



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

计算机网络实验报告

流式 socket 编程

于文明

年级：2020 级

专业：信息安全

指导教师：张建忠，徐敬东

2022 年 10 月 22 日

摘要

关键字：socket,TCP,Protocol Design

目录

一、 实验要求	1
二、 模块划分	1
(一) 模块流程图	1
(二) 客户端	2
1. 主线程	2
2. 接收线程	2
(三) 服务端	2
1. 主线程	2
2. 子线程	2
(四) 消息处理	3
三、 协议设计	3
(一) 时序	3
(二) 语法和语义	4
1. INFO 结构体	4
2. 收发消息缓冲区	4
3. 对应的类型转换	4
四、 核心代码	5
(一) socket 初始化函数	5
1. print_std	5
2. 发送消息	6
五、 附加功能实现	7
(一) C-S-C 转发模式	7
(二) 多线程	7
(三) 群聊	7
(四) 私聊	7
(五) 命令	7
(六) 消息格式设计	8
六、 实验结果	8
(一) 开启服务端	8
(二) 开启客户端	8
1. 开启客户端 1	8
2. 开启客户端 2	8
(三) 服务器日志	9

（四） 聊天室消息发送和接收 (群聊,c-s-c)	9
1. 发送	9
2. 所有用户接受	9
（五） 指定用户消息发送和接收 (私聊)	9
（六） 退出命令	10
七、 附录	10

一、 实验要求

- 1) 使用流式 Socket, 设计一个两人聊天协议, 要求聊天信息带有时间标签。需要完整说明交互信息的类型、语法、语义、时序等具体的信息处理方式
- 2) 对聊天程序进行设计, 给出模块划分说明。模块的功能和模块的流程图
- 3) 在 Windows 系统下, 利用 C/C++ 对设计的程序进行实现。程序界面可以采用命令行方式, 但需给出使用方法。
- 4) 对实现的程序进行调试
- 5) 撰写实验报告, 并将实验报告和源码提交至本网站

二、 模块划分

程序整体可以分为两大模块: 服务端和客户端, 具体的分发消息控制由模块内的不同线程控制。因为程序整体采用了 C/S 架构, 设计了 C-S-C 的转发模式¹, 消息经过客户端发送, 在服务端接受, 再经由服务端转发至用户端。

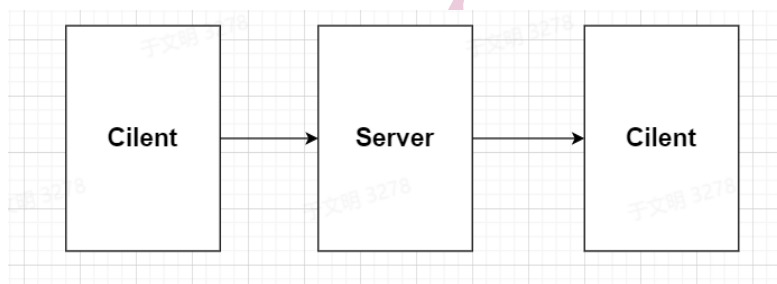


图 1: c2s2c 转发模式

(一) 模块流程图

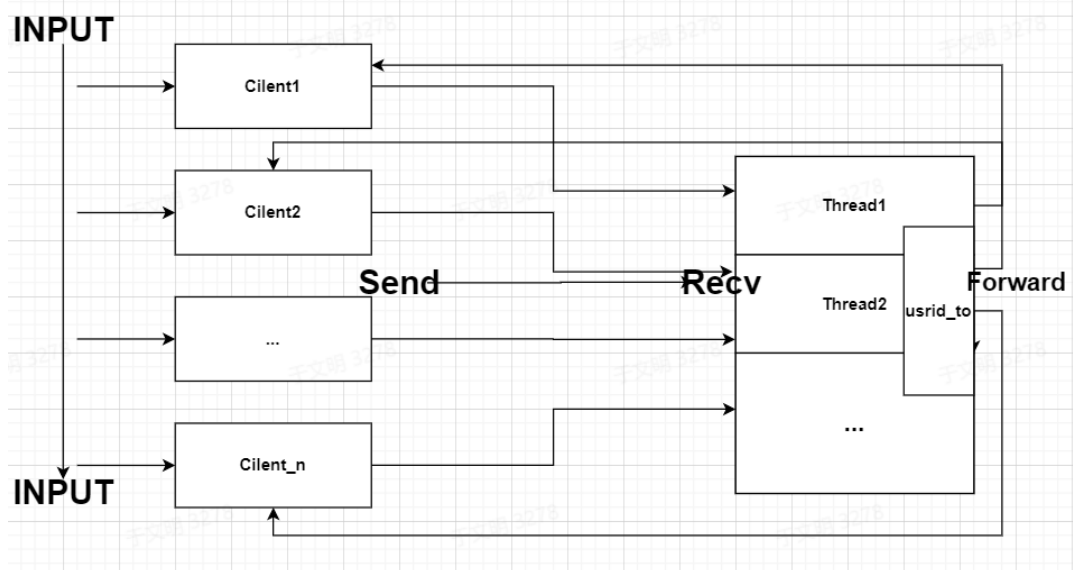


图 2: C/S 模块流程图

(二) 客户端

1. 主线程

在客户端启动后，主线程会完成以下工作：

- 初始化 socket, 包括 WSA 的初始化, Cilent socket 和 Cilent address 的初始化, 注意这里没有使用 bind 套接字绑定函数。
- 向服务器发起连接请求 (使用 connect 函数)
- 创建接收线程, 用来接收服务器的消息 (和发送消息是异步时序)
- 等待用户输入 (输入阻塞线程), 对输入文本进行识别匹配 (比如识别匹配是否是特殊命令, 如 sendto, exit 等, 默认为 sendtoall), 进行相应处理, 将消息发送至服务器端 (send)
- 上一步识别到发送 exit 字符串时, 结束线程

2. 接收线程

在客户端启动 (程序需要先启动服务端, 再启动客户端) 后, 接收线程会完成以下工作

- 主线程启动接收线程 (在 main 函数)
- 使用循环等待服务器发送消息, 按照消息的类型属性匹配不同的处理 (这里使用 switch case 结构完成, 具体见第 (五) 部分))

(三) 服务端

1. 主线程

在服务端启动时, 主线程会完成以下工作

- 初始化 socket
- 建立 socket 监听 (listen)
- 接收连接请求 (accept), 如果没有连接请求, 这里会阻塞
- 接受请求后记录 cilent 用户信息 (程序中用的是 userusrid, usurname 的结构体和全局向量 userlist<user> 来记录)
- 启动子线程, 用来处理对应 cilent 用户的消息接收和发送以及一些其他命令
- 继续等待其他用户连接

2. 子线程

在子线程启动后, 会完成以下工作:

- 进入循环, 接收用户消息 (recv)
- 根据消息类型, 进行对应处理 (可见第 (五) 部分)

(四) 消息处理

在客户端和服务端都涉及到了接收消息的处理,因为在程序中都使用了类似的 switch...case 结构实现,所以在这里进行说明根据协议设计,我们将发送的信息进行了结构化处理,将其打包为了一个结构体 struct INFO

INFO 结构体

```
1 struct INFO
2 {
3     int ID_from; //发送者ID
4     int ID_to; //接收者ID
5     char cmd[10]; //命令
6     SYSTEMTIME systime; //nowtime
7     int len; //发送的消息主体的长度
8     char text[1024]; //消息主体
9     int infoType;
10 }
```

在结构体内我们定义了一个打印信息函数 print_std, 用来将信息按照一定的格式打印到屏幕上 (具体格式为时间 [消息类型] 消息文本), 具体的理论设计我们将在下面的协议设计和核心代码部分进行介绍在 switch...case 结构中, 我们根据消息的类型 infoType, 进行对应处理, 比如

消息处理结构

```
1 case SENDTOALL:
2     //将消息发送至所有用户
3 case SENDTO:
4     //将消息发送至指定用户
5 case EXIT:
6     //退出线程
7     ...
```

三、 协议设计

(一) 时序

在程序中, 我们的时序都是异步的, 具体体现为 send 和 recv 的先后顺序, 在服务端启动后, 启动客户端, 创建连接, 分别由以下的操作

- 用户端 send 用户名
- 服务端主线程接受用户名 (recv), 绑定对应的用户 ID, 创建子线程, 将对应信息返回给用户端 (send)
- 用户端接收线程接收到用户名, 用户 ID, 初始化对应结构体 USERA
- 用户输入文本信息, 根据信息字符串, 选择对应的 INFO 变量处理, 发送 INFO 到服务端
- 服务端对应子线程接收到文本, 根据消息类型匹配对应的 case 处理, 比如调用 sendtoall/sendto/print_std 函数, 打印日志

- 用户端接收线程接收消息，根据消息类型进行对应处理, 打印信息, 返回等待阶段...
- 用户端接收到 exit, 将消息发送到服务器, 退出线程

(二) 语法和语义

1. INFO 结构体

在程序中我们对消息做了一些处理, 定义了一个结构体 INFO(int ID_from; int ID_to; char cmd[10]; SYSTEMTIME systime; int len; //发送的消息主体的长度 char text[1024]; 体 int infoType;), 为了消息类型方便 switch...case 结构体处理, 我们这里预先定义了一些消息类型

宏定义

```
1 #define INIT 0
2 #define SENDTOALL 1
3 #define SENDTO 2
4 #define FORWARD 3
5 #define OK 4
6 #define JOIN 5
7 #define RECV 6
8 #define EXIT 7
```

INFO 的属性具体如下: int ID_from; //发送者 ID

int ID_to; //接收者 ID

char cmd[10]; //命令

SYSTEMTIME systime; //nowtime

int len; //发送的消息主体的长度

char text[1024]; //消息主体

int infoType; 消息类型在结构体中, 我们还定义了一个标准输出函数 print_std, 用来处理打印消息功能

2. 收发消息缓冲区

在程序中, 我们分别在服务端和客户端预先开辟了两个全局字符数组 sendBuf[2048], recvBuf[2048], 用来接收和发送消息

3. 对应的类型转换

因为我们使用了结构体 INFO 来封装消息, 而 send 和 recv 函数对应的参数都是 char * 类型, 所以需要对消息进行处理, 假如我们声明了一个 INFO 结构体 INFO info. 我们主要要做以下处理:

消息结构体转换

```
1 memset(&info, 0, sizeof(msg)); //清空结构体
2 ... //结构体赋值语句
3 memset(recvBuf, 0, 2048); //清空缓存, 对应send为memset(sendBuf, 0, sizeof(
   sendBuf));
4 recv(CilentSocket, recvBuf, sizeof(recvBuf), 0) //接收消息 对应send为send(
   sockCilent, sendBuf, sizeof(sendBuf), 0)
```

```

5 //接收消息转为结构体
6 memset(&info, 0, sizeof(msg)); //清空结构体
7 memcpy(&info, recvBuf, sizeof(msg));
8 ... //其他操作

```

四、 核心代码

在程序中，我们将一些辅助函数和结构体定义到了 init.h 中，在上面的部分，我们已经介绍了一些，下面我们将介绍其他函数和结构

(一) socket 初始化函数

wsainit

```

1 int WSAINIT() {
2     WORD version = MAKEWORD(2, 2);
3     WSADATA wsadata;
4     int error;
5     error = WSAStartup(version, &wsadata);
6     if (error != 0)
7     {
8         switch (error)
9         {
10            case WSASYSNOTREADY:
11                printf("WSASYSNOTREADY");
12                break;
13            case WSAVERNOTSUPPORTED:
14                printf("WSAVERNOTSUPPORTED");
15                break;
16            case WSAEINPROGRESS:
17                printf("WSAEINPROGRESS");
18                break;
19            case WSAEPROCLIM:
20                printf("WSAEPROCLIM");
21                break;
22            }
23            return -1;
24        }
25        return 0;
26    }
27 }

```

1. print_std

print_std

```

1 void print_std() {

```



```

2         printf("\n");
3         printf("%02d:%02d:%02d  ",
4             systime.wHour, systime.wMinute, systime.wSecond);
5         cout << "[ " << InfoType[infoType] << " ]";
6         cout << "    "<<text<<'\n';
7         cout << USERA.username << ">";
8     }

```

2. 发送消息

主要封装了 sendtoall, sendto 函数用来实现群聊和私聊

sendto

```

1 void SendTo(int usrid, INFO info) {
2     //cout << "sendto:" << usrid << endl;
3     SOCKET cilentSock = searchSock(usrid);
4     return SendTo(cilentSock, info);
5 }
6 void SendTo(char * username, INFO info) {
7     SOCKET cilentSock = searchSock(username);
8     return SendTo(cilentSock, info);
9 }

```

sendtoall 函数

sendto

```

1 void SendToAll(INFO info) {
2     //cout << "sendToall build up\n";
3     //cout << "text"<<info.text << endl;
4     //cout << "type"<<info.infoType << endl;
5     vector<user>::iterator iter = usrlist.begin();
6
7     for (; iter != usrlist.end(); iter++) {
8         //cout << iter->usrid << "—id\n";
9         if (iter->usrid != -1) {
10             memset(sendBuf, 0, sizeof(sendBuf)); //清空缓存
11             memcpy(sendBuf, &info, sizeof(info));
12             int ret=send(iter->UserSocket, sendBuf, sizeof(
13                 sendBuf), 0);
14             if (ret == -1) {
15                 perror("SendToAll");
16             }
17             //cout << "status:" << ret << '\n' << endl;
18         }
19     }
20     return;
}

```

其中涉及到了 searchsock 函数, 是用来根据 usrid 和 username 查找对应的 socket

```
sendto

1 SOCKET searchSock(int usrid) {
2     vector<user>::iterator iter = usrlist.begin();
3
4     for (; iter != usrlist.end(); iter++) {
5         //cout << "idlist" << iter->usrid << endl;
6         if (iter->usrid == usrid) {
7             //cout << "find it" << endl;
8             return iter->UserSocket;
9         }
10    }
11    return 0;
12 }
13 SOCKET searchSock(char* username) {
14     vector<user>::iterator iter = usrlist.begin();
15
16     for (; iter != usrlist.end(); iter++) {
17         if (!strcmp(iter->username, username)) {
18             return iter->UserSocket;
19         }
20    }
21    return 0;
22 }
```

五、 附加功能实现

(一) C-S-C 转发模式

主要实现了用户端发送消息到服务端，服务端将消息转发到对应的用户端，具体内容可以参考模块图和时序

(二) 多线程

在服务端，多线程实现对应的用户端的收发消息处理；在客户端，使用两个线程处理收发消息

(三) 群聊

具体实现依赖于 sendtoall 函数，参见核心代码部分

(四) 私聊

具体实现依赖于 sendto 函数，参见核心代码对应部分

(五) 命令

实现 exit 退出和 sendto 私聊命令的识别，依赖于 switch...case 结构

(六) 消息格式设计

消息格式定义为结构体 INFO, 参见上述对应部分描述

六、 实验结果

(一) 开启服务端

```
C:\Users\cultu\source\repos\Server\Debug\Server.exe
Start Server Manager
Start Listening...
```

图 3: C/S 模块流程图

(二) 开启客户端

1. 开启客户端 1

```
C:\Users\cultu\source\repos\Server\Debug\Server.exe
Start Server Manager
Start Listening...
Start Accept request...
23:12:46 [ JOIN ] min Joins,Welcome
Start Accept request...
23:13:08 [ JOIN ] mingm Joins,Welcome
```

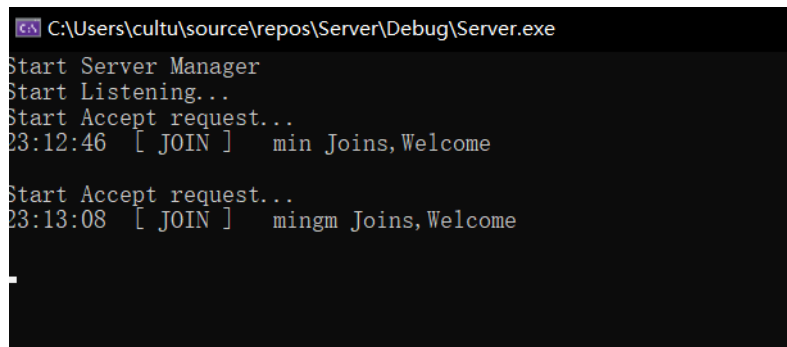
图 4: 开启客户端 1

2. 开启客户端 2

```
C:\Windows\system32\cmd.exe
WSAStartup Complete
Socket Created
Connect to Server ...
Please Enter Your UserName [50 chars LIMIT]:mingm
mingm>
23:13:08 [ JOIN ] mingm Joins,Welcome
mingm>
```

图 5: 开启客户端 2

(三) 服务器日志

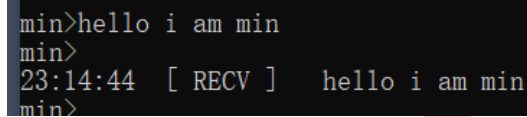


```
C:\Users\cultu\source\repos\Server\Debug\Server.exe
Start Server Manager
Start Listening...
Start Accept request...
23:12:46 [ JOIN ] min Joins,Welcome
Start Accept request...
23:13:08 [ JOIN ] mingm Joins,Welcome
```

图 6: 日志

(四) 聊天室消息发送和接收 (群聊,c-s-c)

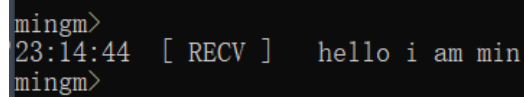
1. 发送



```
min>hello i am min
min>
23:14:44 [ RECV ] hello i am min
min>
```

图 7: 发送

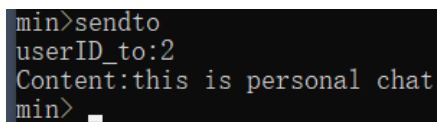
2. 所有用户接受



```
mingm>
23:14:44 [ RECV ] hello i am min
mingm>
```

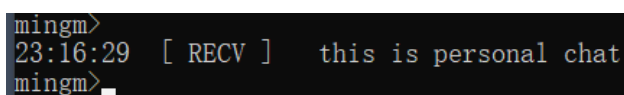
图 8: 接收

(五) 指定用户消息发送和接收 (私聊)



```
min>sendto
userID_to:2
Content:this is personal chat
min> _
```

图 9: 发送



```
mingm>
23:16:29 [ RECV ] this is personal chat
mingm> _
```

图 10: 接收

(六) 退出命令

```
min> exit
exit

C:\Users\cultu\source\repos\Client\Debug\Cilent.exe (进程 31808)已退出，代码为 0。
按任意键关闭此窗口. . .
_
```

图 11: 退出

七、 附录

我们已经将源代码,实验结果,实验报告等上传至 github:<https://github.com/Metetor/network>

NIKU

参考文献

NIKU