(buffer); // 生成一个 CString 类型的缓存
8
9
10  // 下面的代码展示了服务器如何回复客户端
11  if ((str.Left(4) == "EHLO") || (str.Left(4) == "HELO"))
12  {
13    pdlg->m_log.InsertString(-1,(L"C: " + str));
14    char reply[100] = "250 hello\r\n";
15    int s = Send(reply, strlen(reply));
16    // int error = GetLastError();
17    // AfxMessageBox(_T("Accept error: ") + error);
18    AsyncSelect(FD_READ);
19    pdlg->m_log.InsertString(-1,(L"S: 250 localhost"));
20  }
21  if (str.Left(10) == "MAIL FROM:")
22  {
23    pdlg->m_log.InsertString(-1,(L"C: " + str));
24    char reply[100] = "250 OK\r\n";
25    Send(reply, strlen(reply), 0);
26    AsyncSelect(FD_READ);
27    pdlg->m_log.InsertString(-1,(L"S: 250 OK"));
28  }
29  if (str.Left(8) == "RCPT TO:")
30  {
31    pdlg->m_log.InsertString(-1,(L"C: " + str));
32    char reply[100] = "250 OK\r\n";
33    Send(reply, strlen(reply), 0);
34    AsyncSelect(FD_READ);
35    pdlg->m_log.InsertString(-1,(L"S: 250 OK"));
36  }
37  if (str.Left(4) == "DATA")
38  {
39    pdlg->m_log.InsertString(-1,(L"C: " + str));
40    char reply[100] = "354 Go ahead\r\n";
41    Send(reply, strlen(reply), 0);
42    AsyncSelect(FD_READ);
43    pdlg->m_log.InsertString(-1,(L"S: 354 Go ahead"));
44
45    // 标志着在下一次服务器的 Receive 中, 客户端会传输数据过来
46    data = true;
47  }
48
49  // 客户端传输的数据到来的时候
50  if (data == true && str.Left(4) != "DATA")
51  {
52      // 下文会介绍这一段代码, 这里先省略。
53    ...
54  }
55  if (buffer[revLength-3] == '.')
```

```
  {
    pdlg->m_log.InsertString(-1, (L"C: " + str));
    char reply[100] = "250 OK\r\n";
    Send(reply, strlen(reply), 0);
    AsyncSelect(FD_READ);
    pdlg->m_log.InsertString(-1, (L"S: 250 OK\r\n"));
  }
  if (str.Left(4) == "QUIT")
  {
    pdlg->m_log.InsertString(-1, (L"C: " + str));
    char reply[100] = "221\r\n";
    Send(reply, strlen(reply), 0);
    AsyncSelect(FD_READ);
    pdlg->m_log.InsertString(-1, (L"S: 221"));
  }


  // 刷新对话框界面
  pdlg->UpdateData();
  pdlg->Invalidate();
  pdlg->UpdateWindow();
  CAsyncSocket::OnReceive(nErrorCode);
}
```

Listing 5: 根据 SMTP 协议通信

5.6 功能：处理接收到的数据

代码摘要：处理客户端发到服务器的原生邮件报文。

1. void ServerSocket::OnReceive(int nErrorCode)
   - 按类型解码不同类的文件，比如邮件正文部分、文本型附件部分、图片型附件等
   - 在服务器中显示原生报文和解码后的内容

```
void ServerSocket::OnReceive(int nErrorCode)
{
    ...
  // 客户端传输的数据到来的时候
  if (data == true && str.Left(4) != "DATA")
  {
    // 显示未解码的邮件内容（即显示原生邮件报文）
    pdlg->displayString(pdlg->m_message,str);

    // 邮件正文部分：
    // 找到邮件正文区
```

```
    string s(buffer);
    size_t pos1 = s.find("Content-Type: text/plain;");
    size_t pos2 = s.find("Content-Transfer-Encoding: base64", pos1);
    string temp = "Content-Transfer-Encoding: base64\r\n\r\n";
    pos2 += temp.length();
    size_t pos3 = s.find("\r\n\r\n", pos2);
    string truncated_str2 = s.substr(pos2, pos3 - pos2);

    // 解码邮件正文内容
    string decoded2 = base64_decode(truncated_str2);

    // 输出邮件正文解码后的内容
    CString cstr2(decoded2.c_str());
    pdlg->displayString(pdlg->m_text,cstr2);



    // 附件部分:
    // 如果有文本或图片类型的附件
    //(文本型仅支持 txt 类型文件 (ANSI 编码))
    //(图片型仅支持 bmp 类型文件)
    // 若同名文件, 则仅接收第一个文件 (map 容器的限制)

    // 先获取附件的名称和编码
    map<string, string>attachment;
    while (pos2 != s.npos)
    {
      // 获取附件的名称
      pos2 = s.find("Content-Disposition: attachment;\r\n\tfilename=\"",pos2
    );
      if (pos2 == s.npos)
        break;
      pos3 = s.find("\"\r\n\r\n", pos2);
      temp = "Content-Disposition: attachment;\r\n\tfilename=\"";
      pos2 += temp.length();
      truncated_str2 = s.substr(pos2, pos3 - pos2);
      string attachmentName = truncated_str2;

      // 获取附件的 base64 编码
      temp = "\"\r\n\r\n";
      pos3 += temp.length();
      size_t pos4= s.find("\r\n\r\n", pos3);
      truncated_str2 = s.substr(pos3, pos4 - pos3);
      string attachmentEncode = truncated_str2;

      // 将得到的名称-编码对加入 map 映射
      attachment.insert(pair<string, string>(attachmentName,
    attachmentEncode));

      // 输出附件列表, 键重复的只会列出第一个 (不支持同名文件传入)
      CString cstr1(attachmentName.c_str());
```

```
61      pdlg->displayString(pdlg->m_attachmentName, cstr1 + _T("\r\n"));
62    }
63
64    // 处理附件 map 映射中的数据
65    while (!attachment.empty())
66    {
67      string attachmentName = attachment.begin()->first;
68      string attachmentEncode = attachment.begin()->second;
69
70      // 检测文件后缀名
71      size_t pos5 = attachmentName.find(".");
72      string sub = attachmentName.substr(pos5 + 1, attachmentName.npos);
73      if (sub == "txt")
74      {
75        // 解码文本型附件内容（仅支持编码为 ANSI 的文本型附件）
76        string decoded2 = base64_decode(attachmentEncode);
77
78        // 输出文本型附件解码后的内容
79        decoded2 = attachmentName + " 的内容如下：\r\n
    ————————————————————\r\n\r\n" + decoded2;
80        CString cstr2(decoded2.c_str());
81        pdlg->displayString(pdlg->m_attachmentText, cstr2);
82      }
83      else
84      {
85        // 读取图片
86        // 把编码的字符串输入文件，返回值为图片文件名
87        CString FileName = base64_decode_picture(attachmentName,
    attachmentEncode);
88
89        //HBITMAP hBmp = (HBITMAP)::LoadImage(
90        //  AfxGetInstanceHandle(), FileName, IMAGE_BITMAP, 0, 0,
    LR_LOADFROMFILE);  // 将位图加载到 hBmp
91        //pdlg->m_picture.SetBitmap(hBmp);
92
93        // CImage provides enhanced bitmap support, including the ability to
     load and save
94        // images in JPEG, GIF, BMP, and Portable Network Graphics (PNG)
    formats.
95        CImage image;
96        HBITMAP hBmp;
97        image.Load(FileName);
98        hBmp = image.Detach();
99        // 让图片控件显示位图
100       // pdlg->m_picture.SetBitmap(NULL);
101       pdlg->m_picture.SetBitmap(hBmp);
102
103       pdlg->UpdateData();
104       pdlg->Invalidate();
105       pdlg->UpdateWindow();
106
```

```
107         // 将图片加入图片列表里，交给 Picture 按钮用默认程序打开图片
108         // 如果需要打开多张图，必须在收信前按下 Picture 按钮
109         imageList.push(FileName);
110       }
111
112       attachment.erase(attachmentName);
113     }
114
115     // 用默认程序打开图片
116     if (pdlg->openPictures)
117     {
118       while (!imageList.empty())
119       {
120         CString imageName = imageList.front();
121         ShellExecute(nullptr, _T("open"), imageName, _T(""), _T(""), SW_SHOW
    );
122         imageList.pop();
123       }
124     }
125
126
127     // 传输数据终止
128     data = false;
129     // 投递一个“读“的事件，准备接收
130     AsyncSelect(FD_READ);
131   }
132   ...
133 }
```

Listing 6: 处理接收到的数据

5.6.1 功能：邮件正文及文本型附件解码

代码摘要：对邮件正文及文本型附件解码。

1. ServerSocket::ServerSocket()
   - 在 SMTP Socket 初始化函数中构造 base64 字符编码映射
   - 更新日志
   - 监听 Socket 绑定 TCP 端口 25
2. void ServerSocket::CheckSymbol(char& c)
   - 将 base64 编码后的字符解码
   - 若 base64 编码后的符号为'='，则自身更新为 0
   - 若为其他字符，则自身更新为解码后的值

3. string ServerSocket::base64_decode(string encoded_string)

   - 移除传入编码的所有回车换行符
   - 按照编码算法反向思考，解码

```
ServerSocket::ServerSocket()
{
  // 构建 base64 映射表
  for (int i = 0; i <= 'Z' - 'A'; ++i)
  {
    int temp = 'A' + i;
    char temp_char(temp);
    base64_map.insert(pair<char, int>(temp_char, i));
  }
  for (int i = 26; i <= 'z' - 'a' + 26; ++i)
  {
    int temp = 'a' + i - 26;
    char temp_char(temp);
    base64_map.insert(pair<char, int>(temp_char, i));
  }
  for (int i = 52; i <= '9' - '0' + 52; ++i)
  {
    int temp = '0' + i - 52;
    char temp_char(temp);
    base64_map.insert(pair<char, int>(temp_char, i));
  }
  base64_map.insert(pair<char, int>('+', 62));
  base64_map.insert(pair<char, int>('/', 63));
}

// 将 base64 编码后的字符解码
// 参数解释：
// char& c -> 待解码的字符
// 返回值：无
void ServerSocket::CheckSymbol(char& c)
{
  // 若 base64 编码后的符号为 '='，则自身更新为 0
  // '=' 设为 0 不会影响输出，因为在 ASCII 里，0 -> NULL
  if (c == '=')
    c = 0;
  else
  {
    // 若为其他字符，则自身更新为解码后的值
    int temp = base64_map.find(c)->second;
    c = char(temp);
  }
}

```

```
// base64 解码
// 参数解释:
// string encoded_string -> base64 编码
// 返回值: string 解码出的字符串
string ServerSocket::base64_decode(string encoded_string)
{
  // 声明一个用于存储解码出的字符的字符串
  string decoded_string = "";

  // 移除回车换行符
  size_t pos;
  while (pos = encoded_string.find('\r'))
  {
    // 满足下面这个条件就是没找到
    if (pos == encoded_string.npos)
      break;
    encoded_string.erase(pos,1);
  }
  while (pos = encoded_string.find('\n'))
  {
    // 满足下面这个条件就是没找到
    if (pos == encoded_string.npos)
      break;
    encoded_string.erase(pos,1);
  }

  // 开始解码
  pos = 0;
  while (pos < encoded_string.length())
  {
    // 读入 4 个 base64 编码后的字符
    char u, x, y, z;
    u = encoded_string[pos++];
    x = encoded_string[pos++];
    y = encoded_string[pos++];
    z = encoded_string[pos++];

    // 将 base64 编码后的字符解码为 6 位二进制数
    CheckSymbol(u); CheckSymbol(x); CheckSymbol(y); CheckSymbol(z);


    // 声明解码后得到的 3 个字符 i, j, k, 临时变量 temp1, temp2
    char i, j, k, temp1, temp2;
    i = j = k = 0b00000000;

    // 按照编码算法反向思考, 解码
    temp1 = u << 2; temp2 = x >> 4;
    i |= temp1; i |= temp2;


    temp1 = x << 4; temp2 = y >> 2;
```

```
    j |= temp1; j |= temp2;


    temp1 = y << 6; temp2 = z;
    k |= temp1; k |= temp2;

    // 将解码后的字符放入之前声明的字符串
    decoded_string = decoded_string + i + j + k;
  }

  // 在字符串中添加一个终止符号
  decoded_string += '\0';


  return decoded_string;
}
```

Listing 7: 邮件正文及文本型附件解码

5.6.2 功能：图片型附件解码

代码摘要：对图片型附件解码。

1. ServerSocket::ServerSocket()
   - 在 SMTP Socket 初始化函数中构造 base64 字符编码映射
   - 更新日志
   - 监听 Socket 绑定 TCP 端口 25
2. void ServerSocket::CheckSymbol(char& c)
   - 将 base64 编码后的字符解码
   - 若 base64 编码后的符号为'='，则自身更新为 0
   - 若为其他字符，则自身更新为解码后的值
3. CString ServerSocket::base64_decode_picture(string attachmentName, string encoded_string)
   - 移除传入编码的所有回车换行符
   - 按照编码算法反向思考，解码
   - 将解码后的字符流写入文件

```cpp
ServerSocket::ServerSocket()
{
  // 构建 base64 映射表
  for (int i = 0; i <= 'Z' - 'A'; ++i)
  {
    int temp = 'A' + i;
    char temp_char(temp);
    base64_map.insert(pair<char, int>(temp_char, i));
  }
  for (int i = 26; i <= 'z' - 'a' + 26; ++i)
  {
    int temp = 'a' + i - 26;
    char temp_char(temp);
    base64_map.insert(pair<char, int>(temp_char, i));
  }
  for (int i = 52; i <= '9' - '0' + 52; ++i)
  {
    int temp = '0' + i - 52;
    char temp_char(temp);
    base64_map.insert(pair<char, int>(temp_char, i));
  }
  base64_map.insert(pair<char, int>('+', 62));
  base64_map.insert(pair<char, int>('/', 63));
}

// 将 base64 编码后的字符解码
// 参数解释:
// char& c -> 待解码的字符
// 返回值: 无
void ServerSocket::CheckSymbol(char& c)
{
  // 若 base64 编码后的符号为 '=', 则自身更新为 0
  // '=' 设为 0 不会影响输出, 因为在 ASCII 里, 0 -> NULL
  if (c == '=')
    c = 0;
  else
  {
    // 若为其他字符, 则自身更新为解码后的值
    int temp = base64_map.find(c)->second;
    c = char(temp);
  }
}

// base64 图片解码函数
// 参数解释:
// string attachmentName -> 文件名,
// string encoded_string -> 图片 base64 编码
// 返回值: CString 文件名
CString ServerSocket::base64_decode_picture(string attachmentName,string
    encoded_string)
{
```

```
// 声明一个用于存储解码出的字符的队列容器
queue<char>decoded_string;

// 移除回车换行符
size_t pos;
while (pos = encoded_string.find('\r'))
{
  // 满足下面这个条件就是没找到
  if (pos == encoded_string.npos)
    break;
  encoded_string.erase(pos, 1);
}
while (pos = encoded_string.find('\n'))
{
  // 满足下面这个条件就是没找到
  if (pos == encoded_string.npos)
    break;
  encoded_string.erase(pos, 1);
}


pos = 0;
while (pos < encoded_string.length())
{
  // 读入 4 个 base64 编码后的字符
  char u, x, y, z;
  u = encoded_string[pos++];
  x = encoded_string[pos++];
  y = encoded_string[pos++];
  z = encoded_string[pos++];
  CheckSymbol(u); CheckSymbol(x); CheckSymbol(y); CheckSymbol(z);


  // 声明解码后得到的 3 个字符 i, j, k, 临时变量 temp1, temp2
  char i, j, k, temp1, temp2;
  i = j = k = 0b00000000;

  // 按照编码算法反向思考, 解码
  temp1 = u << 2; temp2 = x >> 4;
  i |= temp1; i |= temp2;


  temp1 = x << 4; temp2 = y >> 2;
  j |= temp1; j |= temp2;


  temp1 = y << 6; temp2 = z;
  k |= temp1; k |= temp2;

  // 将解码后的字符放入之前声明的队列
  decoded_string.push(i);
```

```
    decoded_string.push(j);
    decoded_string.push(k);
  }

  // 将解码后的字符流写入文件
  CString FileName(attachmentName.c_str());

  CFile file;
  // 以 Create 方式和 Write 方式打开文件
  // 若文件已存在就会清空后再写入
  BOOL bOpen = file.Open(FileName, CFile::modeCreate | CFile::modeWrite);
  if (bOpen)
  {
    file.SeekToEnd(); // 将指针移至文件末尾进行追加
    while (!decoded_string.empty())
    {
      // 一次循环写入一个字符
      file.Write((CString)decoded_string.front(), 1);
      decoded_string.pop();
    }
  }
  file.Close();

  return FileName;
}
```

Listing 8: 图片型附件解码

5.7 非关键功能：清空对话框 + 使用默认软件打开图片

代码摘要：这一类代码都很简短，也不复杂，就不介绍了。感兴趣的话可以用 Visual Studio 打开项目后，打开资源视图，双击对话框中的 Pictures open button 和 clear 查看。使用默认软件打开图片的功能具体调用在 5.6 小节。

6 用户手册

6.1 程序界面以及使用方法简介

图 1: 程序用户界面

程序用户界面如图 1。各个区域功能简介：

1. 邮件正文区（Mail Text）：显示邮件正文。
2. 附件列表区（Attachment List）：显示客户端发过来的邮件里有哪些附件。
3. 图片显示区（Picture Attachment）：根据图片附件在邮件报文中的顺序，显示邮件报文附件中的第一张图片。
4. 打开图片按钮（Pictures Open Button）：（推荐，在接收邮件前按下按钮）接收到的图片附件会用系统默认软件打开，更好地显示分辨率较高的图片。
5. 服务器日志区（Server Log）：显示服务器日志。
6. 邮件报文区（Mail Message）：显示接收到的原生的邮件报文。

7. 文本型附件显示区（Text Attachment）：如果接收到的邮件中有文本型附件（仅支持 txt 类型的文本型附件），其内容会在此处显示，且显示所有文本型附件。

8. 清空所有显示按钮（Clear Button）：清空所有显示。

9. 开启服务器按钮（Start Button）：开启服务器。

正确收发邮件的过程如下：首先打开服务器，然后打开邮箱客户端，正确添加账号后，将 SMTP 服务器地址绑定为 localhost，端口号设为 25。如图 2 在客户端中编辑邮件，之后发送邮件即可。收到邮件后服务器正确显示如图 3 。（图中的所有附件均可在本作业压缩包里找到）

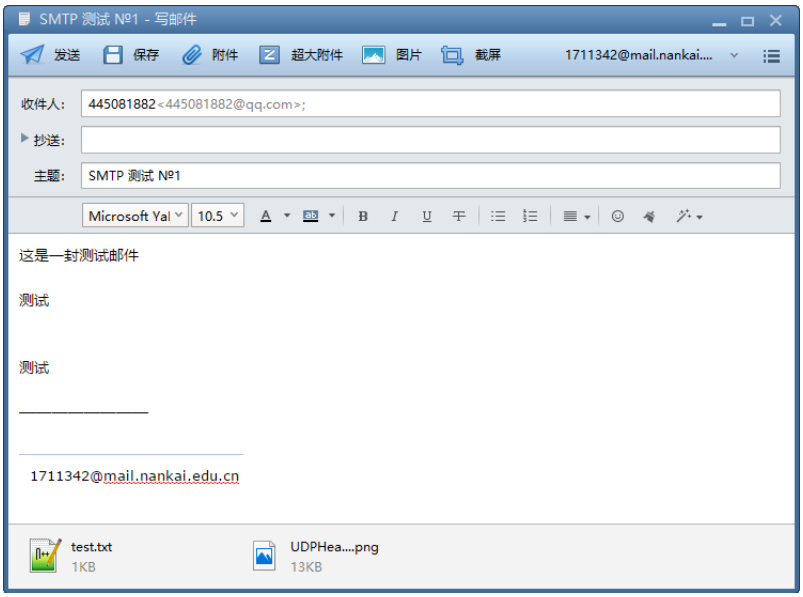

图 2: 客户端中的邮件

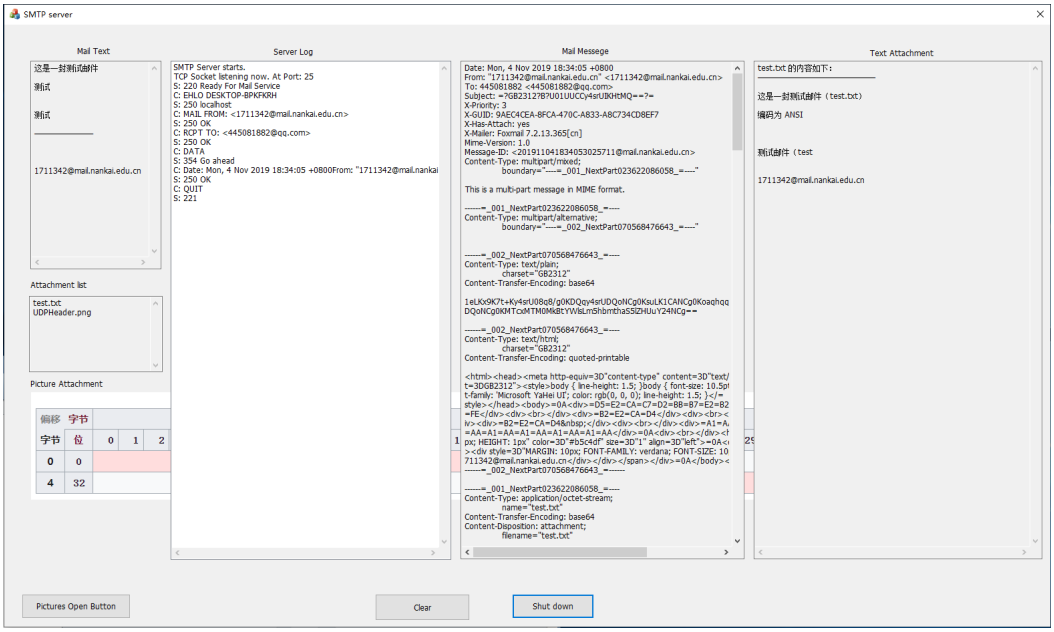
SMTP server
Mail Text
这是一封测试邮件
测试
测试
1711342@mail.nankai.edu.cn
Attachment list
test.txt
UDPHeader.png
Picture Attachment
偏移 字节
字节 位 0 1 2
0 0
4 32
Server Log
SMTP Server starts.
TCP Socket listening now. At Port: 25
S: 220 Ready For Mail Service
C: EHLO DESKTOP-BPKFKRH
S: 250 localhost
C: MAIL FROM: <1711342@mail.nankai.edu.cn>
S: 250 OK
C: RCPT TO: <445081882@qq.com>
S: 250 OK
C: DATA
S: 354 Go ahead
C: Date: Mon, 4 Nov 2019 18:34:05 +0800From: "1711342@mail.nankai
S: 250 OK
C: QUIT
S: 221
Mail Messege
Date: Mon, 4 Nov 2019 18:34:05 +0800
From: "1711342@mail.nankai.edu.cn" <1711342@mail.nankai.edu.cn>
To: 445081882 <445081882@qq.com>
Subject: =?GB2312?B?U01UUCCy4srUBKHtMQ==?=
X-Priority: 3
X-GUID: 9AEC4CEA-8FCA-470C-A833-A8C734CD8EF7
X-Has-Attach: yes
X-Mailer: Foxmail 7.2.13.365[cn]
Mime-Version: 1.0
Message-ID: <20191104183405302571l@mail.nankai.edu.cn>
Content-Type: multipart/mixed;
boundary="----=_001_NextPart023622086058_=----"
This is a multi-part message in MIME format.
------=_001_NextPart023622086058_=----
Content-Type: multipart/alternative;
boundary="----=_002_NextPart070568476643_=----"
------=_002_NextPart070568476643_=----
Content-Type: text/plain;
charset="GB2312"
Content-Transfer-Encoding: base64
1eLKx9K7t+Ky4srU08q8/g0KDQqy4srUDQoNCg0KsuLK1CANCg0Koaqhqq
DQoNCg0KMTCxMTM0MkBtYWlsLmShbmhiaSSIZHUuY2NNCg==
------=_002_NextPart070568476643_=----
Content-Type: text/html;
charset="GB2312"
Content-Transfer-Encoding: quoted-printable
<html><head><meta http-equiv=3D"content-type" content=3D"text/
t=3DGB2312"><style>body { line-height: 1.5; }body { font-size: 10.5pt
t-family: 'Microsoft YaHei UI'; color: rgb(0, 0, 0); line-height: 1.5; }</=
style></head><body>=0A =0S=E2=CA=C7=D2=BB=B7=E2=B2
=FE
 =B2=E2=CA=D4
iv> =B2=E2=CA=D4
 =A1=A
=AA=A1=AA=A1=AA=A1=AA=A1=AA =0A
 <
px; HEIGHT: 1px" color=3D"#b5c4df" size=3D"1" align=3D"left">=0A<
><div style=3D"MARGIN: 10px; FONT-FAMILY: verdana; FONT-SIZE: 10
711342@mail.nankai.edu.cn =0A</body><
------=_002_NextPart070568476643_=------
------=_001_NextPart023622086058_=----
Content-Type: application/octet-stream;
name="test.txt"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename="test.txt"
Text Attachment
test.txt 的内容如下：
这是一封测试邮件（test.txt）
编码为 ANSI
测试邮件（test
1711342@mail.nankai.edu.cn
Pictures Open Button
Clear
Shut down

图 3: 服务器正确显示

6.2 注意事项

1. 目前仅能保证程序在开发及测试环境中正常工作，其他情况均不能保证程序能够稳定运行。开发及测试环境请参考 4.1。
2. 文本型附件**仅测试过采用 ANSI 编码的 txt 类型文本文件**，其他类型文件均未测试。
3. 图片显示区**仅能显示一张图片**，若需要显示所有图片请按下打开图片按钮。
4. 理论上服务器仅支持总大小 9MB 以下大小的邮件。为了保证服务器运行稳定，**一封邮件最好不要超过 1MB**。
5. 若有同一文件名的附件，只接收邮件报文中显示的第一个，其余的丢弃。所以本服务器**不支持**接收同名文件。

7 验证程序的正确性

以上一节 6.1 举的例子继续检验程序正确性，除能看到服务器界面的显示之外，我们还可以使用 Wireshark 来抓包，进一步检验程序正确性。

从图 4 中可以看到服务器与客户端的通信全过程。

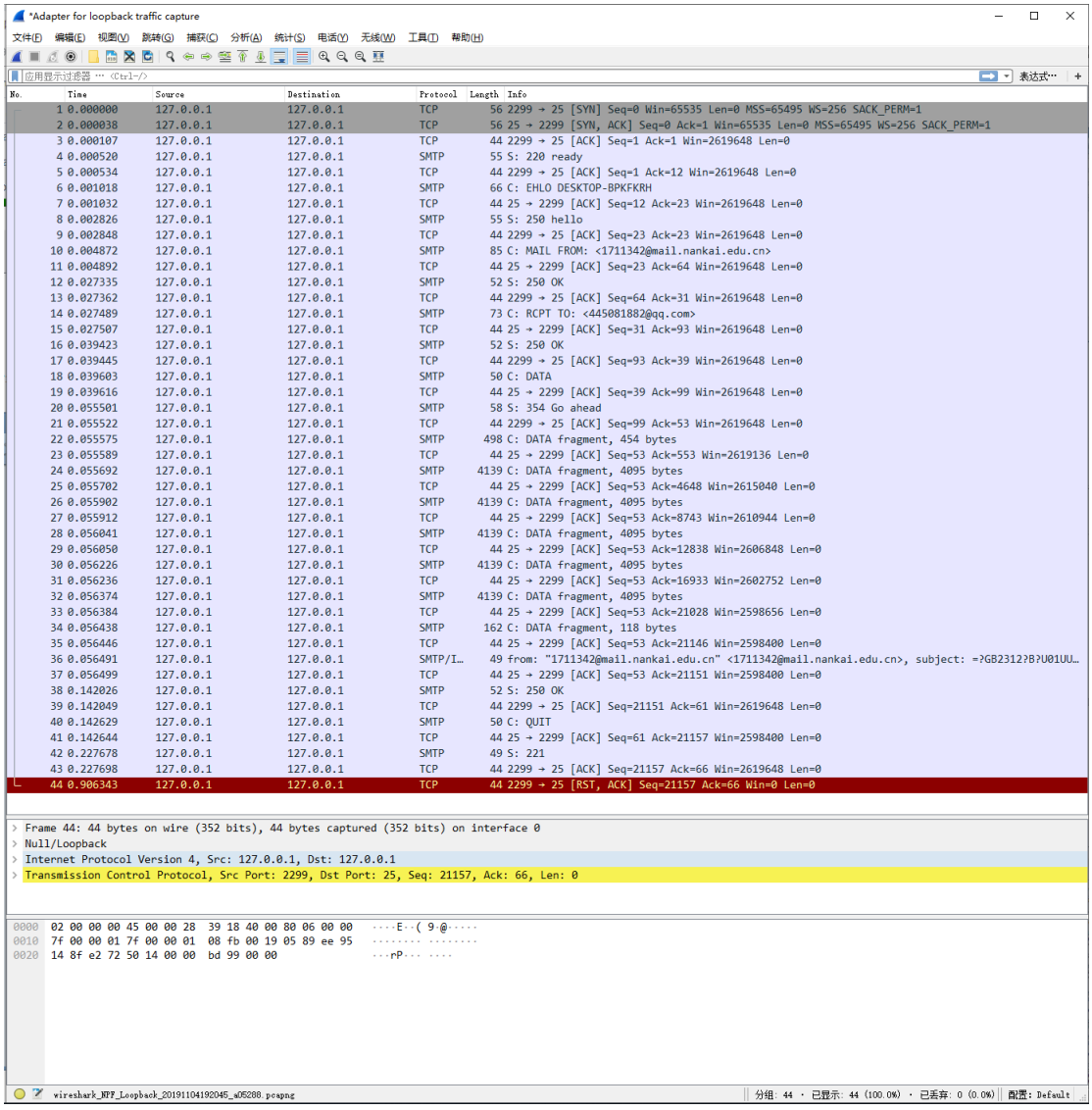

图 4: 服务器与客户端的通信全过程

A SMTP

The **Simple Mail Transfer Protocol** (**SMTP**) is a communication protocol for electronic mail transmission. As an Internet standard, SMTP was first defined in 1982 by RFC 821, and updated in 2008 by RFC 5321 to Extended SMTP additions, which is the protocol variety in widespread use today. Mail servers and other message transfer agents use SMTP to send and receive mail messages. *SMTP servers commonly use the Transmission Control Protocol on port number 25.*[3]

A.1 SMTP transport example

A typical example of sending a message via SMTP to two mailboxes (alice and theboss) located in the same mail domain (example.com or localhost.com) is reproduced in the following session exchange. (In this example, the conversation parts are prefixed with S: and C:, for server and client, respectively; these labels are not part of the exchange.)

After the message sender (SMTP client) establishes a reliable communications channel to the message receiver (SMTP server), the session is opened with a greeting by the server, usually containing its fully qualified domain name (FQDN), in this case smtp.example.com. The client initiates its dialog by responding with a **HELO** command identifying itself in the command's parameter with its FQDN (or an address literal if none is available).

Example:

S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.com
S: 250 smtp.example.com, I am glad to meet you
C: MAIL FROM:<bob@example.com>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>

C: From: ”Bob Example” <bob@example.com>

C: To: Alice Example <alice@example.com>

C: Cc: theboss@example.com

C: Date: Tue, 15 Jan 2008 16:02:43 -0500

C: Subject: Test message

C:

C: Hello Alice.

C: This is a test message with 5 header fields and 4 lines in the message body.

C: Your friend,

C: Bob

C: .

S: 250 Ok: queued as 12345

C: QUIT

S: 221 Bye

The server closes the connection

The client notifies the receiver of the originating email address of the message in a **MAIL FROM** command. This is also the return or bounce address in case the message cannot be delivered. In this example the email message is sent to two mailboxes on the same SMTP server: one for each recipient listed in the To and Cc header fields. The corresponding SMTP command is **RCPT TO**. Each successful reception and execution of a command is acknowledged by the server with a result code and response message (e.g., 250 Ok).

The transmission of the body of the mail message is initiated with a **DATA** command after which it is transmitted verbatim line by line and is terminated with an end-of-data sequence. This sequence consists of a new-line (<CR><LF>), a single full stop (period), followed by another new-line. Since a message body can contain a line with just a period as part of the text, the client sends two periods every time a line starts with a period; correspondingly, the server replaces every sequence of two periods at the beginning of a line with a single one. Such escaping method is called *dot-stuffing.*

The server's positive reply to the end-of-data, as exemplified, implies that the server has taken the responsibility of delivering the message. A message can be doubled if there is a communication failure at this time, e.g.

due to a power shortage: Until the sender has received that 250 reply, it must assume the message was not delivered. On the other hand, after the receiver has decided to accept the message, it must assume the message has been delivered to it. Thus, during this time span, both agents have active copies of the message that they will try to deliver.[20] The probability that a communication failure occurs exactly at this step is directly proportional to the amount of filtering that the server performs on the message body, most often for anti-spam purposes. The limiting timeout is specified to be 10 minutes.

The **QUIT** command ends the session. If the email has other recipients located elsewhere, the client would **QUIT** and connect to an appropriate SMTP server for subsequent recipients after the current destination(s) had been queued. The information that the client sends in the **HELO** and **MAIL FROM** commands are added (not seen in example code) as additional header fields to the message by the receiving server. It adds a **Received** and **Return-Path** header field, respectively.

Some clients are implemented to close the connection after the message is accepted (**250 Ok: queued as 12345**), so the last two lines may actually be omitted. This causes an error on the server when trying to send the **221** reply.

A.2 Extended Simple Mail Transfer Protocol

The original SMTP protocol supported only unauthenticated unencrypted ASCII text communications susceptible to Man in the Middle attacks, spoofing, and spamming, and requiring to encode any binary data before transmission. A number of optional extensions specify various mechanisms to address these problems.

这次实验我遇到的在 Extended SMTP 里的命令只有 EHLO。

A.2.1 Optional extensions discovery

Clients learn a server's supported options by using the EHLO greeting, as exemplified below, instead of the original HELO (example above). Clients fall back to HELO only if the server does not support SMTP extensions. For example:

Example:

S: 220 smtp2.example.com ESMTP Postfix

C: **EHLO** bob.example.com

S: 250-smtp2.example.com Hello bob.example.org [192.0.2.201]

S: 250-SIZE 14680064

S: 250-PIPELINING

S: 250 HELP

B Base64

In computer science, **Base64** is a group of binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. The term Base64 originates from a specific MIME content transfer encoding. Each Base64 digit represents exactly 6 bits of data. Three 8-bit bytes (i.e., a total of 24 bits) can therefore be represented by four 6-bit Base64 digits.[1]

B.1 Base64 table

The Base64 index table:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
padding		=									

图 5: Base64 table

B.2 Base64 Implementations

Implementations may have some constraints on the alphabet used for representing some bit patterns. This notably concerns the last two characters used in the index table for index 62 and 63, and the character used for padding (which may be mandatory in some protocols, or removed in others). The table below summarizes these known variants, and link to the subsections below.

Variants summary table [编辑源代码] 实验涉及到的 MIME 的 base64 实现

Implementations may have some constraints on the alphabet used for representing some bit patterns. This notably concerns the last two characters used in the index table for index 62 and 63, and the character used for padding (which may be mandatory in some protocols, or removed in others). The table below summarizes these known variants, and link to the subsections below.

Encoding	Encoding characters			Separate encoding of lines			Decoding non-encoding characters
	62nd	63rd	*pad*	Separators	Length	Checksum	
Base64 for Privacy-Enhanced Mail (PEM; RFC 1421; deprecated)	+	/	= mandatory	CR+LF	64, or lower for the last line	No	No
Base64 transfer encoding for MIME (RFC 2045)	+	/	= mandatory	CR+LF	At most 76	No	Discarded
base64 for RFC 4648 (previously, RFC 3548; standard)[a]	+	/	= mandatory		N/A		No
base64url URL- and filename-safe (RFC 4648 §5)[a]	-	_	= optional		N/A		No
Radix-64 for OpenPGP (RFC 4880)	+	/	= mandatory	CR+LF	At most 76	Radix-64 encoded 24-bit CRC	No
Base64 for UTF-7 (RFC 2152)	+	/			N/A		No
Base64 encoding for IMAP mailbox names (RFC 3501)	+	,			N/A		No
Y64 URL-safe Base64 from YUI Library[7]	.	_	- optional		N/A		No
Program identifier Base64 variant 1 (non-standard)	_	-		Unknown			
Program identifier Base64 variant 2 (non-standard)	.	_		Unknown			
Freenet URL-safe Base64 (non-standard)	~	-	= optional		N/A		No

图 6: Base64 Implementations

C MIME

Multipurpose Internet Mail Extensions (**MIME**) is an Internet standard that *extends the format of email messages to support text in character sets other than ASCII, as well attachments of audio, video, images, and application programs.* Message bodies may consist of multiple parts, and

header information may be specified in non-ASCII character sets. Email messages with MIME formatting are typically transmitted with standard protocols, such as the Simple Mail Transfer Protocol (SMTP), the Post Office Protocol (POP), and the Internet Message Access Protocol (IMAP).[2]

The MIME standard is specified in a series of requests for comment: RFC 2045, RFC 2046, RFC 2047, RFC 2048, RFC 2049, RFC 4288, RFC 4289. The integration with SMTP email is specified in RFC 1521 and RFC 1522.

Although the MIME formalism was designed mainly for SMTP, its content types are also important in other communication protocols. In the HyperText Transfer Protocol (HTTP) for the World Wide Web, servers insert a MIME header field at the beginning of any Web transmission. Clients use the content type or media type header to select an appropriate viewer application for the type of data indicated. Browsers typically contain GIF and JPEG image viewers.

C.1 MIME header fields

Note: 主要介绍实验中遇到的知识点，不会全部涉及。若要查看所有相关 MIME 知识，请参看相关 RFC。

MIME-Version

The presence of this header field indicates the message is MIME-formatted. The value is typically "1.0". The field appears as follows:

$$Mime - Version : 1.0$$

Content-Type

This header field indicates the media type of the message content, consisting of a type and subtype, for example

$$Content - Type : text/plain$$

Content-Transfer-Encoding

In June 1992, MIME (RFC 1341, since made obsolete by RFC 2045) defined a set of methods for representing binary data in formats other than ASCII text format.

$$Content - Transfer - Encoding : base64$$

Base64 encoding is suitable for use with normal SMTP, and used to encode arbitrary octet sequences into a form that satisfies the rules of 7bit. Designed to be efficient for non-text 8 bit and binary data. Sometimes used for text data that frequently uses non-US-ASCII characters.

C.2 Multipart messages

The MIME multipart message contains a boundary in the header field *Content-Type:*; this boundary, which must not occur in any of the parts, is placed between the parts, and at the beginning and end of the body of the message, as follows:

```
Content-Type: multipart/mixed;
    boundary="—-=_001_NextPart650301145202_=—-"

This is a multi-part message in MIME format.

——=_001_NextPart650301145202_=—-
Content-Type: multipart/alternative;
    boundary="—-=_002_NextPart333026758736_=—-"
```

Multipart subtypes

The MIME standard defines various multipart-message subtypes, which specify the nature of the message parts and their relationship to one another. The subtype is specified in the "Content-Type" header field of the overall message. For example, a multipart MIME message using the digest subtype would have its Content-Type set as "multipart/digest".

Mixed: Multipart/mixed is used for sending files with different "Content-Type" header fields inline (or as attachments). If sending pictures or other easily readable files, most mail clients will display them inline (unless otherwise specified with Content-disposition). Otherwise, it offers them as attachments. The default content-type for each part is "text/plain".

Alternative: The multipart/alternative subtype indicates that each part is an "alternative" version of the same (or similar) content, each in a different format denoted by its "Content-Type" header. The order of the parts is significant. RFC1341 states: *In general, user agents that compose*

multipart/alternative entities should place the body parts in increasing order of preference, that is, with the preferred format last.

References

[1] Wikipidia. *Base64*. URL: https://en.wikipedia.org/wiki/Base64. (accessed: 10.18.2019).

[2] Wikipidia. *MIME*. URL: https://en.wikipedia.org/wiki/MIME. (accessed: 10.18.2019).

[3] Wikipidia. *SMTP*. URL: https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol. (accessed: 9.23.2019).

[4] 张建忠、徐敬东. 计算机网络技术与应用. 北京清华大学学研大厦 A 座: 清华大学出版社, 2019.