一、 实验详细设计

（注意不要完全照搬实验指导书上的内容，请根据你自己的设计方案来填写图文并茂地描述实验实现的所有功能和详细的设计方案及实验过程中的特色部分。）

1. 邮件发送客户端详细设计

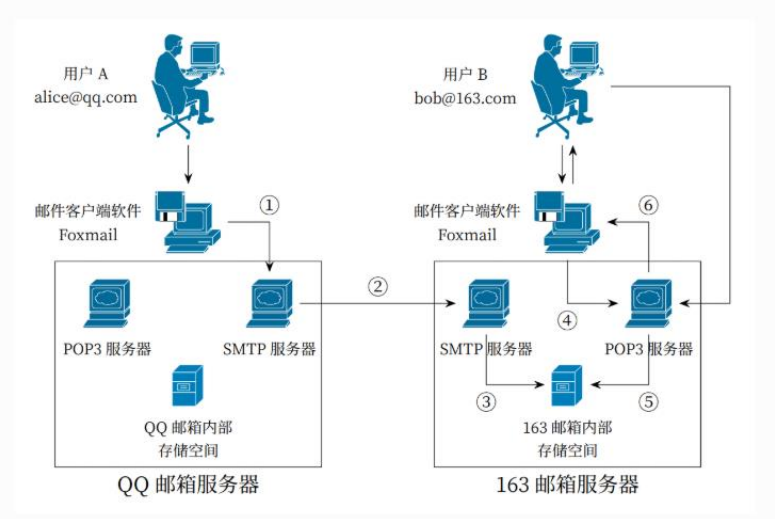
【实验任务与实验要求】

本次实验的第一个任务（send.c），是在不使用其它第三方库的前提下，仅仅基于 Socket 编程、SMTP 协议，实现一个邮件发送客户端。该客户端需要能够连接到邮件服务器，并将包含有附件的邮件发送至任意合法的收件地址。

此外，程序要能够使用网络上知名的邮件服务提供商（如 QQ、网易等）进行测试，并且需要打印出交互过程中服务器的回复信息。

【实验总体设计思路】

电子邮件的发送和接收过程如下图所示：



其中涉及了邮件发送的过程包括有①、②、③。也就是说，在发送邮件时，客户端需要先与 SMTP 服务器进行通信，将想要发送的邮件交付给 SMTP 服务器；然后由 SMTP 判断收件地址是否位于自己管辖的范围内，如果是则存放到自己的存储空间，如果不是则会转发给其它供应商的 SMTP 服务器，再由其它供应商的 SMTP 服务器存放它们对应的存储空间。到此为止，我们可以认为成功完成了一次邮件发送。

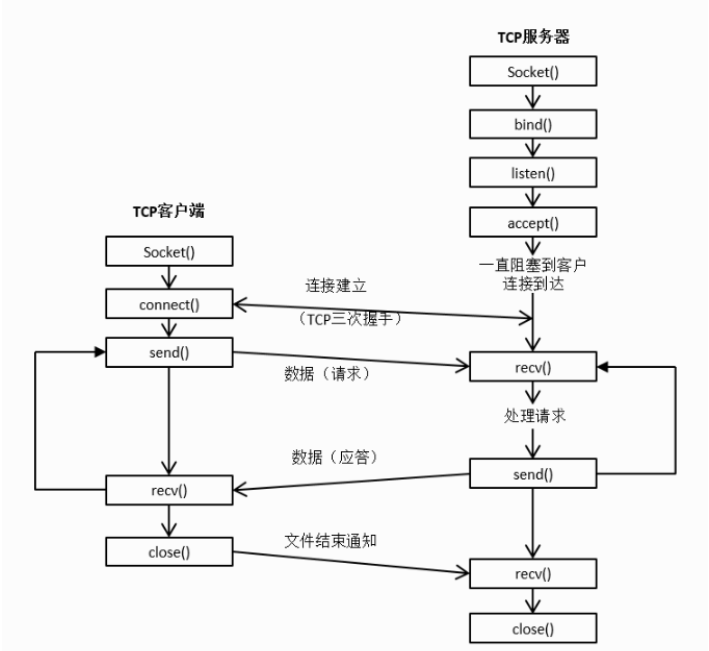
可见，完成一次邮件客户端的关键点在于实现好邮件客户端与 SMTP 服务器之间，以

及不同 SMTP 服务器之间的通信。然而，如何确定上述交互过程的具体规则呢？这就由此引出了本小节实验要使用到的 SMTP 协议（**SMTP 协议就定义了上述过程的交互规则**）。SMTP 协议共定义了 18 条命令，完成电子邮件的发送经常会用的 6 条命令包括有：“EHLO”、“AUTH”、“MAIL FROM”、“RCPT TO”、“DATA”、“QUIT”等。

因此，使用 SMTP 协议完成一次发送邮件的主要过程如下：

- 服务器发送欢迎消息。
- 客户端发送 EHLO 命令表明身份（EHLO），然后接收到服务器所列出的它支持的命令。
- 客户端选择登录认证方式，本次实验选择的是 login 方式（AUTH login）。
- 客户端依次发送经过 Base64 编码的用户名、口令给服务器。如果用户名、口令正确，服务器会提示认证成功，否则认证失败。
- 客户端指定邮件的发送人（MAIL FROM）和接收人（RCPT TO）
- 客户端输入 DATA 命令，然后编写要发送的内容（需要遵守 Internet 消息格式）。
- 客户端输入“.”表示邮件内容输入完毕，服务器提示成功。
- 客户端输入 QUIT 命令断开与服务器的连接。

上述过程可以通过在 Linux 操作系统下使用 **Telnet 命令**交互完成，但本次实验要求通过 C 语言编程实现，而**不是使用终端命令行的方式**。那又该怎么办呢？这也就由此引出了本次实验要用到的 **Socket 编程方法**。Socket（套接字）是一种**独立于协议**的网络编程接口，提供了许多函数或例程，可以供我们开发应用程序。由于 SMTP 协议、POP3 协议运行在 TCP 协议上，因此本次实验所用到的是基于 **TCP 协议**的**流式 Socket**：



从上图不难看出，在客户端与服务器分别完成初始化，并成功连接到一起之后，会通过 send()、recv()函数实现通信。我们不难想到，**原本在 Linux 终端下用 Telnet 所发送的各个 SMTP 协议命令，同样也可以使用 Socket 的 send()函数来进行发送，并使用 recv()函数来接收来自服务器的应答**。另外，本次实验我们做的是客户端，所以也仅需关心客户端的处理。

综上所述，**从整体来看，我们所要做的就是**：初始化 Socket 套接字接口，封装好 send()、recv()函数，然后使用封装后的新函数来与服务器进行 SMTP 协议的交互。这样就可以完成邮件发送客户端的实现了。接下来，本文会详细介绍各部分的设计思路与代码实现。

【各部分的设计思路与代码实现】

（0）准备工作：初始化 Socket 接口，封装好发送函数、接收函数

首先，**调用 socket()函数创建并初始化**一个新的 Socket 套接字接口。然后，创建一个用于表示**服务器地址、端口号**的 sockaddr_in 结构体变量，命名为 servaddr，并对其中的信息进行填充。值得注意的是，与协议栈实验类似，需要对端口号进行**大小端序的转换**。填充好后，调用 connect 函数建立一个到服务器的连接。代码如下所示：

```
    // TODO: Create a socket, return the file descriptor to s_fd, and
establish a TCP connection to the mail server
    s_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (s_fd == -1)
    {
        perror("socket");
        exit(EXIT_FAILURE);
    }
    struct sockaddr_in servaddr;
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = swap16(port);
    servaddr.sin_addr = (struct in_addr) {inet_addr(dest_ip)};
    bzero(servaddr.sin_zero, 8);
    connect(s_fd, (struct sockaddr *) (&servaddr), sizeof(struct
sockaddr_in));
```

原本的模板代码中提供了使用 recv()函数的程序片段。但是，由于我们本次实验在完成 send()、recv()之后，**还需要将其打印出相应的中间过程、调试信息**，并且这样的发送、接收、打印信息的操作步骤在后面实现 SMTP 协议各个过程的时候都会用到，所以我们可以对 send()、recv()这两个函数进行进一步的封装。代码如下所示：

```
void recv_from_server(int s_fd, void* buf, int flags, char* err_msg)
{
```

```
    int r_size = -1;
    if ((r_size = recv(s_fd, buf, MAX_SIZE, 0)) == -1)
    {
        perror(err_msg);
        exit(EXIT_FAILURE);
    }
    char* buf_temp = (char *) buf;
    buf_temp[r_size] = '\0'; // Do not forget the null terminator
    printf("%s", buf_temp);
}

void send_to_server(int s_fd, void* buf, int len, int flags, char* err_msg)
{
    int s_size = -1;
    if ((s_size = send(s_fd, buf, len, 0)) == -1)
    {
        perror(err_msg);
        exit(EXIT_FAILURE);
    }
    char* buf_temp = (char *) buf;
    printf("%s", buf_temp);
}
```

（1）服务器发送欢迎消息，然后客户端再发送 EHLO，然后再接收来自服务器的应答。代码如下所示：

```
    // Print welcome message
    recv_from_server(s_fd, (void *) buf, 0, "recv welcome");

    // Send EHLO command and print server response
    const char* EHLO = "EHLO 163.com\r\n"; // TODO: Enter EHLO command here
    send_to_server(s_fd, (void *) EHLO, strlen(EHLO), 0, "send EHLO");
    // TODO: Print server response to EHLO command
    recv_from_server(s_fd, (void *) buf, 0, "recv EHLO");
```

（2）客户端进行身份验证。在 SMTP 协议中，用户名不能直接发送，**还要经过 Base64 编码的转换**。代码如下所示：

```
    // TODO: Authentication. Server response should be printed out.
    const char* AUTH = "AUTH login\r\n";
    send_to_server(s_fd, (void *) AUTH, strlen(AUTH), 0, "send AUTH");
    recv_from_server(s_fd, (void *) buf, 0, "recv AUTH");
```

```
    char* user64 = encode_str(user);
    send_to_server(s_fd, (void *) user64, strlen(user64), 0, "send
user64");
    recv_from_server(s_fd, (void *) buf, 0, "recv user64");

    char* pass64 = encode_str(pass);
    send_to_server(s_fd, (void *) pass64, strlen(pass64), 0, "send
pass64");
    recv_from_server(s_fd, (void *) buf, 0, "recv pass64");
```

（3）发送“MAIL FROM”、“RCPT TO”命令指定发送人、收件人，然后发送“DATA”命令，告诉服务器接下来要发送邮件内容了。

```
    // TODO: Send MAIL FROM command and print server response
    sprintf(buf, "MAIL FROM:<%s>\r\n", from);
    send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send MAIL FROM");
    recv_from_server(s_fd, (void *) buf, 0, "recv MAIL FROM");

    // TODO: Send RCPT TO command and print server response
    sprintf(buf, "RCPT TO:<%s>\r\n", receiver);
    send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send RCPT TO");
    recv_from_server(s_fd, (void *) buf, 0, "recv RCPT TO");

    // TODO: Send DATA command and print server response
    const char* DATA = "DATA\r\n";
    send_to_server(s_fd, (void *) DATA, strlen(DATA), 0, "send DATA");
    recv_from_server(s_fd, (void *) buf, 0, "recv DATA");
```

（4）完成邮件正文文本的发送。

由于需要使用字符串“qwertyuiopasdfghjklzxcvbnm”来分隔不同部分的内容，且这个分隔符在后面很多地方都会用到，因此先用变量 boundary 将其保存下来，便于后面复用。此外，邮件内容包括有文本以及之后会介绍的附件，即消息格式不同，因此 Content-Type 需要设定为“multipart/mixed”。填充完 DATA 头部的各个信息后，将该头部发送出去。

接下来，发送的是邮件的正文，所以 Content-Type 为 text/plain。由于在测试时所传入的往往是文件路径（该文件中会存储正文文本），因此需要先将该文件读入内存，然后将其中的文本内容转换成字符串，接着再进行发送。另外，为了使程序具有更好的扩展性，在设计时还考虑了传入的字符串不是文件路径的情况。

```
    // TODO: Send message data
    const char* boundary = "qwertyuiopasdfghjklzxcvbnm";
```

```
    // send header
    sprintf(buf,
    "FROM: %s\r\nTo: %s\r\nContent-Type: multipart/mixed;
boundary=%s\r\n",
    from, receiver, boundary);
    if (subject != NULL)
    {
        strcat(buf, "Subject: ");
        strcat(buf, subject);
        strcat(buf, "\r\n\r\n");
    }
    send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send DATA header");

    // send message
    if (msg)
    {
        sprintf(buf, "--%s\r\nContent-Type:text/plain\r\n\r\n",
boundary);
        send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send DATA msg
header");
        if (!access(msg, F_OK))
        {
            char *content = get_content_from_file(msg);
            strcat(content, "\r\n");
            send_to_server(s_fd, (void *) content, strlen(content), 0,
"send DATA msg content");
            free(content);
        }
        else
        {
            char *content_end = "\r\n";
            send_to_server(s_fd, (void *) msg, strlen(msg), 0, "send DATA
msg content");
            send_to_server(s_fd, (void *) content_end, strlen(content_end),
0, "send DATA msg content");
        }
    }
```

（5）完成邮件附件的发送。

除了文本之外，本次实验还要求能发送附件。因此，在确定 Internet 消息格式的时候，本人决定使用 **MIME 中的 application/octet-stream 类型**。该类型能告诉邮件客户端，这段消息应当复制进一个文件，再打开，后面如何打开则由用户来操作。因此这种类型能很好地

表示邮件附件。

将要发送的附件读入内存后，同样需要对其进行 Base64 编码转换，转换成字符串之后，发送给服务器。值得注意的是，为了避免内存泄漏，应当及时使用 free()函数释放附件在内存中占用的空间。

完成附件的发送后，可以再发送一次分隔符，告诉服务器邮件内容的传输完成了。

```
    // send attachment
    if (att_path)
    {

        FILE *fp = fopen(att_path, "r");
        if (!fp)
        {
            perror("file not found");
            exit(EXIT_FAILURE);
        }

        sprintf(buf,
        "--%s\r\nContent-Type:application/octet-stream\r\nContent-Dispos
ition:attachment; name=%s\r\nContent-Transfer-Encoding: base64\r\n\r\n",
        boundary, att_path);
        send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send DATA att
header");

        FILE *fp_temp = fopen("temp", "w");
        encode_file(fp, fp_temp);
        fclose(fp);
        fclose(fp_temp);
        char *content = get_content_from_file("temp");
        send_to_server(s_fd, (void *) content, strlen(content), 0, "send
DATA att content");
        remove("temp");
        free(content);
    }
    sprintf(buf, "--%s--\r\n", boundary);
    send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send DATA end");
```

（6）结束邮件内容的发送（使用模板代码中已经定义好的 end_msg），然后使用 QUIT 命令退出。代码如下：

```
    // TODO: Message ends with a single period
    send_to_server(s_fd, (void *) end_msg, strlen(end_msg), 0, "send END");
```

```
    recv_from_server(s_fd, (void *) buf, 0, "recv END");

    // TODO: Send QUIT command and print server response
    const char* QUIT = "QUIT\r\n";
    send_to_server(s_fd, (void *) QUIT, strlen(QUIT), 0, "send QUIT");
    recv_from_server(s_fd, (void *) buf, 0, "recv QUIT");
```

2. 邮件接收客户端详细设计

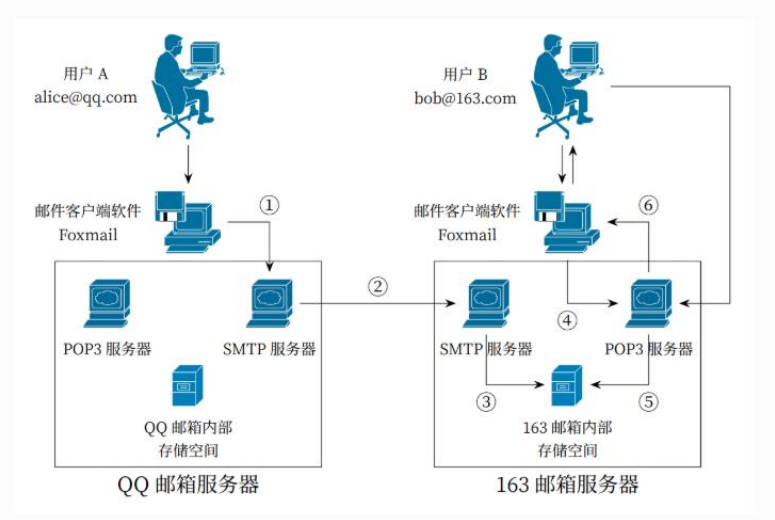
【实验任务与实验要求】

本次实验的第二个任务（recv.c），是在不使用其它第三方库的前提下，仅仅基于 Socket 编程、POP3 协议，实现一个邮件接收客户端。该客户端需要能够与服务器进行交互，分别获取总邮件个数及大小、每封邮件的编号及大小、第一封邮件的内容。

与第一个任务同理，程序要能够使用网络上知名的邮件服务提供商（如 QQ、网易等）进行测试，并且需要打印出交互过程中服务器的回复信息。

【实验总体设计思路】

电子邮件的发送和接收过程如下图所示：



其中涉及了邮件接收的过程包括有④、⑤、⑥。也就是说，在接收邮件时，客户端需要先与 POP3 服务器进行通信，通知 POP3 服务器来取回该邮件供应商内部存储空间中存放的邮件，然后再转交给邮件客户端。

与上一个小节的任务同理，这一过程主要通过 **POP3 协议**来实现。POP3 协议定义了邮件客户端与 POP3 服务器之间的通信规则，常用的命令包括有：USER、PASS、STAT、LIST、RETR、RETR msg、DELE msg、QUIT 等。

因此，使用 POP3 协议来接收邮件的主要步骤如下：

- 服务器发送欢迎消息。
- 客户端输入用户名和密码进行认证，如果认证成功，服务器会返回成功消息。
- 认证成功后，客户端输入一系列命令获取信息，如 STAT、LIST、RETR、DELE 等。
- 客户端发送 QUIT 命令，结束会话。

本小节实验的 Socket 编程部分与上一小节几乎是完全一致的，代码也可以复用，因此不再赘述。

综上所述，**从整体来看，我们所要做的就是**：初始化 Socket 套接字接口，封装好 send()、recv()函数，然后使用封装后的新函数来与服务器进行 POP3 协议的交互。这样就可以完成邮件接收客户端的实现了。接下来，本文会详细介绍各部分的设计思路与代码实现。

【各部分的设计思路与代码实现】

（0）准备工作：初始化 Socket 接口，封装好发送函数、接收函数

该部分要完成的工作，即初始化 Socket 接口、封装发送函数和接收函数，与上一小节的任务是几乎一样的，代码也基本上可以直接复用。因此略去，不再赘述。

（1）进行用户身份认证。值得注意的是在 POP3 协议中不需要进行 Base64 编码。代码如下：

```
// TODO: Send user and password and print server response
sprintf(buf, "USER %s\r\n", user);
send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send USER");
recv_from_server(s_fd, (void *) buf, 0, "recv USER");

sprintf(buf, "PASS %s\r\n", pass);
send_to_server(s_fd, (void *) buf, strlen(buf), 0, "send PASS");
recv_from_server(s_fd, (void *) buf, 0, "recv PASS");
```

（2）使用 STAT 命令查看邮件统计信息（包括邮件总数、邮件大小）。代码如下：

```
// TODO: Send STAT command and print server response
const char* STAT = "STAT\r\n";
send_to_server(s_fd, (void *) STAT, strlen(STAT), 0, "send STAT");
recv_from_server(s_fd, (void *) buf, 0, "recv STAT");
```

（3）使用 LIST 命令列出每个邮件的编号及大小（字节为单位）。代码如下：

```
// TODO: Send LIST command and print server response
```

```
const char* LIST = "LIST\r\n";
send_to_server(s_fd, (void *) LIST, strlen(LIST), 0, "send LIST");
recv_from_server(s_fd, (void *) buf, 0, "recv LIST");
```

（4）使用 RETR 命令获取第一封邮件，并打印其中的内容。由于 buf 的大小有限，因此需要分成多次进行打印。代码如下：

```
// TODO: Retrieve the first mail and print its content
int size;
const char* RETR1 = "RETR 1\r\n";
send_to_server(s_fd, (void *) RETR1, strlen(RETR1), 0, "send RETR 1");
r_size = recv_from_server(s_fd, (void *) buf, 0, "recv RETR 1");
sscanf(buf, "+OK %d octets", &size);
size -= r_size;
while (size > 0)
{
    r_size = recv_from_server(s_fd, (void *) buf, 0, "recv RETR 1");
    size -= r_size;
}
```

（5）使用 QUIT 命令退出。代码如下：

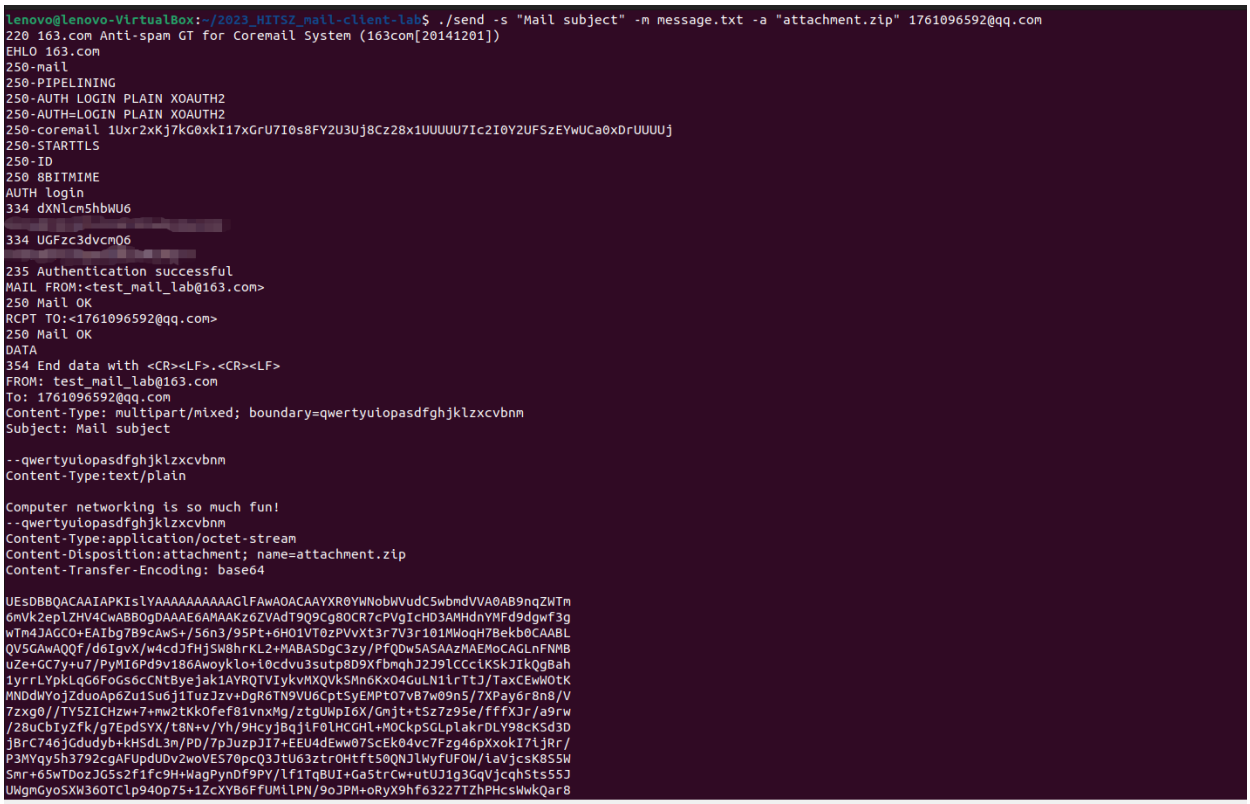
```
// TODO: Send QUIT command and print server response
const char* QUIT = "QUIT\r\n";
send_to_server(s_fd, (void *) QUIT, strlen(QUIT), 0, "send QUIT");
recv_from_server(s_fd, (void *) buf, 0, "recv QUIT");

close(s_fd);
```

二、 实验结果截图及分析

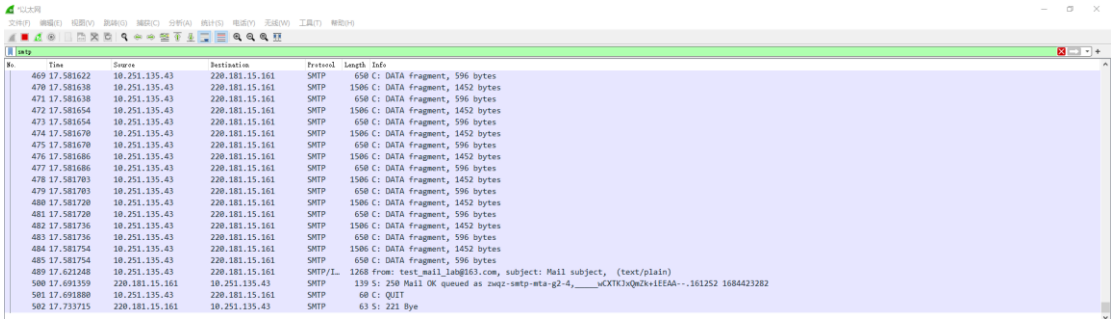
（对你自己实验的测试结果进行评价）

1. 邮件发送客户端实验结果及分析

成功使用了本人的 163 邮箱，发送邮件给另一个 QQ 邮箱，并打印出了与服务器的完整交互过程（个人隐私信息部分进行了打码处理。另外附件部分太长了，因此略去中间部分，只截图了附件的开始、结尾部分），如下图所示：



通过使用 wireshark，可以成功捕获到相应的报文，如下图所示：



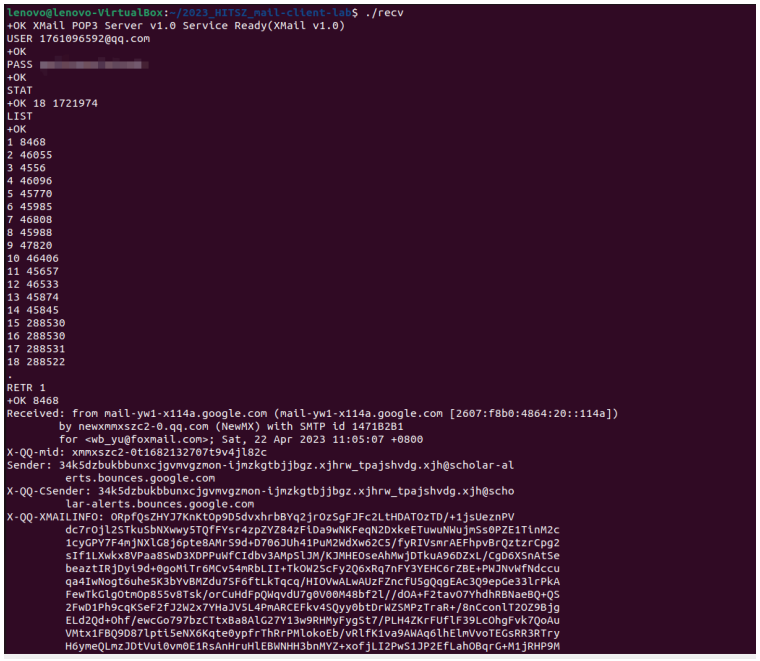
No.	Time	Source	Destination	Protocol	Length	Info
469	17.581622	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
470	17.581638	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
471	17.581638	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
472	17.581654	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
473	17.581654	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
474	17.581670	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
475	17.581670	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
476	17.581686	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
477	17.581686	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
478	17.581703	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
479	17.581703	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
480	17.581720	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
481	17.581720	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
482	17.581736	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
483	17.581736	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
484	17.581754	10.251.135.43	220.181.15.161	SMTP	1506	C: DATA fragment, 1452 bytes
485	17.581754	10.251.135.43	220.181.15.161	SMTP	650	C: DATA fragment, 596 bytes
489	17.621248	10.251.135.43	220.181.15.161	SMTP/I…	1268	from: test_mail_lab@163.com, subject: Mail subject, (text/plain)
500	17.691359	220.181.15.161	10.251.135.43	SMTP	139	S: 250 Mail OK queued as zwqz-smtp-mta-g2-4,____wCXTK3xQmZk+iEEAA--.161252 1684423282
501	17.691880	10.251.135.43	220.181.15.161	SMTP	60	C: QUIT
502	17.733715	220.181.15.161	10.251.135.43	SMTP	63	S: 221 Bye

QQ 邮箱也能收到含有附件的邮件：

综上，邮件发送客户端的功能实现正确。

2. 邮件接收客户端实验结果及分析

成功获取了本人 QQ 邮箱的总邮件个数及大小、每封邮件的编号及大小、第一封邮件的内容等信息。截图如下所示（个人隐私信息部分进行了打码处理。另外邮件内容部分太长了，因此略去中间部分，只截图了邮件内容的开始、结尾部分）：



使用 wireshark 也能捕获相应的报文，部分截图如下：

9463	1076.232000	10.251.135.43	183.47.101.192	POP	60 C: STAT
9465	1076.293645	183.47.101.192	10.251.135.43	POP	70 S: +OK 18 1721974
9466	1076.294217	10.251.135.43	183.47.101.192	POP	60 C: LIST
9468	1076.337571	183.47.101.192	10.251.135.43	POP	235 S: +OK
9469	1076.338003	10.251.135.43	183.47.101.192	POP	62 C: RETR 1

综上，邮件接收客户端的功能实现正确。

三、 实验中遇到的问题及解决方法

（包括设计过程中的错误及测试过程中遇到的问题）

- 问题 1：在配置邮件客户端的实验环境时遇到了困难。
  - **说明及解决办法：**一开始以为这个实验像协议栈实验一样，可以 Windows 下完成，结果发现需要 Linux 环境（本人的电脑没有安装 Linux 系统）。本来想用实验中心的远程实验平台，结果发现指导书上不建议用远程实验平台（因为该平台不支持 wireshark）。最后只能用实验楼电脑的虚拟机来完成，操作系统为 Ubuntu。但实验楼经常会上课，其中的电脑还会时不时地关机，所以耽误了一些进展速度。

- 问题 2：实验刚开始时，对 Socket 编程、SMTP 协议、POP3 协议等内容的相关知识不够熟悉，缺乏上手思路，进展缓慢。
  - **说明及解决办法：**只能静下心来反复阅读 PPT 和指导书。指导书为了降低上手难度，一开始没有让我们考虑 Socket，而是引导我们去“玩”一下 Telnet 命令行操作，让我们体验一把如何在 Linux 终端下使用 POP3 协议、SMTP 协议与服务器进行交互。成功熟悉上述交互方式后，能大大降低我们再去用 Socket 编程重新实现这一过程的难度。一步一步地来，最终也能建成高楼大厦。

- 问题 3：对部分实验的细节之处把握不到位，编程完成后遭遇了 bug。
  - **说明及解决办法：**在利用 Socket 接收并打印出来自服务器的应答信息时，出现了 Segmentation Fault 的错误。仔细检查过后，发现原因在于自己没有理解清楚 strlen() 函数的用法，对 buf 进行下标访问的时候造成了数组越界。另外，一开始我没有接收到来自服务器的应答，仔细检查后发现原来是漏了“\r\n”。以上问题都能通过耐心调试得到解决。

四、 实验收获和建议

（关于本学期计算机网络实验的三种类型：配置验证实验、协议栈系列实验、Socket 编程实验，请给出您对于这三种类型实验的收获与体会，给出评论以及改进的建议。）

收获与体会：

- **从总体上看**，这三种类型的实验，分别涉及不同方面，形式多样、互为补充，几乎涵盖到了计算机网络协议栈的每一层，较为全面地覆盖了本学期《计算机网络》的授课内容。
- 关于**配置验证实验**，这部分需要我们自己设计的内容并不多，绝大部分操作都是“按图索骥”地按照指导书的步骤说明进行。所使用到的工具（Cisco Packet Tracer）能具体而全面地展现各台主机、服务器、路由器之间的通讯、交互情况，帮助我对 VLAN 的作用与特点、动态路由协议与 RIP 协议、网络地址转换技术等内容有了更充分的思考、更深入的理解、更形象的认知。
- **协议栈系列实验**包括了 Eth、ARP、IP、ICMP、UDP 等协议的实现。该系列实验较多地涉及了对各协议字段（例如，源/目的 IP 地址、源/目的 MAC 地址、校验和、协议号等）的填充或解析操作。通过处理这些协议字段的细节，我在无形之中对各层协议的数据报结构有了更透彻的认识。正所谓“纸上得来终觉浅”，如果仅仅停留在“看”，而没有上手 coding，是很难有这样的深刻认识的。
- **Socket 编程实验**，要求我们自己动手“造轮子”，即仅使用 Socket、不使用其它第三方库开发一个邮件客户端，完成邮件收发。本人在大一立项的时候开发过邮件客户端（不过用了较多 Python 库函数，显然不符合本次实验的要求）；本人在大二下的 Android 开发课上也接触过 Socket 编程。因此这个任务对于我来说并不陌生。但是，在克服了所遇到的重重困难完成本次实验后，我发现自己还是能获得很多新体会。在大一立项的时候，关于 SMTP、POP3 的具体交互过程，我使用的是别人已经实现并封装好的库函数，缺乏对底层细节的了解。但在本次实验中，我仅仅基于 Socket 来实现 SMTP、POP3，真正做到了“自己动手，丰衣足食”，获得了真正的能力锻炼。
- 通过完成这三种类型的实验，我在实践中深化了对理论课内容的理解，并进一步提升了自己的 C 语言编程水平，也熟悉了 wireshark、Cisco Packet Tracer、Cmake、Telnet、Socket 等工具或命令的使用方式，为今后使用计算机网络的知识解决实际问题奠定坚实基础。
- 总的来说，本学期《计算机网络》的课程实验还是让我收获较大。

评论与建议：

- 总体上，本学期《计算机网络》的课程实验还是让我获得了良好的体验，配套的图文讲解非常到位，老师甚至还录制了演示视频。代码框架也比较完善，需要我们实现的地方也都添加了“TO-DO”注释进行提示，便于我们快速上手开发。
- 建议邮件客户端实验环境的搭建部分可以讲解得更详细一些（我在这上面折腾了不少时间）。此外，本课程总共有 9 个实验任务（均需提交分析报告），但只安排了 6 次实验课，时间略微有点不够。希望之后可以适当增多留给学生完成任务的时间。