
南开大学
Nankai University

# 南　　开　　大　　学

## 计　算　机　学　院

**计算机网络实验报告**

---

# 利用 Socket 编写聊天程序

---

汤清云 2013536

年级：2020 级

专业：计算机科学与技术

指导教师：张建忠　徐敬东

2022 年 10 月 22 日

# 摘要

啊啊啊啊啊


**关键词：**


# 目录

# 一、 实验目的

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。完整地说明交互消息的类型、语法、语义、时序等具体的消息处理方式。

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，但需要给出使用方法。编写程序时，只能使用基本的 Socket 函数，不允许使用对 socket 封装后的类或架构。

4）对实现的程序进行测试。

# 二、 实验过程

## (一) 程序设计流程

### 1. 服务器端流程

对于**基础要求**的实现**双人**聊天功能（只能一**来一回**的交互式发送和接收信息），服务器端设计流程如下:

1：加载套接字库，创建套接字 (WSAStartup()/socket())；

2：绑定套接字到一个 IP 地址和一个端口上 (bind())；

3：将套接字设置为监听模式等待连接请求 (listen())；

4：请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字 (accept())；

5：用返回的套接字和客户端进行通信 (send()/recv())；

6：返回，等待另一连接请求；

7：关闭套接字，关闭加载的套接字库 (closesocket()/WSACleanup())。

对于**中级要求**的实现**双人**聊天功能（双方能够各自发送或接收**多条消息**，不拘泥于一来一回），服务器端设计流程如下:

1：加载套接字库，创建套接字 (WSAStartup()/socket())；

2：绑定套接字到一个 IP 地址和一个端口上 (bind())；

3：将套接字设置为监听模式等待连接请求 (listen())；

4：请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字 (accept() )；

5. **创建一个专门用来接收来自于客户端消息的线程，而主线程只负责发送消息。**

6：返回，等待另一连接请求；

7：关闭套接字，关闭加载的套接字库

对于**高级要求**的实现**多人**聊天功能（多个客户能够各自发送或接收**多条消息**，不拘泥于一来一回），服务器端设计流程如下:

1：加载套接字库，创建套接字 (WSAStartup()/socket())；

2：绑定套接字到一个 IP 地址和一个端口上 (bind())；

3：将套接字设置为监听模式等待连接请求 (listen())；

4：**主线程用于接收请求 (accept())，当到来的请求没有超过当前设定的监听人数上限时，为每一个客户端分配一个线程。**

5. **处理客户端线程中额外开辟接收消息处理线程。**

6. **接收消息处理线程把收到的来自于某个客户端的消息进行消息的接收和分发。**

7：关闭套接字，关闭加载的套接字库

### 2. 客户端流程

对于**基础要求**的实现**双人**聊天功能（只能**一来一回**的交互式发送和接收信息），客户端设计流程如下：

1：加载套接字库，创建套接字 (WSAStartup()/socket())；

2：向服务器发出连接请求 (connect())；

3：和服务器端进行通信 (send()/recv())；

4：关闭套接字，关闭加载的套接字库 (closesocket()/WSACleanup())。

而对于**中级要求**和**高级要求**的客户端实现，只需实现对收发消息的双线程处理，因此两者的设计流程如下：

1：加载套接字库，创建套接字 (WSAStartup()/socket())；

2：向服务器发出连接请求 (connect())；

3：**创建一个线程用来接收来自于服务器端的消息，主线程只用来发送消息。**

4：关闭套接字，关闭加载的套接字库 (closesocket()/WSACleanup())。

## (二) 实验模块介绍

### 1. 服务器实验模块介绍

1. 加载套接字库

wVersionRequested 是 Windows Sockets API 提供的调用方可使用的最高版本号。flag 表示是否加载套接字库成功。

```
WSADATA wsaData;
    WORD mVersionRequested = MAKEWORD(2, 2);
    int flag = WSAStartup(mVersionRequested, &wsaData);
    time_t now = time(nullptr);
    char* curr_time = ctime(&now);
    if (flag == 0)
    {
        cout << curr_time << "套接字库加载成功" << endl;
    }
    else
    {
        cout << curr_time << "套接字库加载失败" << endl;
    }
```

2. 创建 socket 并绑定端口号与地址，指定地址类型为 AF_INET，流式套接字，TCP 协议。

```
SOCKET server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    //为之后绑定套接字做准备
    SOCKADDR_IN ServerAddr;
    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_addr.s_addr = inet_addr("127.0.0.1");//IP地址
    ServerAddr.sin_port = htons(6666);//端口号
    // 绑定套接字到一个IP地址和一个端口上
    flag = bind(server, (SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
    now = time(nullptr);
    curr_time = ctime(&now);
```

```
        if (flag == 0)//判断绑定
        {
                cout << curr_time << "绑定成功" << endl;
        }
        else
        {
                cout << curr_time << "绑定失败" << endl;
        }
```

3. 设置监听模式, 同一个队列中最多十个请求

```
    flag = listen(server, 10);
```

4. 在主线程中写一个 while 循环，保持其不会结束。设置一个 accepter 来对应此次连接，当当前客户端人数不超过监听人数上限时，给其创建一个线程，以此实现多个客户端的聊天功能

```
    while (1)
        {
                SOCKET accepter = accept(server, (SOCKADDR*)&ServerAddr, &len
                    );
                //创建线程用来负责多客户端
                if (curr_clients<10)//不能多于监听数目上限
                {
                        clientsarray[curr_clients] = accepter;//记录不同客户
                            端
                        HANDLE hThread = CreateThread(NULL, 0, ClientThread,
                            (LPVOID)&accepter, 0, 0);
                        CloseHandle(hThread);//关闭对线程的引用

                }
        }
```

5. 对于每个客户端线程，记录当前客户端人数、开辟转收消息线程

```
    SOCKET accepter = *(SOCKET*)IpParameter;//获取服务器端的SOCKET参数
        //再开个线程收数据
        HANDLE hThread = CreateThread(NULL, 0, servEventThread, (LPVOID)&
            accepter, 0, 0);
        CloseHandle(hThread);
        //用返回的套接字和客户端进行通信(send()/recv());
        cout << "----------聊天开始----------" << endl;
        curr_clients += 1;
        cout << "当前客户端人数为: " << curr_clients << endl;
```

6. 在每个收发消息线程中，判断消息来自于哪个客户端，**将所收到的消息进行打包，使其加上消息头【时间】以及【来自于某个客户端】**，之后再将此消息转发到其余客户端。

```
    while (1)
        {
                int recvlength = recv(accepter, recvtext, 500, 0);
                for (int i = 0;i < curr_clients;i++)//判断来自于哪个客户端
```

```
5                 {
6                         if (accepter == clientsarray[i])
7                                 from_client = i;
8                 }
9                 sendtext = curr_time;//封装时间
10                sendtext.append("客户端");
11                sendtext.append(to_string(from_client + 1));
12                sendtext.append("发来消息:\"");
13                string temp=recvtext;
14                sendtext.append(temp);
15                sendtext.append("\"");//将消息封装打包为时间+客户端i发来消息
                      “__”
16                //之后进行send()
17                }
```

**2. 客户端实验模块介绍**

1. 加载套接字库, 代码略。

2. 创建 socket 并绑定端口号与地址，指定地址类型为 AF_INET，流式套接字，TCP 协议。

3. 向服务器发送连接请求。

```
1     connect(client, (SOCKADDR*)&ClientAddr, sizeof(SOCKADDR));
```

4. 当连接请求被准许后，开创接收消息的线程，而主线程用于发送消息给服务器。当本客户端的消息发送成功后，在客户端打印成功信息，如果输入 quit() 则说明退出聊天。

```
1     while (1)
2         {
3                 cin.getline(clienttext, 499);
4                 if (!(strcmp(clienttext, "quit()")))//退出聊天信息
5                 {
6                         cout << "您已经退出聊天" << endl;
7                         break;
8                 }
9                 send(client, clienttext, sizeof(clienttext), 0);
10                now = time(nullptr);
11                curr_time = ctime(&now);
12                cout << curr_time << "信息\""<<clienttext<<"\"发送成功" <<
                      endl;
13                memset(clienttext, 0, sizeof(clienttext));
14        }
```

5. 接收消息线程对所收到的消息进行判断，若收不到任何消息则说明服务器已关闭，聊天结束，否则则将所收到的消息进行打印。

```
1     while (1)
2         {
3                 int recvlength=recv(client, servertext, sizeof(servertext),
                      0);//sizeof(servertext)
4                 printtext = servertext;
```

```
5                   if (无消息)
6                   {}
7                   else 收到消息
8                   {
9                           打印消息
10                  }
11                  清空消息缓冲区
12          }
```

### 3. 模块流程图

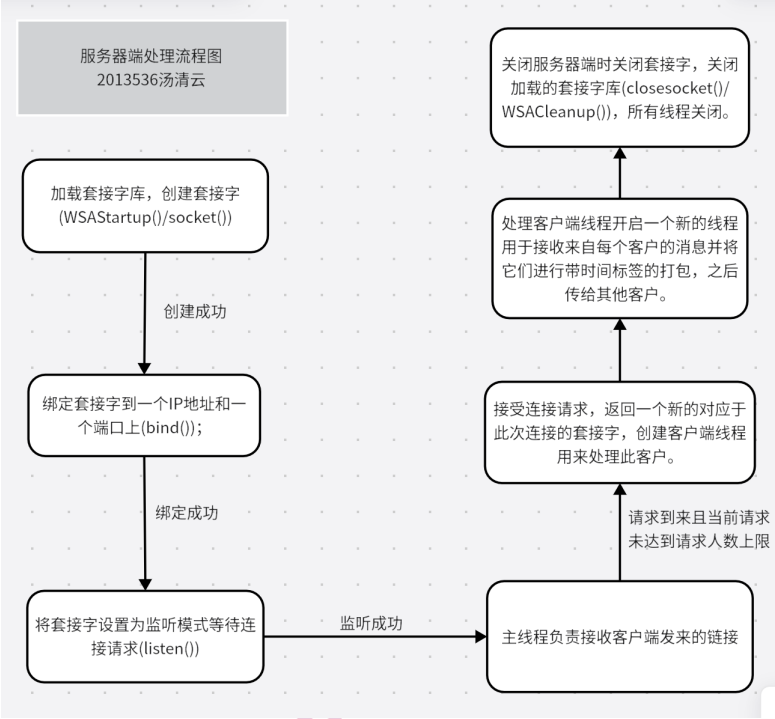


图 1: 服务器端流程图

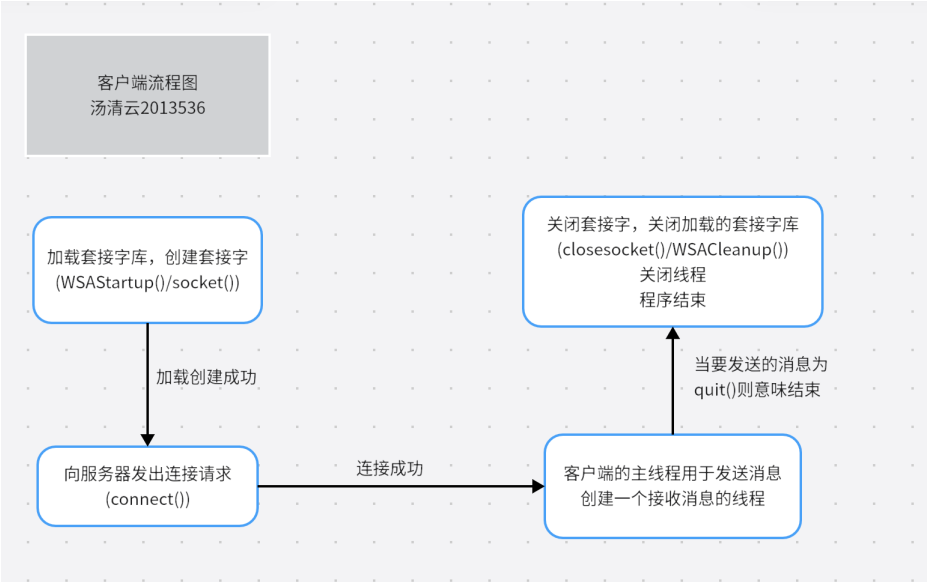


图 2: 客户端流程图

#### 4. 消息设计

**服务器端消息类型：**

所收到的消息只包含客户端 x 所发的具体内容 text。服务器在分发此消息之前对接收消息时间进行取值并判断当前消息来自于哪个客户端。之后对原消息 text 进行打包，变成**服务器端收到客户端 x 消息的时间 +“来自于客户端 x”+ 消息内容 text**。

**客户端消息类型：**

所收到的消息为服务器分发来的其他客户端的消息，类型为**服务器端收到客户端 x 消息的时间 +“来自于客户端 x”+ 消息内容 text**，而发送的消息只有**具体输入内容**。

**语法、语义、时序**

客户端 i 发送消息给服务器；服务器再将此消息进行打包，按与服务器建立连接的时序依次发送给其他客户端。语法语义无特殊定义。

## (三) 实现效果图

1. 开启服务器

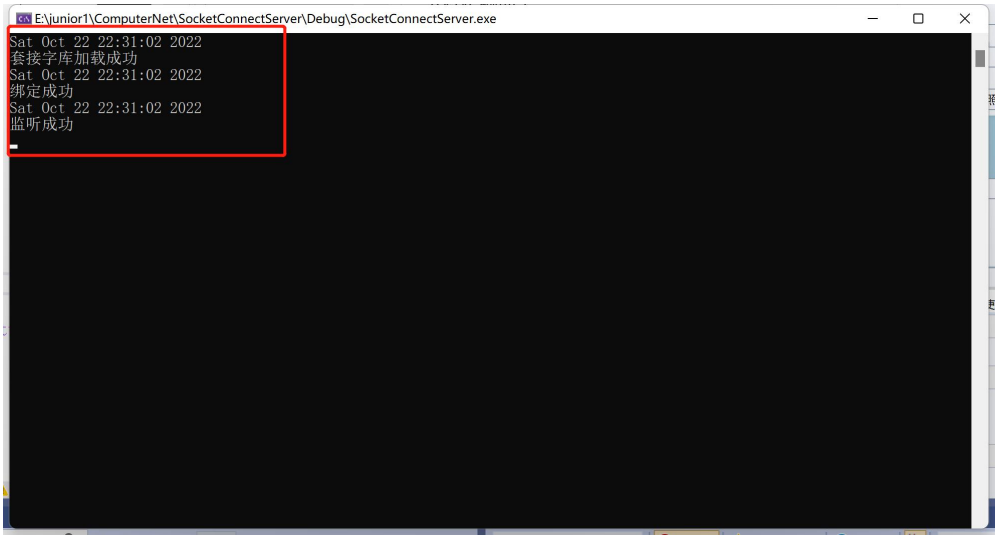


图 3: 开启服务器效果图

2. 开启多个客户端

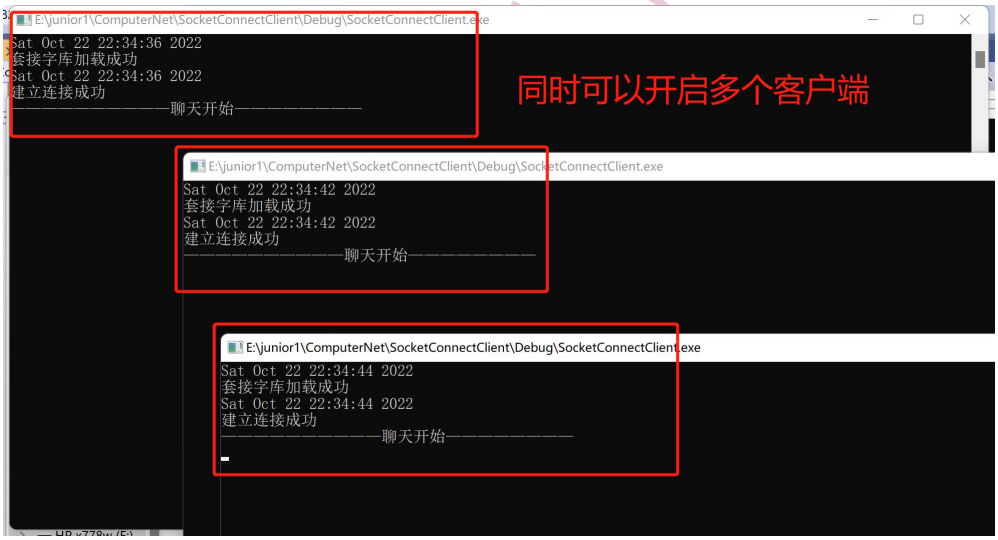


图 4: 多个客户端同时在线效果图

3. 服务器端显示当前在线人数

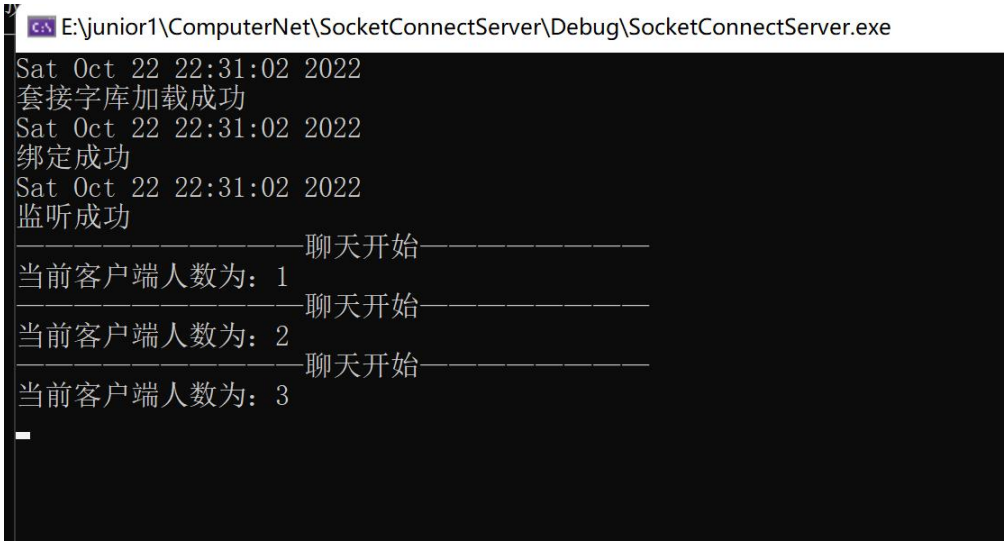


图 5: 服务器端显示

4. 在一个客户端发送消息后，其他客户端接收打包后的消息


图 6: 客户端接收

5. 服务器后台显示消息的接收与处理转发过程:


图 7: 服务器后台显示

6. 当一个客户端输入 quit() 退出时，服务器修改在线人数。


图 8: 客户端流程图

# 三、 总结

在此次实验中，我学习了流式套接字的使用，初步完成了一个简陋版多人聊天室的建立；对其中牵扯到的各个函数有了更加深刻的了解和认识。