

# 计算机网络 第一次实验

姓名：周末

专业：计算机科学与技术

学号：2211349

- 一、实验要求
- 二、协议设计
- 三、各模块功能
- 四、功能展示
- 四、思考总结

## 实验要求

- 给出聊天协议的完整说明
- 利用 C 或 C++ 语言，使用基本的 Socket 函数完成程序。不允许使用CSocket 等封装后的类编写程序
- 使用流式套接字、采用多线程（或多进程）方式完成程序
- 程序应该有基本的对话界面，但可以不是图形界面。程序应该有正常的退出方式
- 完成的程序应该支持多人聊天，支持英文和中文聊天
- 编写的程序应该结构清晰，具有较好的可读性
- 在实验中观察是否有数据丢失，提交可执行文件、程序源码和实验报告。

## 协议设计

### 1. 用户连接协议

- 当客户端连接到服务器时，首先发送用户名。服务器根据用户名确认用户加入，并向其他在线用户广播新用户加入的消息。
- 协议流程：
  - 客户端发送用户名。
  - 服务器接收用户名并记录，广播用户加入消息给其他用户。

## 2. 公共消息广播协议

- 客户端向服务器发送普通消息时，服务器接收到消息后，将该消息广播给所有其他在线用户。
- 协议流程：
  - 客户端发送普通消息。
  - 服务器接收并将消息广播给除发送者以外的所有客户端。

## 3. 私聊协议

- 当用户想私聊其他用户时，使用特定的格式 `@用户名 消息` 发送私聊消息，服务器将该消息转发给目标用户。
- 协议流程：
  - 客户端输入 `@用户名 消息`。
  - 服务器解析消息，找到目标用户，将消息发送给该用户。如果目标用户不在线，服务器告知发送者目标用户不在线。

## 4. 用户列表查询协议

- 用户可以向服务器发送 `list` 命令，服务器会返回当前在线用户列表。
- 协议流程：
  - 客户端发送 `list` 请求。
  - 服务器查询当前在线用户列表，并将结果返回给请求客户端。

## 5. 断开连接协议

- 当客户端断开连接时，服务器会从在线用户列表中删除该用户，并通知其他在线用户此用户已离开。
- 协议流程：
  - 客户端断开连接。
  - 服务器检测到连接中断，从在线用户列表中删除该用户，并广播该用户离开消息。

## 6. 服务器端关闭协议

- 服务器可以通过输入特定指令（如 `exit`）关闭，广播给所有客户端“服务器即将关闭”的消息，并终止所有连接。
- 协议工作：
  - 服务器输入关闭命令。
  - 服务器广播关闭消息，并断开所有客户端连接。

## 7. 时间戳和消息格式

- 每条广播消息或私聊消息附带时间戳，确保消息的发送时间能够记录。
- 协议流程：
  - 服务器在处理消息时，自动添加时间戳并广播或发送。

## 各模块功能

### 1. 服务器主程序 ( main )

- **功能：**服务器端执行逻辑，初始化服务器套接字，启动服务器，等待客户端连接，并通过多线程处理多个客户端。
- **核心代码：**

```
int main() {  
    pthread_mutex_init(&clientsMutex, nullptr); // 初始化互斥锁  
    if (!createServerSocket()) {  
        return -1;  
    }  
    bindAddress();  
    startServer();  
    close(serverSocket);  
    pthread_mutex_destroy(&clientsMutex);  
    return 0;  
}
```

- **主要函数：**
  - createServerSocket()：创建服务器的套接字。
  - bindAddress()：绑定服务器地址和端口。
  - startServer()：启动服务器监听并处理客户端连接。

### 2. 服务器初始化和连接

- **功能：**初始化服务器、绑定地址、开始监听客户端连接。
- **核心代码：**

```

bool createServerSocket() {
    serverSocket = socket(AF_INET, SOCK_STREAM, 0); // 创建套接字
    if (serverSocket == -1) {
        cout << "套接字创建失败" << endl;
        return false;
    }
    return true;
}

void bindAddress() {
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(SERVER_PORT);
    serverAddr.sin_addr.s_addr = INADDR_ANY; // 接受任意IP地址连接
}

void startServer() {
    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == -1) {
        cout << "绑定地址失败" << endl;
        close(serverSocket);
        exit(-1);
    }

    if (listen(serverSocket, BACKLOG) == -1) {
        cout << "监听端口失败" << endl;
        close(serverSocket);
        exit(-1);
    }

    cout << "服务器启动，等待客户端连接..." << endl;

    // 创建监听服务器输入的线程
    pthread_t inputThread;
    pthread_create(&inputThread, nullptr, monitorServerInput, nullptr);
    pthread_detach(inputThread);

    while (serverRunning) { // 检查 serverRunning 状态
        struct sockaddr_in clientAddr;
        socklen_t clientAddrLen = sizeof(clientAddr);
        int clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr, &clientAddrLe

        if (!serverRunning) break; // 检查 serverRunning 状态

        if (clientSocket == -1) {
            cout << "接受客户端连接失败" << endl;
            continue;

```

```

    }

    pthread_t tid;
    pthread_create(&tid, nullptr, handleClient, &clientSocket);
    pthread_detach(tid);
}
}

```

#### • 主要函数：

- `createServerSocket()`：创建服务器套接字。调用 `socket()` 函数，指定 IPv4 协议 (AF\_INET)、面向连接的 TCP 传输方式 (SOCK\_STREAM) 和默认协议。若创建失败，返回错误信息并终止程序。
- `bindAddress()`：配置套接字绑定的地址和端口。将 `serverAddr` 结构体中的 `sin_family` 设置为 AF\_INET，表示使用 IPv4；`sin_port` 设置为服务器端口（例如 8080），并使用 `htons()` 进行网络字节序转换；`sin_addr.s_addr` 设置为 INADDR\_ANY，允许接受任何 IP 地址的连接。
- `startServer()`：监听并接受客户端连接，创建新线程处理每个客户端。通过 `listen()` 开始监听客户端连接请求，设置最大连接队列数。然后调用 `accept()` 阻塞等待客户端连接，接受连接后为每个客户端创建新的线程来处理通信。
- `pthread_create`：创建的新线程允许服务器在主线程外并发执行其他任务。对于聊天服务器，监控服务器输入（如输入退出指令）是一个常见的需求，可以通过一个独立线程来处理，避免影响主线程的功能（如处理客户端连接）。

## 3. 广播消息 ( broadcastMessage )

- **功能**：将消息广播给所有连接的客户端，使用互斥锁确保在遍历客户端列表时不发生竞争条件。
- **核心代码**：

```

void broadcastMessage(const string& message, int senderSocket) {
    pthread_mutex_lock(&clientsMutex); // 加锁保护客户端列表
    for (int i = 0; i < clientSockets.size(); ++i) {
        if (clientSockets[i] != senderSocket) { // 不发送给消息发送者
            send(clientSockets[i], message.c_str(), message.size(), 0);
        }
    }
    pthread_mutex_unlock(&clientsMutex); // 解锁
}

```

#### • 实现：

在 `pthread_mutex_lock()` 加锁的保护下，遍历 `clientSockets` 列表，将消息通过 `send()` 发送给每个在线的客户端，除了消息发送者本身（通过 `senderSocket` 参数进行过滤）。消息内容是经过时间戳处理后的完整字符串。

## 4. 用户列表查询

- **功能：**处理客户端请求在线用户列表，返回当前所有在线的用户名。
- **核心代码：**

```
if (strcmp(buffer, "list") == 0) {
    int userCount = userNames.size();
    string onlineUsers = "在线用户人数: " + to_string(userCount) + "\n";
    for (size_t i = 0; i < userNames.size(); ++i) {
        onlineUsers += to_string(i + 1) + ". " + userNames[i] + "\n";
    }
    send(clientSocket, onlineUsers.c_str(), onlineUsers.size(), 0);
    cout << "[" + getTimeStamp() + "]" + " 用户 " << userName << " 请求用户列表" << endl;
}
```

- **实现：**

服务器在收到 list 命令后，遍历 userNames 列表，构造一个包含所有在线用户名的字符串，并通过 send() 将该字符串发送给请求的客户端。同时，服务器端打印该用户请求在线用户列表的操作日志。

## 5. 私聊消息 ( sendPrivateMessage )

- **主要功能：**
  - 通过目标用户名查找在线用户，并将私聊消息发送到该用户的套接字。（这里使用互斥锁确保用户列表的操作线程安全）
- **核心代码：**

```
void sendPrivateMessage(int clientSocket, const char* targetUser, const char* privateMessage) {
    pthread_mutex_lock(&clientsMutex); // 加锁保护客户端列表
    auto it = find(userNames.begin(), userNames.end(), string(targetUser));
    if (it != userNames.end()) {
        int targetSocket = clientSockets[it - userNames.begin()];
        string privateMsg = "私聊 (" + string(sender) + "): " + privateMessage;
        send(targetSocket, privateMsg.c_str(), privateMsg.size(), 0); // 发送私聊消息
        cout << "私聊消息发送给 " << targetUser << ": " << privateMessage << endl;
    } else {
        cout << "用户 " << targetUser << " 不在线" << endl;
    }
    pthread_mutex_unlock(&clientsMutex); // 解锁
}
```

- **实现：**

- 服务器从消息中解析出目标用户名和私聊内容。如先识别出符号 @，之后在空格之前为目标用户名，空格后为发送的消息。
- 使用 find() 在 usernames 列表中查找目标用户名，如果找到目标用户，则获取对应的套接字并发送消息。

## 6. 客户端处理模块 ( handleClient )

- **功能：**这是客户端整个执行逻辑，为每个连接的客户端处理消息发送、接收以及私聊、公共广播等功能。
- **核心代码：**

```

void* handleClient(void* arg) {
    int clientSocket = *(int*)arg;
    // 接收用户名
    recv(clientSocket, userName, sizeof(userName), 0);
    // 广播加入消息
    string joinMessage = "欢迎" + string(userName) + "加入了聊天! ";
    broadcastMessage("[ " + getTimestamp() + " ] " + joinMessage, clientSocket);
    // 将客户端加入列表
    pthread_mutex_lock(&clientsMutex);
    clientSockets.push_back(clientSocket);
    userNames.push_back(userName);
    pthread_mutex_unlock(&clientsMutex);

    while (true) {
        memset(buffer, 0, BUFFER_SIZE);
        int bytesReceived = recv(clientSocket, buffer, BUFFER_SIZE, 0);
        if (bytesReceived <= 0) break;

        if (strlen(buffer) > 0) {
            // 私聊处理
            if (buffer[0] == '@') {
                // 提取目标用户和消息内容
                sendPrivateMessage(clientSocket, targetUser.c_str(), privateMessage.c_str())
            }
            // 列表请求
            else if (strcmp(buffer, "list") == 0) {
                // 返回在线用户列表
                send(clientSocket, onlineUsers.c_str(), onlineUsers.size(), 0);
            }
            // 公共广播消息
            else {
                broadcastMessage(string(userName) + ": " + buffer, clientSocket);
            }
        }
    }
    // 处理客户端断开连接
}

```

#### • 主要函数：

- recv()：接收客户端消息。
- send()：发送消息到客户端。
- sendPrivateMessage()：处理私聊消息。



- `broadcastMessage()` : 广播消息到所有客户端。

## 7. 设置时间戳模块 ( `getTimeStamp` )

- **主要功能:**

获取当前系统时间并格式化为 `YYYY-MM-DD HH:MM:SS` , 为每条广播或私聊消息附带发送时间。

- **核心代码:**

```
string getTimeStamp() {  
    time_t now = time(0);  
    tm* localTime = localtime(&now);  
    char buffer[80];  
    strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", localTime);  
    return string(buffer);  
}
```

- **实现:**

通过 `time()` 获取系统当前时间, 再通过 `localtime()` 将时间转换为本地时间格式。使用 `strftime()` 函数将时间格式化为 `YYYY-MM-DD HH:MM:SS` 的字符串, 并返回这个字符串作为时间戳。

## 8. 用户退出模块

- **功能:** 监听用户的退出命令, 当用户输入 `exit` 时, 服务器断开用户的连接。

- **核心代码:**

```
// 用户输入  
void handleQuit() {  
    char quitMsg[CBUF_SIZE] = "quit";  
    send(LocalhostSocket, quitMsg, strlen(quitMsg), 0);  
    close(LocalhostSocket);  
    cout << "退出聊天, 关闭连接" << endl;  
}
```

```
// 服务器处理, 关闭用户连接, 并清理  
pthread_mutex_lock(&clientsMutex);  
clientSockets.erase(remove(clientSockets.begin(), clientSockets.end(), clientSocket), clientSocket);  
userNames.erase(remove(userNames.begin(), userNames.end(), string(userName)), userName);  
pthread_mutex_unlock(&clientsMutex);
```

**实现:**

- 在每个客户端线程中，增加对用户输入的监控。如果用户输入 `exit` 命令，客户端将向服务器发送退出请求，并断开与服务器的连接。
- 服务器在接收到用户退出请求后，会广播一条消息，通知其他在线用户该用户已退出，随后关闭与该用户的套接字连接。

## 9. 服务器关闭模块

- **功能：**在服务器端通过输入特定命令关闭服务器，并通知所有客户端服务器即将关闭。
- **核心代码：**

```
void sendDummyConnection() {
    int dummySocket = socket(AF_INET, SOCK_STREAM, 0);
    if (dummySocket == -1) {
        cout << "虚拟客户端创建失败" << endl;
        return;
    }

    struct sockaddr_in dummyAddr;
    dummyAddr.sin_family = AF_INET;
    dummyAddr.sin_port = htons(SERVER_PORT);
    dummyAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(dummySocket, (struct sockaddr*)&dummyAddr, sizeof(dummyAddr));
    close(dummySocket); // 立即关闭连接
}

void* monitorServerInput(void* arg) {
    string input;
    while (true) {
        getline(cin, input);
        if (input == "exit") {
            serverRunning = false;
            close(serverSocket);
            sendDummyConnection();
            cout << "服务器关闭" << endl;
            break;
        } else{
            broadcastMessage( "[" + getTimeStamp() + "]" + "[系统消息]" + input, -1);
        }
    }
    return nullptr;
}
```

- **实现：**

服务器端在主线程中开启一个独立的线程，持续监听管理员输入。如果管理员在服务器端输入 `exit` 命令，服务器将广播一条“服务器断开连接，服务器关闭”的消息给所有在线用户，随后关闭服务器套接字并终止程序。

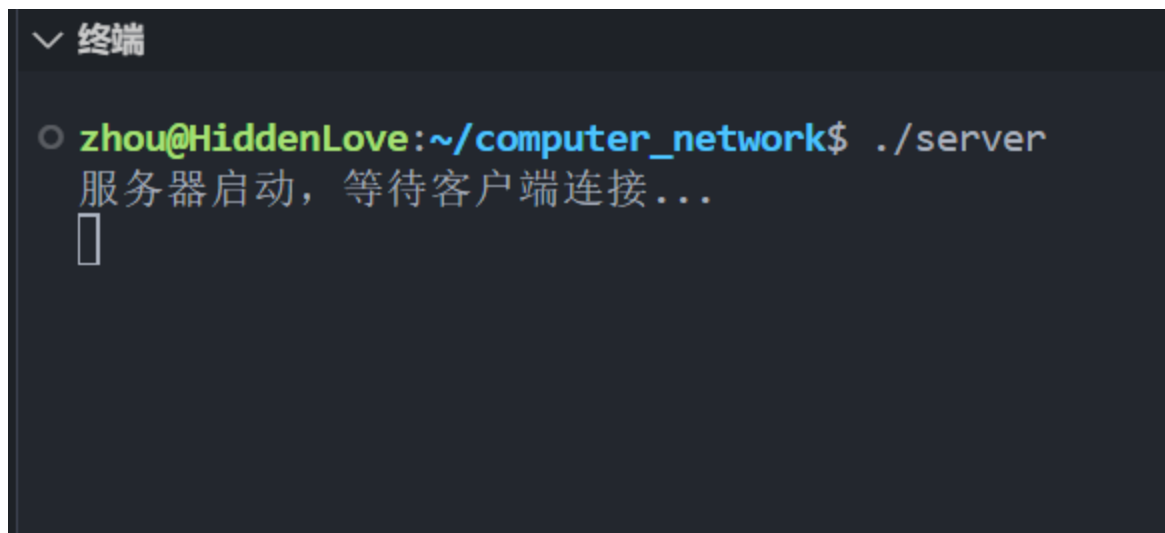
如果当前没有客户端连接的话，可能会导致错误，所以创建了一个短暂的虚拟客户端，进行连接，然后将安全的将服务器关闭。

## 功能展示

### 1. 服务器初始化，等待连接

在代码中默认规定服务器端口号和ip：

- `u_short ClientPort = 12870;`
- `const char* LocalIP = "127.0.0.1";`

A terminal window titled '终端' (Terminal) with a dropdown arrow. It shows a user prompt 'zhou@HiddenLove:~/computer\_network\$' followed by the command './server'. Below the command, the text '服务器启动，等待客户端连接...' (Server started, waiting for client connection...) is displayed, followed by a cursor icon.

```
终端
zhou@HiddenLove:~/computer_network$ ./server
服务器启动，等待客户端连接...
█
```

### 2. 客户端连接

使用与服务器相同的的 IP 地址 和 端口号发起连接，连接服务器成功后，客户端接受相应提示，并进行用户名输入进入聊天。服务器接受用户连接消息。

```
调试控制台 终端 端口 1
> 终端
o zhou@HiddenLove:~/computer_network$ ./server
服务器启动, 等待客户端连接...
[2024-10-18 17:52:46] 用户 aa 已经连接到服务器
[ ]

o zhou@HiddenLove:~/computer_network$ ./client
成功连接, 欢迎加入聊天!
请输入你的聊天用户名 (不要含有空格): aa
@===== 聊天室 =====@
[ ]
```

### 3. 用户正常消息发送

可接收多人加入,新客户端连接加入时,对所有用户广播消息提醒。消息发送后, 服务器对所有人进行广播, 在他人客户端聊天界面上显示有 用户名 : 消息 , 支持中英文输入。同时在服务器上进行输入后, 客户端在聊天界面上会接受到类型为系统消息的消息。

```
调试控制台 终端 端口 1
> 终端
o zhou@HiddenLove:~/computer_network$ ./server
服务器启动, 等待客户端连接...
[2024-10-18 18:31:08] 用户 aa 已经连接到服务器
客户端断开连接:
exit
服务器关闭
o zhou@HiddenLove:~/computer_network$ ./server
服务器启动, 等待客户端连接...
[2024-10-18 18:32:41] 用户 aa 已经连接到服务器
[2024-10-18 18:32:49] 用户 bb 已经连接到服务器
[2024-10-18 18:33:58] (aa): hello
[2024-10-18 18:34:05] (bb): 你好
大家好, 欢迎加入聊天
[ ]

o zhou@HiddenLove:~/computer_network$ ./client
成功连接, 欢迎加入聊天!
请输入你的聊天用户名 (不要含有空格): aa
@===== 聊天室 =====@
[2024-10-18 18:32:49] 欢迎bb加入了聊天
hello
bb: 你好
[2024-10-18 18:34:30] [系统消息] 大家好, 欢迎加入聊天
[ ]

o zhou@HiddenLove:~/computer_network$ ./client
成功连接, 欢迎加入聊天!
请输入你的聊天用户名 (不要含有空格): bb
@===== 聊天室 =====@
aa: hello
你好
[2024-10-18 18:34:30] [系统消息] 大家好, 欢迎加入聊天
[ ]
```

### 3. 用户发送指令消息

- 获取当前在线用户列表: list

```

○ zhou@HiddenLove:~/computer_network$ ./client
成功连接,欢迎加入聊天!
请输入你的聊天用户名(不要含有空格): aa
@===== 聊天室 =====@
[2024-10-18 17:56:01] 欢迎bb加入了聊天
hello
bb: 你好
[2024-10-18 18:18:41] [系统消息] 好,欢迎大家
[2024-10-18 18:18:56] [系统消息] 大家好,欢迎大家
list
在线用户人数: 2
1. aa
2. bb

```

- 退出聊天: quit。断开与服务器连接。其他人也会接受到该用户退出聊天消息。

<pre> ○ zhou@HiddenLove:~/computer_network\$ ./client 成功连接,欢迎加入聊天! 请输入你的聊天用户名(不要含有空格): aa @===== 聊天室 =====@ [2024-10-18 18:32:49] 欢迎bb加入了聊天 hello bb: 你好 [2024-10-18 18:34:30] [系统消息] 大家好,欢迎加入聊天 [2024-10-18 18:35:57] bb 离开了聊天 </pre>	<pre> ● zhou@HiddenLove:~/computer_network\$ ./client 成功连接,欢迎加入聊天! 请输入你的聊天用户名(不要含有空格): bb @===== 聊天室 =====@ aa: hello 你好 [2024-10-18 18:34:30] [系统消息] 大家好,欢迎加入聊天 quit 退出聊天,关闭连接 ○ zhou@HiddenLove:~/computer_network\$ </pre>
--	---

## 4. 系统日志

- 记录用户连接退出情况
- 记录用户消息内容
- 记录用户发送的指令,如申请用户列表,退出指令等
- 显示 [时间戳] [用户名] [信息]

## ▼ 终端

- **zhou@HiddenLove:~/computer\_network\$ ./server**  
服务器启动，等待客户端连接...  
[2024-10-18 18:31:08] 用户 已经连接到服务器  
客户端断开连接：  
exit  
服务器关闭
- **zhou@HiddenLove:~/computer\_network\$ ./server**  
服务器启动，等待客户端连接...  
[2024-10-18 18:32:41] 用户 aa 已经连接到服务器  
[2024-10-18 18:32:49] 用户 bb 已经连接到服务器  
[2024-10-18 18:33:58] (aa): hello  
[2024-10-18 18:34:05] (bb): 你好  
大家好，欢迎加入聊天  
[2024-10-18 18:35:57] 用户 bb 退出  
[2024-10-18 18:39:35] 用户 cc 已经连接到服务器  
[2024-10-18 18:39:41] 用户 cc 请求用户列表  
□

## 思考总结

### 遇到问题

- 客户端发送消息时丢失数据

客户端发送消息后，服务器端接收到的数据不完整，或者消息被截断。

解决：消息的大小超过了接收缓冲区的大小。使用 `recv()` 函数时，未处理好分段接收问题（TCP 是面向流的协议）。

在接收消息时，循环调用 `recv()` 直到完整接收到所有数据。确保发送消息时，分块发送较大数据。

- 多线程并发问题

在多线程环境中，多个客户端同时操作共享资源（如用户列表、消息队列）时，可能出现竞争条件，导致数据不一致或程序崩溃。

解决：缺乏对共享资源的适当保护，未使用互斥锁来防止竞争条件。

在访问共享资源时，使用 `pthread_mutex` 保护临界区，确保同一时刻只有一个线程可以访问和修改共享数据。

- 私聊功能中，客户端发送私聊消息后，服务器找不到目标用户，即使用户名是正确的。

## 心得

通过这个实验，深入理解了网络通信的基本原理、多线程并发编程以及如何设计一个稳定的、用户友好的客户端-服务器应用程序。网络编程中最大的挑战是处理并发、数据传输的完整性以及客户端与服务器的同步。完成一个完整的程序，通过实现一个个模块，并在此基础上通过不断优化代码结构和协议设计，实现功能更丰富的聊天应用程序。