# 实验三 Socket 编程实现网络通信 实验报告


> 姓名：李宇飏
>
> 班级：自05
>
> 学号：2020011645


## SMTP

### 实验内容

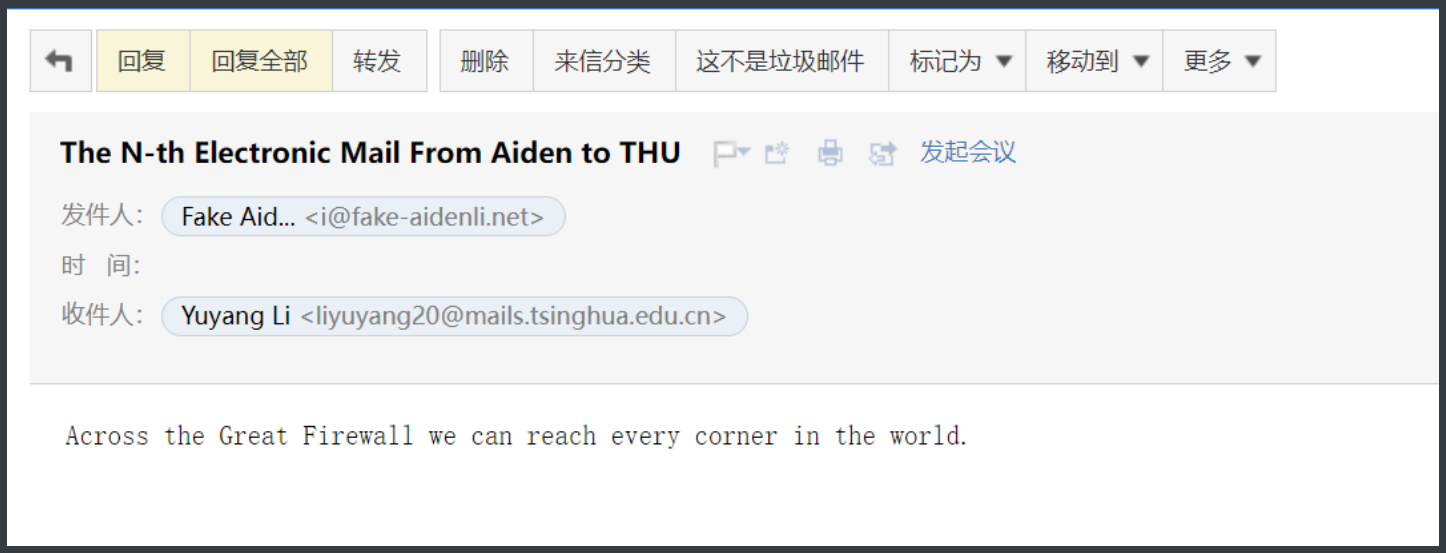
1. 根据 SMTP 协议（RFC2821），补全无身份验证的 SMTP 程序；
2. 根据 SMTP 拓展协议（RFC2554），补全含有身份验证的 SMTP 程序。

### 实验过程

#### 补全 `SimpleSender.java`

补全代码请见 `CodeBase/SMTP/SimpleSender.java` 标注 `// DONE: x` 处。编译并运行，命令行输出见 `CodeBase/SMTP/SimpleSender.out` 。

验证在邮箱 `liyuyang20@mails.tainghua.edu.cn` 收到了对应邮件：




## 补全 `AuthSender.java`

补全代码请见 `CodeBase/SMTP/AuthSender.java` 标注 `// DONE: x` 处。编译并运行，命令行输出见 `CodeBase/SMTP/AuthSender.out`，密码用 `*` 代替。

验证在邮箱 `i@aidenli.net` 收到了对应邮件：


# TCP 聊天室

## 实验内容

1. 使用 Python 编写基于 Socket 的多人文字聊天室
2. 测试聊天室功能，包括：连接、收发消息、断连处理

## 实验过程

### 编写聊天室

首先编写客户端、服务器类，由于二者具有一些相似的功能（比如创建接收/发送信息线程、记录 Log 等），首先编写父类 `ChatPeer`，然后派生 `ChatClient`，`ChatServer`。

服务器主要功能为：

1. 创建 `welcomeSocket`，监听客户端连接；
2. 维护 `clientPool`，将连接的客户端的客户端 accept 为 `serverSocket`；
3. 通过每个 `serverSocket` 监听并从客户端接收消息，并转发给其他所有连接；
4. 对于发送失败的 `serverSocket`，关闭与客户端的连接。

客户端主要功能为：

1. 创建 `clientSocket`，连接服务器；
2. 通过 `clientSocket` 监听并从服务器接收消息，并打印内容，同时显示发送方 IP、端口、时间；
3. 通过 `clientSocket` 向服务器发送消息；

此外，客户端、服务器均应当：

1. 充分记录 log，以 `INFO`，`DEBUG`，`ERROR` 等级别区分；
2. 妥当处理所有潜在的 exception 抛出；
3. 妥当地新建、维护和关闭线程，使前端非阻塞。

为了前端非阻塞，需要采用多线程/异步方法。为简洁记，本次采用 `threading.Thread` 通过多线程实现。具体地，

- `welcomeSocket` 及每个 `serverSocket` 单独新建线程，前者在服务器端程序结束时释放，后者在该 socket 连接失败后释放；
- `clientSocket` 单独新建线程，并在结束连接时释放；
- 客户端、服务端在发送每条消息时新建线程发送，在发送结束后释放。

此外，本次实验还是先了一些额外功能，包括但不限于：

1. 消息类型，目前支持：文本消息 `MSG`，控制消息 `CTL`（用于上线、下线通知）。
2. 用户上线、下线通知；
3. 用户端命令：如可以通过 `:q` 断开连接，通过 `::` 转义（如 `::q` 代表发送文本 ":q"）。

源码见 `CodeBase/TCP-Chat`。

### 测试聊天室

聊天室不包含额外依赖，可以直接使用，在 Python 3.8-3.9 测试正常。

运行服务器：

```
python run.py --mode server --name Server --server_port PORT --log
log/server.log
```

运行客户端：

```
python run.py --mode server --name Alice --server_addr ADDR --server_port
PORT --log log/alice.log
```

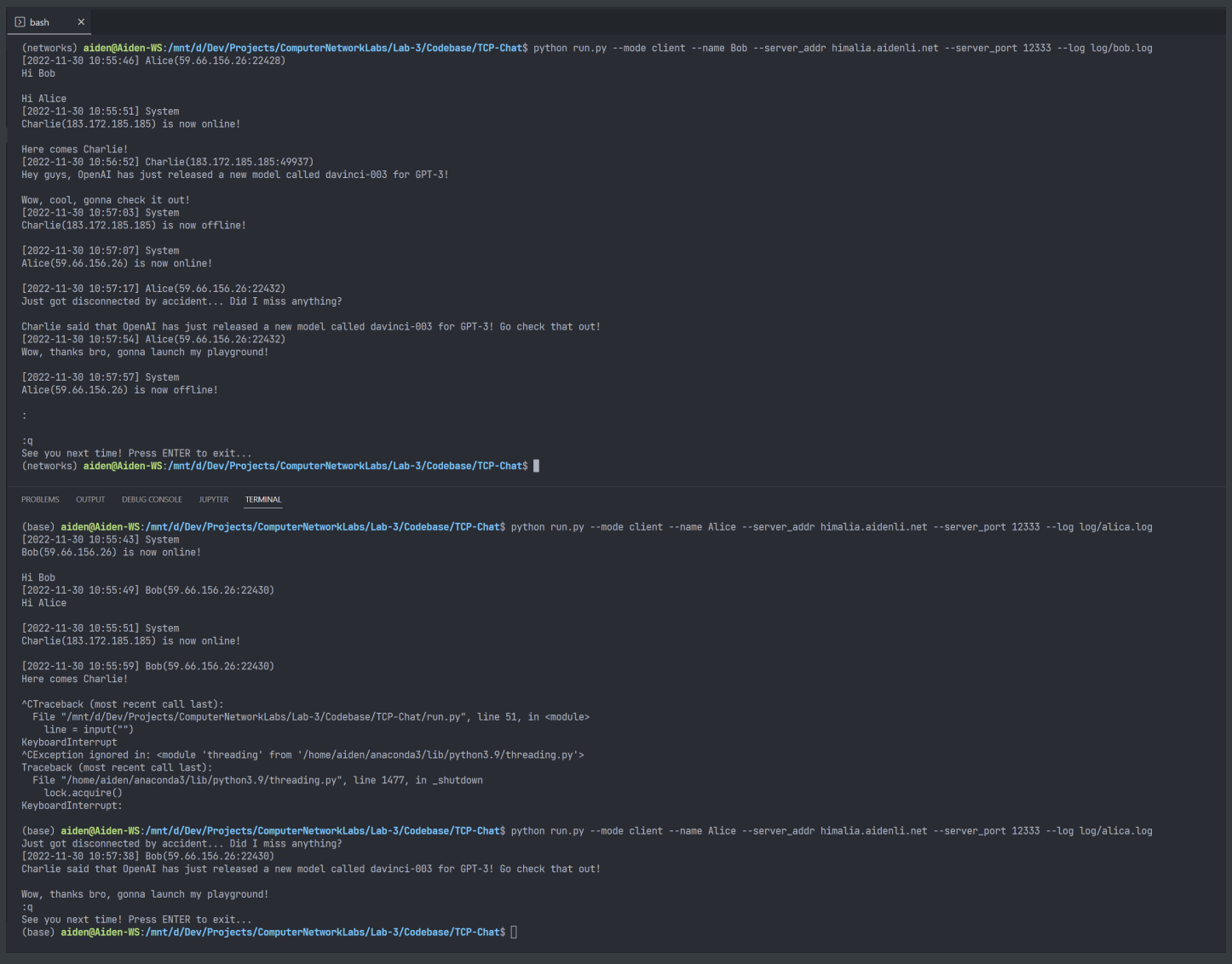
其中，

- `ADDR`，`PORT` 为服务器对外开放服务的地址与端口（由 `welcomeSocket` 绑定）
- 服务端不需要指定 `server_addr`
- 而应当允许防火墙在 `server_port` 端口的 TCP 出入连接。

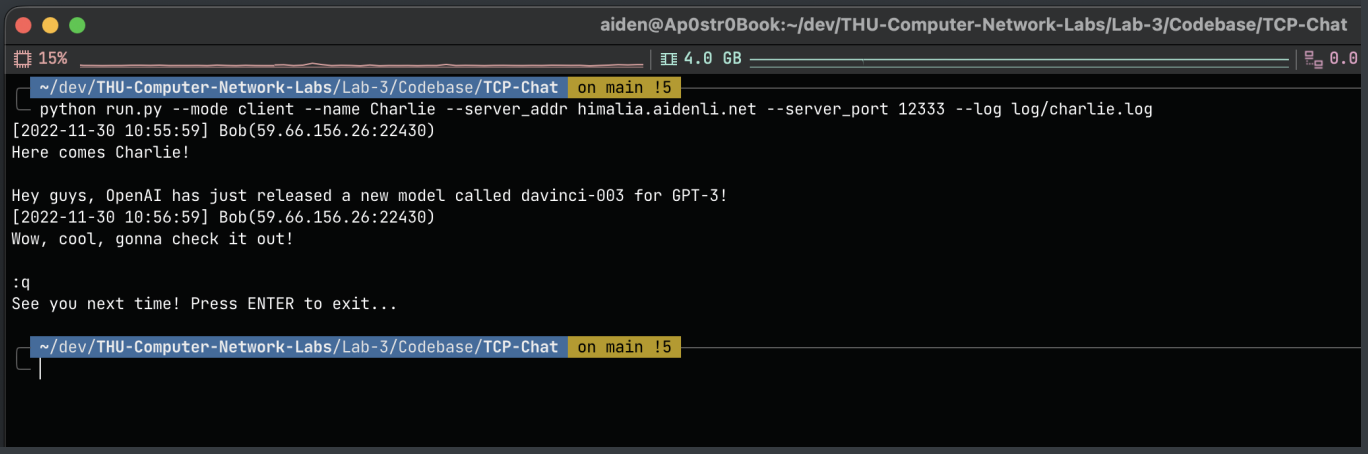
在服务器上启动服务器，以 `himalia.adeinli.net:12333` 为例，分别从 2 台设备的 3 个 shell 启动客户端连接服务器，测试：

1. 连接服务器；
2. 收发消息；
3. 手动退出程序；
4. 客户端程序中断，不影响服务端；重新连接并回到会话。

## Alice、Bob （同一个出口 IP，不同端口）

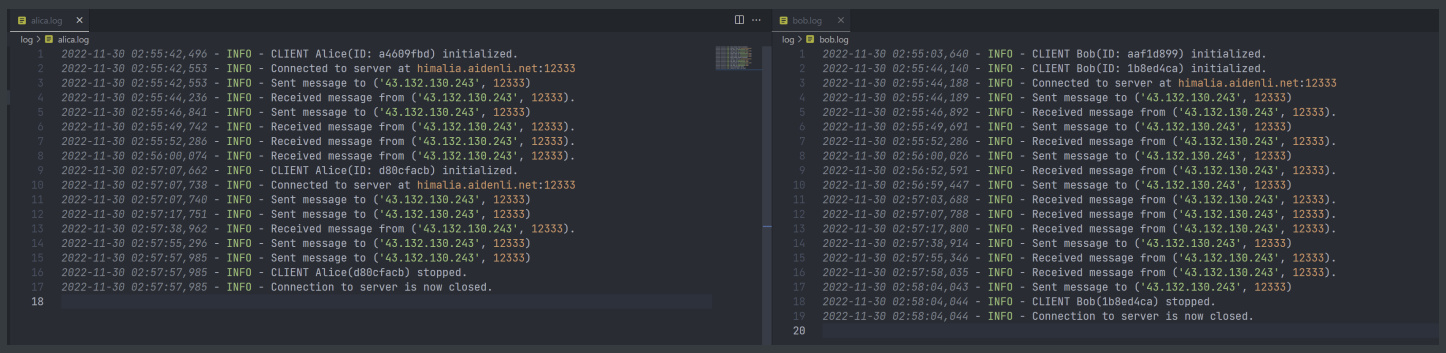


## Charlie（不同出口 IP）

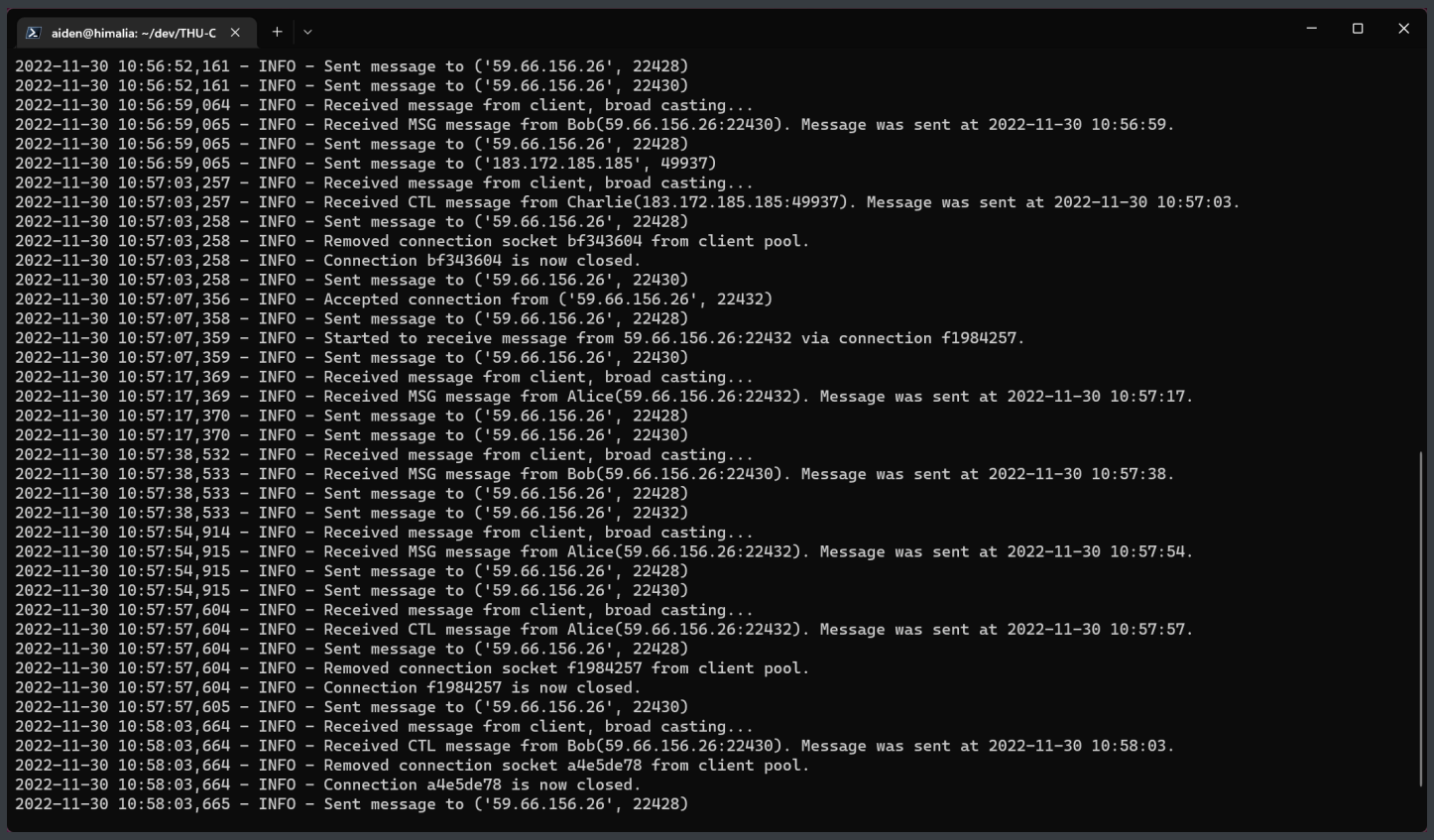



## Log 数据

客户端，以 Alice、Bob 的 log 为例：




服务器端（节选）：




# 思考题与分析题

## Simple SMTP 和常用的 E-mail 客户端在功能结构上的比较

Simple SMTP 仅仅是提供了一个发送邮件的功能，相比常见 E-Mail 客户端，还缺少若干功能，包括但不限于：

- 身份验证功能（在 AuthSMTP 里部分实现）
- 收件、通过 IMAP/POP3 拉取收件箱
- 草稿、已发件等其他结构功能
- 发送附件或富文本内容
- 使用 TLS 等加密数据

## 使用 TCP 和 UDP 各自的优缺点比较

| 优点 | TCP | UDP |
| --- | --- | --- |
| 无需建立连接 | 否 | **是** |
| 可靠数据连接 | **是** | 否 |
| 拥塞机制/流量控制 | **有** | 无 |
| 支持超时重传 | **是** | 否 |
| 实时性↑ | 相比较低 | **相比较高** |
| 资源消耗↓ | 相比较高 | 相比较低 |

### 针对程序中出现的问题及解决方法，写出实验体会

实验中主要遇到了以下问题：

1. 若客户端意外断开连接，服务器端维护的 `serverSocket` 保持，在线程发送模式下，若不做异常处理，则会在下次发送时报错

   解决方案：为线程设置 `Event()` 机制、 `ExceptionHandler` 处理错误。
2. 客户端连接和下线时直接关闭连接，导致服务端需要通过异常处理的方式结束链接

   解决方案：加入新消息类型 `CTL` ，通过该类消息控制连接。

实验感想

通过此次实验，我感受到了网络通讯中，为了保障程序正常和消息收发正常，而需要考虑的状态机的复杂性。尽管此次实验中，程序的状态机较为简单，但在复杂场景中，有更多需要考量的因素和情况。

此外，我体会到了 TCP 向上提供可靠数据连接的易用性。即使 TCP 自身处理报文收发、确认、重传等机制十分复杂，但在基于 Socket 编程时，这些机制已经被封装好并向上提供可靠数据连接。

此外，此次实验没有考虑数据安全性，且客户端、服务端仅仅实现了基础操作功能，并采用手动的线程管理模式，可在实验结束后继续完善。