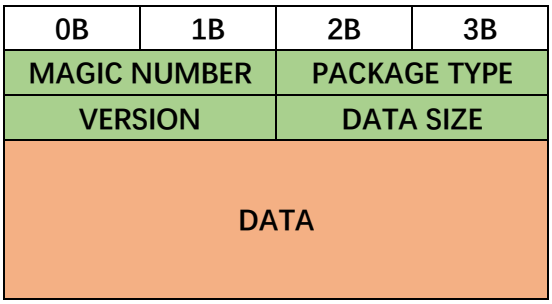
计算机网络实验报告 1——聊天程序的设计和实现（提高要求）

一、协议设计

多人聊天程序使用二进制协议，协议分为两层。首先分辨信息的类型，再依据信息的类型对信息进行特定格式的解析。

1 数据基本格式



MAGIC NUMBER：一个给定常数。

PACKAGE TYPE：表示消息的类型，目前有 8 种类型

- **TYPEERR：**用于服务端回复，接收到类型未知的数据。
- **GREET：**用于客户端测试连通性，服务端应原样回复。
- **LOGIN：**用于登录，登录信息以特定格式保存在 DATA 中。
- **LOGINOK：**用于服务端回复，登录成功。
- **LOGINERR：**用于服务端回复，登录失败（目前唯一原因是名称被占用）。
- **USER_UPATE：**用于服务端向客户端通知房间内成员变化，信息以特定格式保存在 DATA 中。
- **PUSH_MSG：**用于服务端向客户端通知其他成员发送的消息，信息以特定格式保存在 DATA 中。
- **SEND：**用于客户端通过服务端向其他成员发送消息，信息以特定格式保存在 DATA 中。

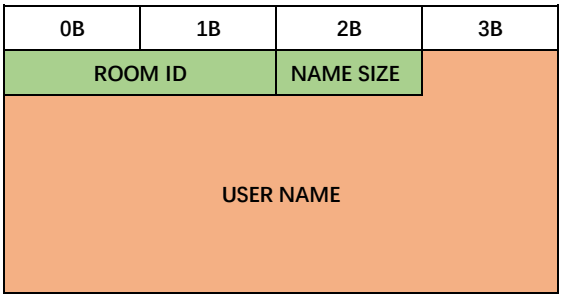
VERSION：表示软件版本。

DATA SIZE：表示变长数据 DATA 的尺寸。

DATA：表示特定类型的附加信息。

2 登录附加信息格式

登录附加信息即 **PACKAGE TYPE** 为 **LOGIN** 的数据的 DATA 内容部分的数据格式。

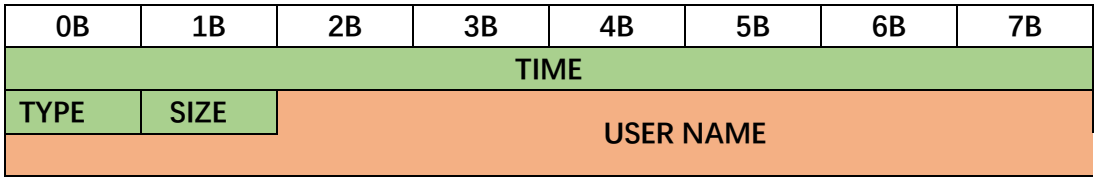


ROOM ID: 表示客户端要加入的房间的编号。

NAME SIZE: 表示变长用户名字符串的长度。

USER NAME: 客户端设置的用户名，C 语言格式字符串。同时在 **LOGINERR** 类型中也被用于保存登录失败原因；在 **LOGINOK** 类型中用于保存房间内其他客户端名称列表。

3 房间内容成员变化附加信息格式



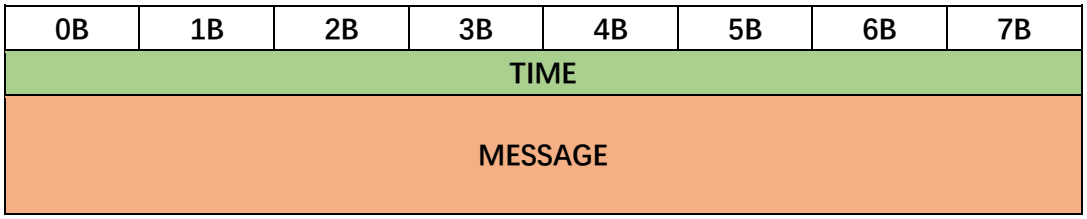
TIME ： 表示成员变化发送的时间。

TYPE ： 表示成员变化的类别，有进入房间、离开房间两类。

SIZE： 表示成员名称的长度。

USER NAME： 表示成员名称，C 语言格式字符串。

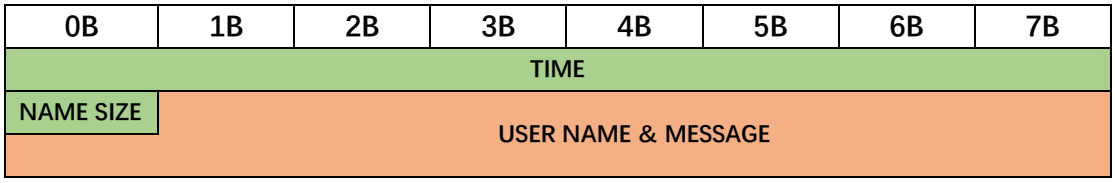
4 客户端发送消息的附加信息格式



TIME： 表示发送消息的时间。

MESSAGE： 表示发送的消息，C 语言格式字符串。

5 服务端通知消息的附加信息格式



TIME：消息被客户端发出的时间。

NAME SIZE：发出客户端的名称。

USER NAME & MESSAGE：两段 C 语言格式字符串，先客户端名称，后消息。

二、程序设计

程序要实现的功能是客户端-服务器模型的多人分组聊天功能，用户通过房间号加入不同的房间实现分组聊天，同一房间内客户端可以收到彼此消息，在不同房间不能收到其他房间的消息。

1 客户端

1) 客户端测试和服务器的连通性，验证服务端正确性。

2) 客户端向服务端发送登录信息，汇报名称和加入的房间号，等待服务器回复。

3) 若登录失败，显示失败原因。若成功，则开始聊天，开启一个新线程不断接受并解析来自服务端的数据，将接受到的聊天信息保存在本地缓存中，不立即显示，主线程则处理和解析用户输入。

4) 用户可通过输入指令显示当前缓冲区中的聊天信息，可通过指令发送消息。

2 服务端

服务器用于实现分组和信息的转发。

工作模式上，使用单线程事件响应模型，借助 SELECT（使用比较简单） 系统 API（轮询非阻塞 SOCKET 容易造成忙等待）获得可读和可写套接字。对于可读套接字使用阻塞套接字 recv()函数读取所有数据并解析处理，回复消息不立即写回，而是暂存；对于可写套接字使用阻塞套接字 send()函数将暂存数据一次性写回。

分组处理上，为每个客户端建立一个到房间集合的映射，并将客户端加入同一房间的集合内，转发消息时向同一房间内所有客户端转发。

程序启动时不预分配房间，当某房间的第一个客户端登录时，分配内存创建房间；当最后一个客户端退出房间后，销毁房间。

三、程序实现

1 协议表示

用结构体表示二进制协议。

```
enum class PkgType :uint16_t
{
    TYPEERR,    //size=0
    GREET,      //size=0
    LOGIN,      //size=val
    LOGINOK,    //size=val
    LOGINERR,   //size=val
    USER_UPATE, //size=val
    PUSH_MSG,   //size=val
    SEND,       //size=val
    SENDTO,     //size=val
};
```

```
struct Pkg
{
    uint16_t magic;
    PkgType type;
    uint16_t version;
    uint16_t bodySize;
    uint8_t data[];
};
```

```
struct UserUpdate {
    time_t time;
    enum class UpdateType:uint8_t
{
        ENTER,
        LEAVE,
    }type;
    uint8_t strsize;
    char name[];
};
```

```
struct LogInData {
    int roomId;
    uint8_t strsize;
    char username[];
};
```

```
struct MsgData {
    time_t time;
    char msg[];
};
```

```
struct PushData {
    time_t time;
    uint8_t namesize;
    char name_and_msg[];
};
```

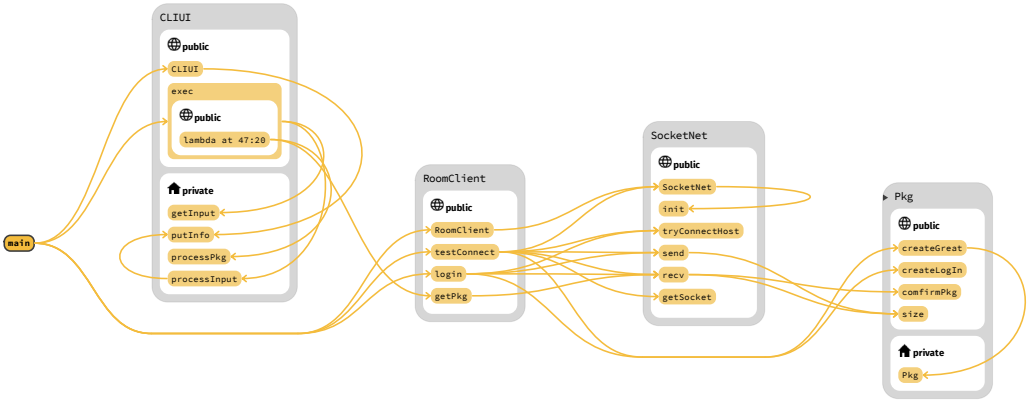
2 核心实现

程序的配置信息通过命令行参数给出。

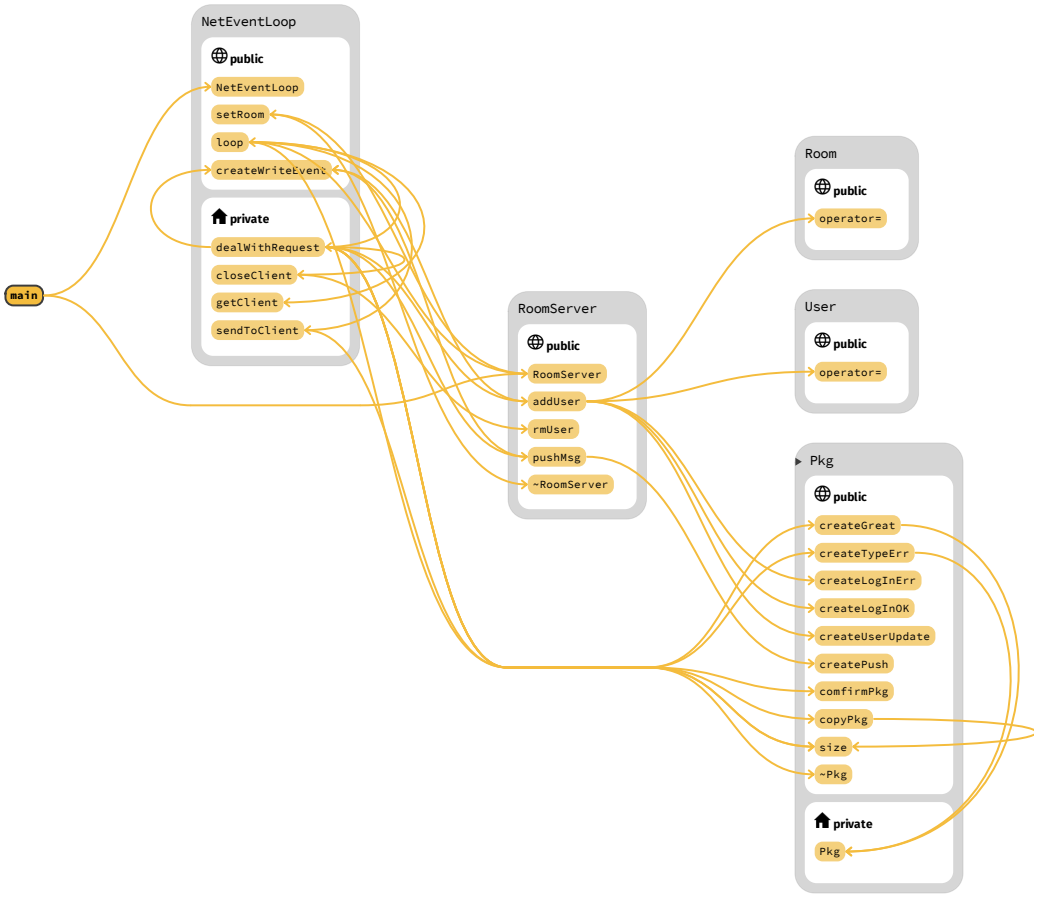
服务端网络处理部分逻辑如下。

```
void NetEventLoop::loop()
{
    fd_set readFds, writeFds, expFds;
    while (!stop) {
        FD_ZERO(&readFds);
        FD_ZERO(&writeFds);
        for (const auto& pair : writeEvents) {
            FD_SET(pair.first, &readFds);
            if (pair.first != listenSocket)
                FD_SET(pair.first, &writeFds);
        }
        select(0, &readFds, &writeFds, NULL, NULL);
        for (int i = 0; i < readFds.fd_count; i++) {
            if (readFds.fd_array[i] == listenSocket) {
                getClient();
            }
            else {
                dealWithRequest(readFds.fd_array[i]);
            }
        }
        for (int i = 0; i < writeFds.fd_count; i++) {
            SOCKET clientSocket = readFds.fd_array[i];
            if (writeEvents.find(clientSocket) != writeEvents.end()) {
                queue<Pkg*>& writeQueue = writeEvents[clientSocket];
                while (!writeQueue.empty()) {
                    Pkg* pPkg = writeQueue.front();
                    sendToClient(clientSocket, pPkg);
                    delete pPkg;
                    writeQueue.pop();
                }
            }
        }
    }
}
```

3 客户端整体框架



4 服务端整体框架



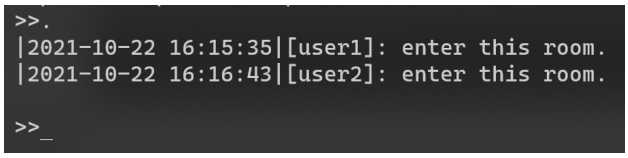
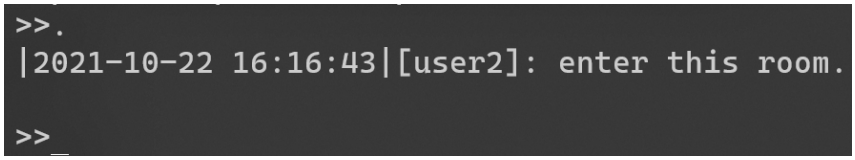
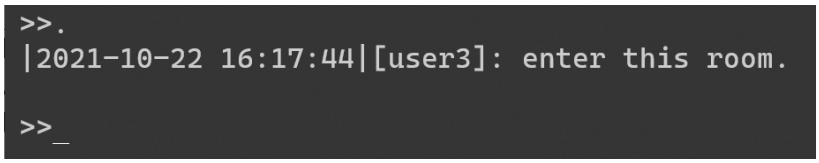
四、程序运行说明

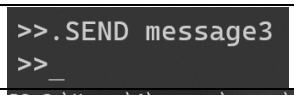
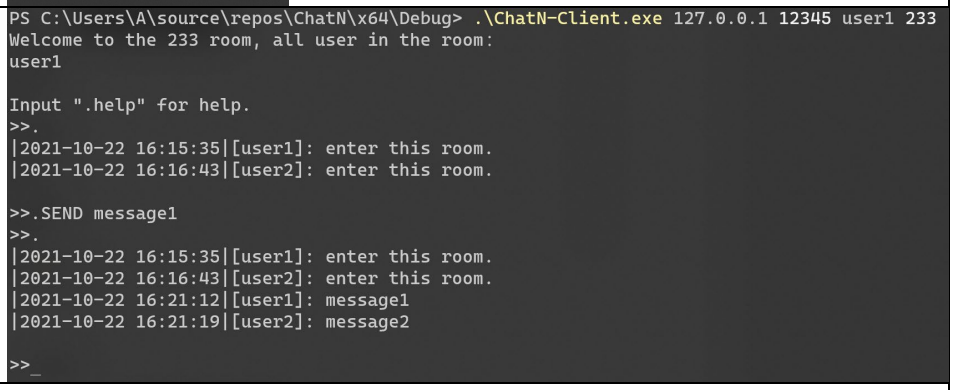
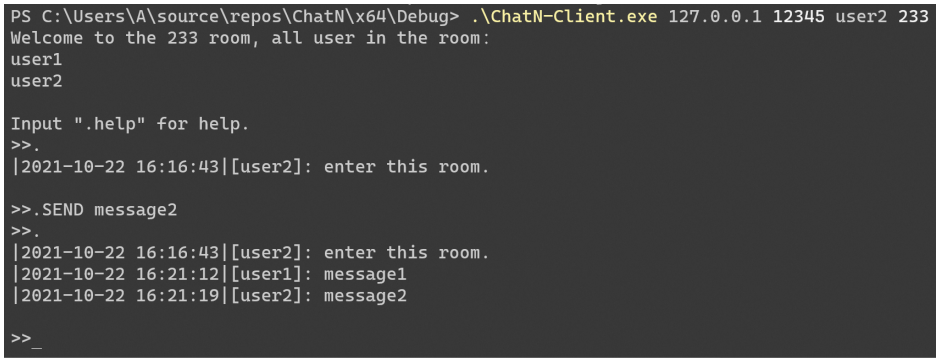
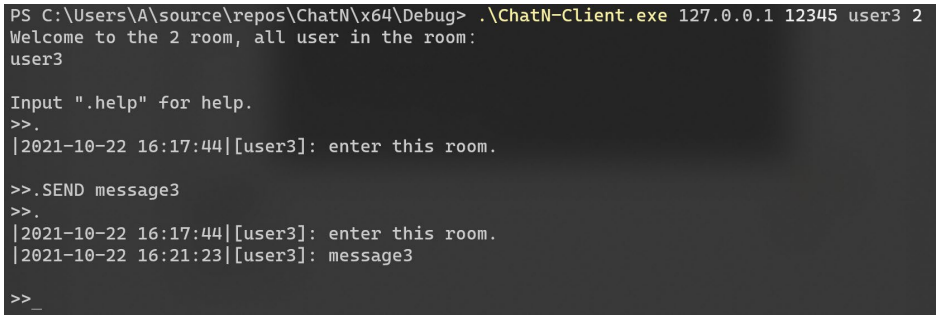
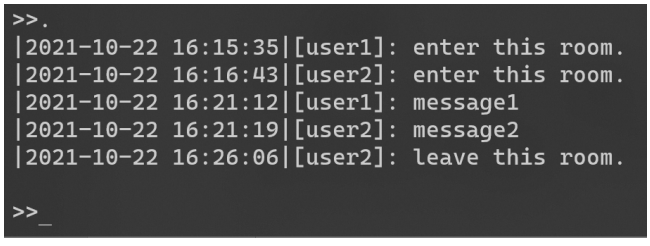
服务端程序直接启动即可，便于调试 IP 端口默认为 127.0.0.1:12345

客户端使用方法如下：**需要在命令行参数中给出服务端 IP，服务端端口，名称，加入的房间号。**

1 运行演示

运行过程如下：

操作	运行截图
启动服务端	
用户 1 进入 233 号房间	
用户 2 进入 233 号房间	
用户 3 进入 2 号房间	
用户 1 查看当前聊天信息	
用户 2 查看当前聊天信息	
用户 3 查看当前聊天信息	
用户 1 发送消息 1	
用户 2 发送消息 2	

用户 3 发送消息 3	
用户 1 查看当前聊天信息	
用户 2 查看当前聊天信息	
用户 3 查看当前聊天信息	
用户 2 退出	正常退出应输入.QUIT。此处 CTRL+C 直接中断。
用户 1 查看聊天信息	

2 演示分析

用户 1 和用户 2 同在 233 房间，可以收到彼此消息，不能收到用户 3 消息；用户 3 在 2 房间，不能收到其他房间内用户 1 和用户 2 的消息。通过划分房间实现了分组聊天。

同一房间内客户端可以收到其他客户端进入和退出的消息，服务端可正常处理客户端的异常退出。

五、**总结**

问题 1：数据粘包问题

多个数据包可能会同时被写入接收端的缓存中，这是需要根据数据包长度和时机接收长度进行分包。