南開大學
Nankai University

计算机学院

计算机网络实验报告

利用 Socket 设计并编写一个聊天程序

姓名：杨馨仪

学号：2011440

专业：计算机科学与技术

2022 年 10 月 22 日

目录

1 实验要求

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。并且在 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。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。

2 协议设计

服务器端主要实现四个功能：

- **功能一：**将“公告”广播到每个客户端
- **功能二：**接受某个客户端连接的请求，并将该客户端加入的信息发送给其他客户端
- **功能三：**接收某个客户端断开连接的请求，并将该客户端退出的消息发送给其他客户端
- **功能四：**将某个客户端发送来的消息转发给其他客户端

对于功能一，服务器接受服务器端输入的一行字符串（以换行符为结尾），并将这行字符串与“[公告]”和时间戳连接在一起。循环发送给每一个客户端。

对于功能二、三、四，服务器循环遍历每一个终端（包括服务器端与客户端），对每一个终端，查看是否发生事件，等待 WAIT_TIME 毫秒，如果此时有事件发生在该终端，则对事件进行进一步的处理，如果超时或返回的索引非法则跳过此终端，访问下一个终端。

客户端输入正确的服务器 IP 地址及端口号后，会向服务器端发出 connect 请求，当服务器端发现了 accept 消息时，服务器首先查看现在的连接数量是否达到上限，如果未达到上限，则调用 accept 函数与该客户端建立连接。然后为新的客户端 socket 创建事件对象，接着将欢迎词（一行字符串）广播到所有的客户端。

客户端输入“exit”字符串后，将断开与服务器端的连接，当服务器端发现了 close 消息时，则在服务器端的命令行打印出退出聊天的消息，并将这行字符串广播到其他的客户端，代表某用户退出了群聊。

当某个客户端在成功与服务器端建立连接后，在命令行中输入一行字符串并按下回车后，客户端会将这行字符串 send 到服务器端。当服务器端接收到 read 消息时，会遍历每一个客户端，发现是哪个客户端发送来的消息，并且把这条消息发送给其他的所有客户端。

功能一与功能二、三、四位于不同的两个线程，二者并行执行。对于功能二、三、四，会循环多次遍历每一个终端，如果在 10ms 之内，事件队列中有事件发生，将处理排序在第一位的事件，然后轮换到下一个终端，如果 10ms 之内，该终端没有事件发生，则判定为超时，直接轮换到下一个终端。

该聊天程序实现了多线程及群聊等拓展功能，具体代码及解析在模块分析中呈现。

3 模块功能介绍

3.1 模块划分说明

3.1.1 服务器端

在服务器端有两个线程, 其中一个线程用来处理 socket 的创建（socket 模块），socket 与地址的绑定（bind）模块，监听（listen 模块），服务器信息的广播（send 模块）以及关闭服务器（clean 模块）

等功能。除此之外，还设有时间戳函数模块，调用 timei() 函数即可返回 “%Y-%m-%d %H:%M:%S” 形式的时间戳字符串。

另一个线程总体上分为三个模块，分别用来处理客户端的连接（accept 模块），客户端的关闭（close 模块），以及接受客户端消息并转发给其他客户端（recv 模块）。

3.1.2 客户端

在客户端有两个线程，其中一个线程用来处理 socket 的创建（socket 模块），与服务器连接（connect 模块），向服务器发送消息（send 模块），以及关闭服务器（clean 模块）。除此之外，还设有时间戳函数模块，调用 timei() 函数即可返回 “%Y-%m-%d %H:%M:%S” 形式的时间戳字符串。

另一个线程用来接收来自服务器的消息并打印（recv 模块）。

3.2 模块功能说明

3.2.1 服务器端

时间戳 timei 函数

```
char* timei()
{
  time_t t;
    struct tm *tmp;

  time(&t);
    tmp = localtime(&t);
  strftime(buf2, 64, "%Y-%m-%d %H:%M:%S", tmp);
  return buf2;
}
```

调用该函数即可返回 “%Y-%m-%d %H:%M:%S” 格式的时间戳字符串。

socket 模块

```
printf("%s [ INFO ] Start Server Manager\n",timei());

WSADATA wsaData; //可用socket详细信息
WSAStartup(MAKEWORD(2, 2), &wsaData);
printf("%s [ OK   ] WSAStartup Complete!\n",timei());

printf("%s [ INFO ] Local Machine IP Address: 127.0.0.1\n",timei());

SOCKET sockSrv = socket(AF_INET, SOCK_STREAM, 0);
if(sockSrv !=INVALID_SOCKET)
  printf("%s [ OK   ] Socket Created!\n",timei());
```

此模块发生在主线程中，首先打印出 “Start Server Manager” 代表服务器端开始工作，然后初始化 Socket DLL，协商使用的 Socket 版本，完成后打印 “WSAStartup Complete!”，接着打印出服务器端的 IP 地址等信息。最后创建一个 socket，并绑定到 TCP 传输层服务，如果创建成功则打印 “Socket Created !”。

bind 模块

```
SOCKADDR_IN addrSrv = { 0 };
addrSrv.sin_family = AF_INET;//用AF_INET表示TCP/IP协议。
addrSrv.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");//设置为本地回环地址
addrSrv.sin_port = htons(819711);

if(bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR))!= SOCKET_ERROR)
  printf("%s [ OK   ] Bind Success!\n",timei());
```

此模块发生在主线程中，首先对服务器端的地址信息（包括 IP 地址与端口号）进行初始化，然后使用 bind 函数将本地地址绑定到服务器 socket 上，如果绑定成功则打印 “Bind Success !”。

listen 模块

```
WSAEVENT servEvent = WSACreateEvent();
WSAEventSelect(sockSrv, servEvent, FD_ALL_EVENTS);

cliSock[0] = sockSrv;
cliEvent[0] = servEvent;

CloseHandle(CreateThread(NULL, 0, handlerRequest, (LPVOID)&sockSrv, 0, 0));
printf("%s [ INFO ] Broadcast thread create success!\n",timei());

listen(sockSrv, 5);
printf("%s [ INFO ] Start listening...\n",timei());
```

此线程发生在主线程中，主要完成服务器事件对象的创建，并且绑定事件对象用于监听所有事件。然后创建新的线程，并打印 “Broadcast thread create success!” 。最后调用 listen() 函数，使服务器 socket 进入监听状态，监听远程链接是否到来，并打印 “Start listening...”。

send 模块

```
while (1) {
  char contentBuf[BUFFER_SIZE] = { 0 };
  char sendBuf[BUFFER_SIZE] = { 0 };
  cin.getline(contentBuf, sizeof(contentBuf));
  sprintf(sendBuf, "[ 公告 ]%s", contentBuf);
  for (int j = 1; j <= total; j++)
  {
    send(cliSock[j], sendBuf, sizeof(sendBuf), 0);
  }
}
```

此线程发生在主线程中，主要完成获取服务器端输入的信息，并将信息广播到其他所有的客户端。实际情况类似于游戏中服务器向所有玩家发出公告。

clean 模块

```
closesocket(sockSrv);

WSACleanup();
```

此模块位于主线程中，用于关闭服务器 socket，并且释放 socket DLL 资源。

接下来的一些模块位于另一线程中，在该线程中，循环每一个终端（包括服务器端与客户端），对每一个终端，查看是否发生事件，等待 WAIT_TIME 毫秒，如果此时有事件发生在该终端，则对事件进行进一步的处理，如果超时或返回的索引非法则跳过此终端，访问下一个终端。

accept 模块

```
if (networkEvents.lNetworkEvents & FD_ACCEPT)
{
  if (networkEvents.iErrorCode[FD_ACCEPT_BIT] != 0)
  {
    cout << "连接时产生错误，错误代码" << networkEvents.iErrorCode[FD_ACCEPT_BIT] <<
        endl;
    continue;
  }
  if (total + 1 <= MAX)
  {
    int nextIndex = total + 1;
    int addrLen = sizeof(SOCKADDR);
    SOCKET newSock = accept(sockSrv, (SOCKADDR*)&cliAddr[nextIndex], &addrLen);
    if (newSock !=, INVALID_SOCKET)
    {
      printf("%s [ OK   ] Accept Success!\n",timei());
      cliSock[nextIndex] = newSock;

      WSAEVENT newEvent = WSACreateEvent();
      WSAEventSelect(cliSock[nextIndex], newEvent, FD_CLOSE | FD_READ | FD_WRITE);
      cliEvent[nextIndex] = newEvent;

      total++;//客户端连接数增加

      printf("%s [ JOIN ] user#%d just join,
          welcome!\n",timei(),nextIndex,inet_ntoa(cliAddr[nextIndex].sin_addr));
      //inet_ntoa() 将网络地址转换成“.”点隔的字符串格式

      char buf[BUFFER_SIZE];
      sprintf(buf,"%s [ JOIN ] welcome user#%d enter the chat
          room",timei(),nextIndex);

      for (int j = i; j <= total; j++)
      {
        send(cliSock[j], buf, sizeof(buf), 0);
      }
    }
  }
}
```

该模块用于处理 accept 事件的发生。如果此时服务器端监听到有客户端向服务器发出连接请求，服务器首先查看现在的连接数量是否达到上限，如果未达到上限，则调用 accept 函数与该客户端建立

连接，并打印 “Accept Success!”。然后为新的客户端 socket 创建事件对象，并在服务器端的命令行打印出 “[JOIN] user#%d just join, welcome!”，接着将欢迎词广播到所有的客户端，实际情景类似于“某人加入了群聊。

close 模块

```
else if (networkEvents.lNetworkEvents & FD_CLOSE)//客户端被关闭，即断开连接
{
  //i表示已关闭的客户端下标
  total--;
  printf("%s [ EXIT ] user#%d just exit the chat
      room\n",timei(),i,inet_ntoa(cliAddr[i].sin_addr));
  //释放这个客户端的资源
  closesocket(cliSock[i]);
  WSACloseEvent(cliEvent[i]);

  for (int j = i; j < total; j++)
  {
    cliSock[j] = cliSock[j + 1];
    cliEvent[j] = cliEvent[j + 1];
    cliAddr[j] = cliAddr[j + 1];
  }

  char buf[BUFFER_SIZE];
  sprintf(buf,"%s [ EXIT ] user#%d just exit the chat room\n",timei(),i);

  for (int j = 1; j <= total; j++)
  {
    send(cliSock[j], buf, sizeof(buf), 0);
  }
}
```

该模块用于处理客户端关闭事件的发生。如果此时某个客户端关闭，则在服务器端的命令行打印出 “[EXIT] user #%d just exit the chat room”，并将这个字符串广播到其他的客户端，代表某用户退出了群聊。当让对于一些存储客户端 socket，地址，事件对象的数组都要进行调整。

recv 模块

```
else if (networkEvents.lNetworkEvents & FD_READ)//接收到消息
{

  char buffer[BUFFER_SIZE] = { 0 };//字符缓冲区，用于接收和发送消息
  char buffer2[BUFFER_SIZE] = { 0 };

  for (int j = 1; j <= total; j++)
  {
    int nrecv = recv(cliSock[j], buffer, sizeof(buffer), 0);//nrecv是接收到的字节数
    if (nrecv > 0)//如果接收到的字符数大于0
    {
      sprintf(buffer2,"%s [ RECV ] user#%d: %s",timei(),j,buffer);
      cout << buffer2 << endl;
```

```
        //在其他客户端显示（广播给其他客户端）
        for (int k = 1; k <= total; k++)
        {
          send(cliSock[k], buffer2, sizeof(buffer), 0);
        }
      }
    }
}
```

该模块用于处理客户端发送来的消息。服务器端接收到消息后，会遍历每一个客户端，发现是哪个客户端发送来的消息，并且把这条消息发送给其他的所有客户端。

3.2.2 客户端

时间戳 timei 函数

```
char* timei()
{
  time_t t;
    struct tm *tmp;

  time(&t);
    tmp = localtime(&t);
  strftime(buf2, 64, "%Y-%m-%d %H:%M:%S", tmp);
  return buf2;
}
```

调用该函数即可返回 “%Y-%m-%d %H:%M:%S” 格式的时间戳字符串。

socket 模块

```
WSADATA wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData); //主版本号为2，副版本号为2
printf("%s [ OK   ] WSAStartup Complete!\n",timei());

SOCKET sockClient = socket(AF_INET, SOCK_STREAM, 0);
if(sockClient !=INVALID_SOCKET)
  printf("%s [ OK   ] Socket Created!\n",timei());
```

此模块发生在主线程中，首先初始化 Socket DLL，协商使用的 Socket 版本，完成后打印“WSAStartup Complete!”，接着创建一个 socket，并绑定到 TCP 传输层服务，如果创建成功则打印 “Socket Created !”。

connect 模块

```
printf("%s [ GET  ] Input the IP of server\n",timei());
char ip[20]={0};
gets(ip);
printf("%s [ GET  ] Input the Port of server\n",timei());
int port;
cin>>port;
```

```
7
8  //服务端
9  SOCKADDR_IN addrSrv = { 0 };
10 addrSrv.sin_family = AF_INET;
11 addrSrv.sin_addr.S_un.S_addr = inet_addr(ip);
12 addrSrv.sin_port = htons(port);
13
14 //客户端
15 SOCKADDR_IN cliAddr = { 0 };
16 cliAddr.sin_family = AF_INET;
17 cliAddr.sin_addr.s_addr = inet_addr("127.0.0.1");//IP地址
18 cliAddr.sin_port = htons(12345);//端口号
19
20 if (connect(sockClient, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR)) == SOCKET_ERROR)
21 {
22   cout << timei()<<" [ INFO ] 链接出现错误, 错误代码" << WSAGetLastError() << endl;
23 }
24 else
25   printf("%s [ INFO ] Server connected succesfully!\n",timei());
26
27 //创建接受消息线程
28 CloseHandle(CreateThread(NULL, 0, recvMsgThread, (LPVOID)&sockClient, 0, 0));
```

此模块发生在主线程中，首先请用户输入要链接的服务器的 IP 地址及端口号，然后初始化服务器端与客户端的地址结构。接着尝试与服务器端建立连接，如果建立连接成功，便打印 “[INFO] Server connected succesfully!”。最后创建接收消息的线程。

send 模块

```
1  while (1)
2  {
3    char buf[BUF_SIZE] = { 0 };
4    cin.getline(buf, sizeof(buf));
5    if (strcmp(buf, "exit") == 0)
6    {
7      break;
8    }
9    send(sockClient, buf, sizeof(buf), 0);
10 }
```

此模块发生在主线程中，用于得到用户在客户端输入的语句，并将字符串发送到服务器端。

clean 模块

```
1  closesocket(sockClient);
2  WSACleanup();
```

此模块位于主线程中，用于关闭服务器 socket，并且释放 socket DLL 资源。

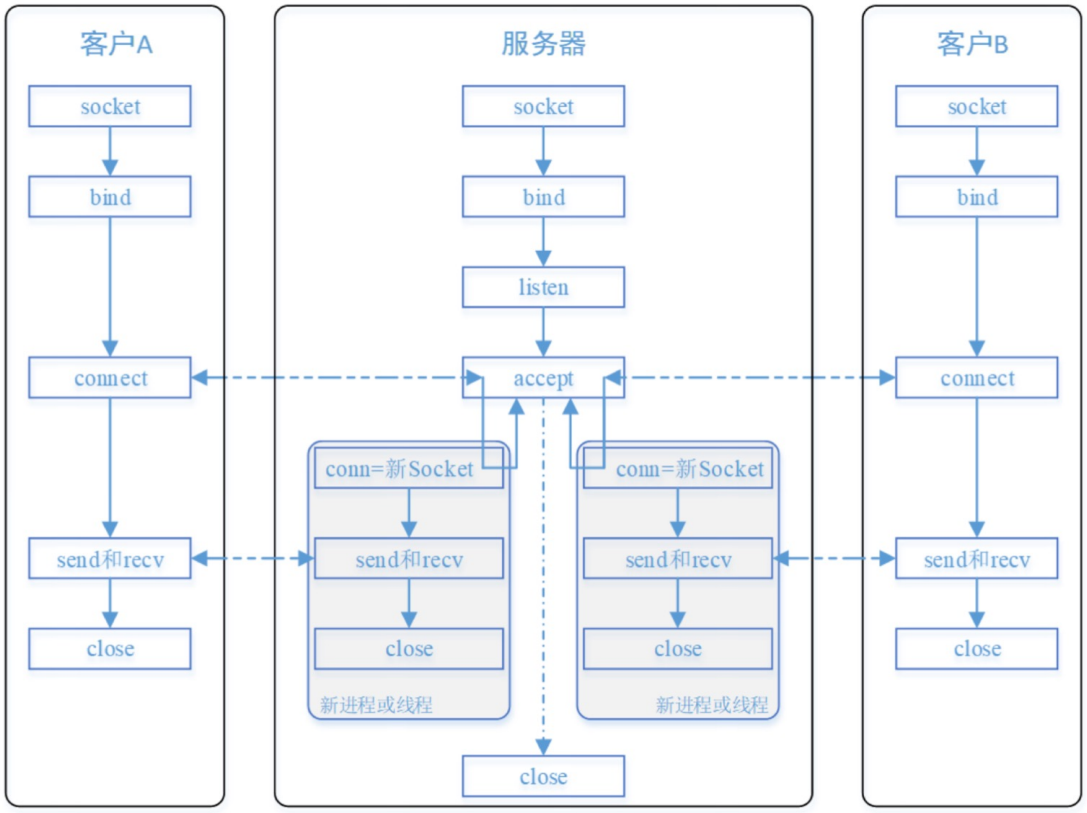
接下来的 recv 模块发生在另一个线程。

recv 模块

```
while (1)
{
  char buffer[BUF_SIZE] = { 0 };
  int nrecv = recv(cliSock, buffer, sizeof(buffer), 0);//nrecv是接收到的字节数
  if (nrecv > 0)//如果接收到的字符数大于0
  {
    cout << buffer << endl;
  }
  else if (nrecv < 0)//如果接收到的字符数小于0就说明断开连接
  {
    printf("%s [ INFO ] Disconnect from server\n",timei());
    break;
  }
}
```

该模块用于接收来自服务器端发送来的消息，并且将字符串打印在客户端。如果发现此时已经与服务器端断开了连接，则打印“Disconnect from server”。

3.3 模块间流程图



4 程序界面展示及运行说明

4.1 双人聊天模式

服务器开始运行

```
2022-10-22 21:14:41 [ INFO ] Start Server Manager
2022-10-22 21:14:41 [ OK   ] WSAStartup Complete!
2022-10-22 21:14:41 [ INFO ] Local Machine IP Address: 127.0.0.1
2022-10-22 21:14:41 [ OK   ] Socket Created!
2022-10-22 21:14:41 [ OK   ] Bind Success!
2022-10-22 21:14:41 [ INFO ] Broadcast thread create success!
2022-10-22 21:14:41 [ INFO ] Start listening...
```

用户 1 和用户 2 输入正确的 IP 地址和端口号成功与服务器端连接，进入聊天室。

C:\Windows\system32\cmd.exe

```
2022-10-22 21:14:41 [ INFO ] Start Server Manager
2022-10-22 21:14:41 [ OK   ] WSAStartup Complete!
2022-10-22 21:14:41 [ INFO ] Local Machine IP Address: 127.0.0.1
2022-10-22 21:14:41 [ OK   ] Socket Created!
2022-10-22 21:14:41 [ OK   ] Bind Success!
2022-10-22 21:14:41 [ INFO ] Broadcast thread create success!
2022-10-22 21:14:41 [ INFO ] Start listening...
2022-10-22 21:20:39 [ OK   ] Accept Success!
2022-10-22 21:20:39 [ JOIN ] user#1 just join, welcome!
2022-10-22 21:20:39 [ RECV ] user#1:
2022-10-22 21:21:22 [ OK   ] Accept Success!
2022-10-22 21:21:25 [ JOIN ] user#2 just join, welcome!
2022-10-22 21:21:25 [ RECV ] user#2:
```

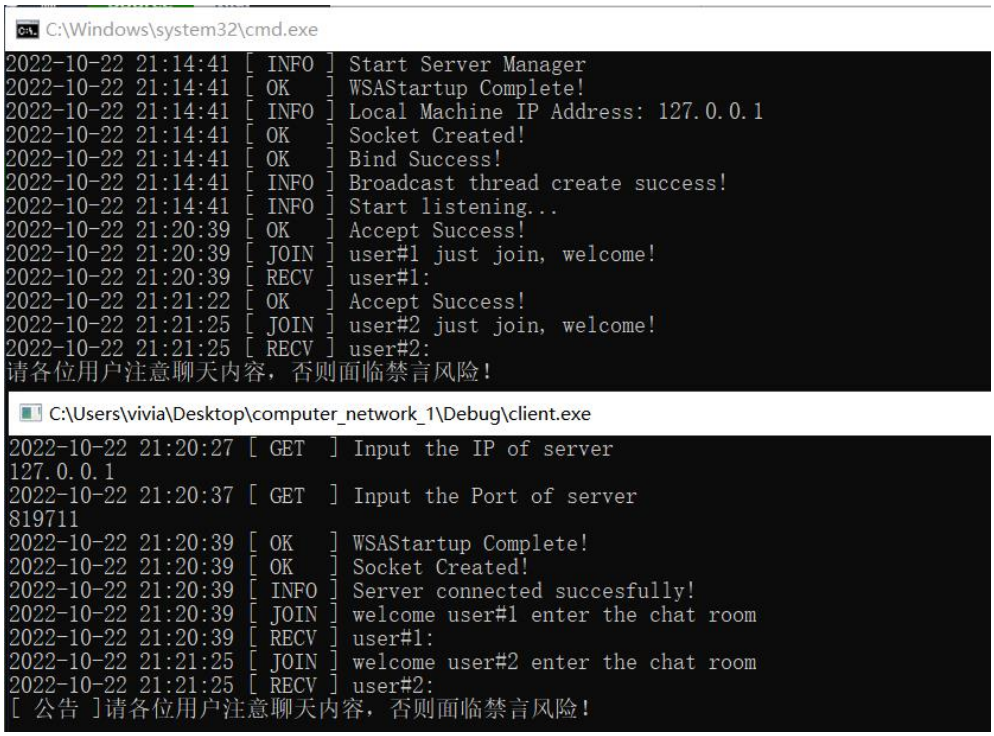
C:\Users\vivia\Desktop\computer_network_1\Debug\client.exe

```
2022-10-22 21:20:27 [ GET  ] Input the IP of server
127.0.0.1
2022-10-22 21:20:37 [ GET  ] Input the Port of server
819711
2022-10-22 21:20:39 [ OK   ] WSAStartup Complete!
2022-10-22 21:20:39 [ OK   ] Socket Created!
2022-10-22 21:20:39 [ INFO ] Server connected succesfully!
2022-10-22 21:20:39 [ JOIN ] welcome user#1 enter the chat room
2022-10-22 21:20:39 [ RECV ] user#1:
2022-10-22 21:21:25 [ JOIN ] welcome user#2 enter the chat room
2022-10-22 21:21:25 [ RECV ] user#2:
```

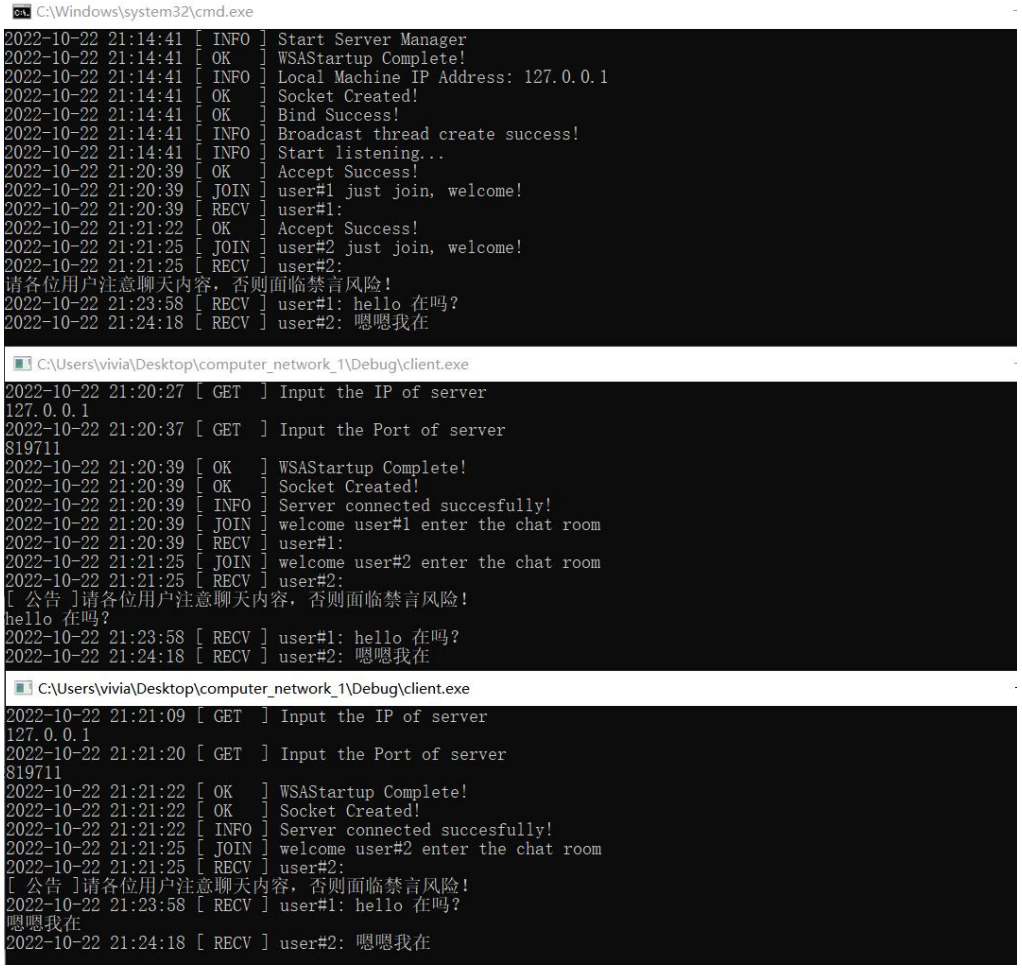
C:\Users\vivia\Desktop\computer_network_1\Debug\client.exe

```
2022-10-22 21:21:09 [ GET  ] Input the IP of server
127.0.0.1
2022-10-22 21:21:20 [ GET  ] Input the Port of server
819711
2022-10-22 21:21:22 [ OK   ] WSAStartup Complete!
2022-10-22 21:21:22 [ OK   ] Socket Created!
2022-10-22 21:21:22 [ INFO ] Server connected succesfully!
2022-10-22 21:21:25 [ JOIN ] welcome user#2 enter the chat room
2022-10-22 21:21:25 [ RECV ] user#2:
```

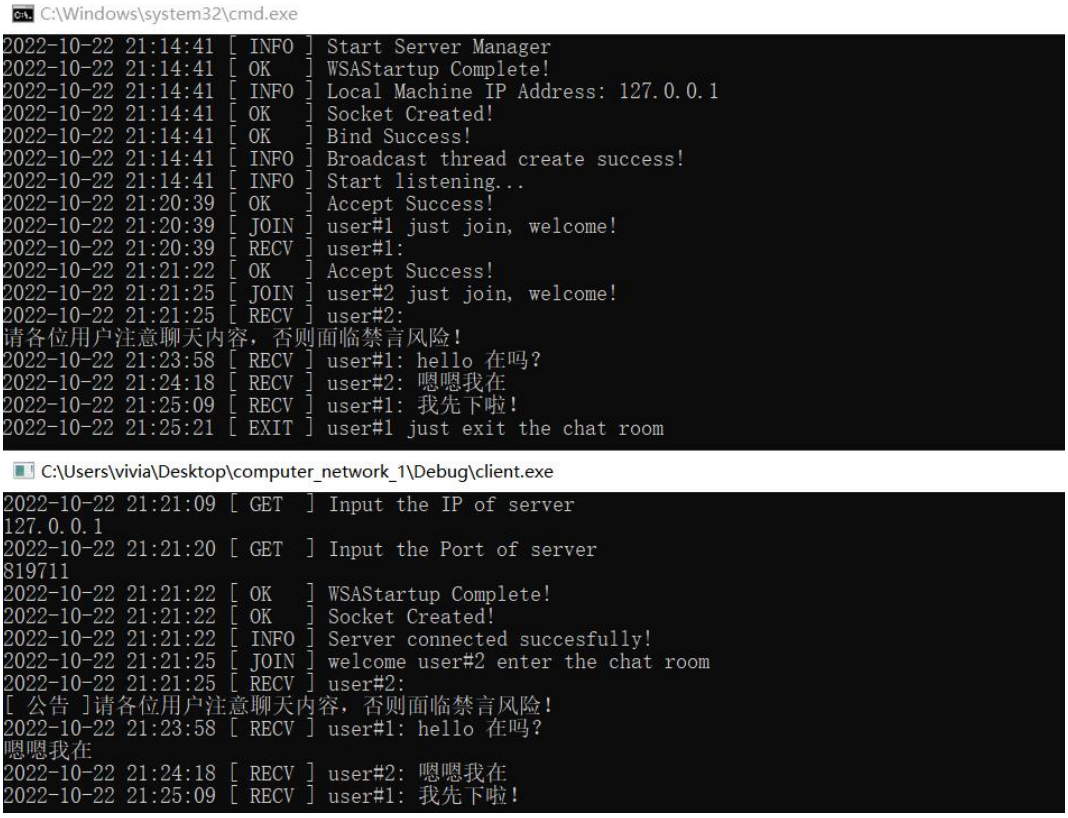
从服务器端向客户端发出公告



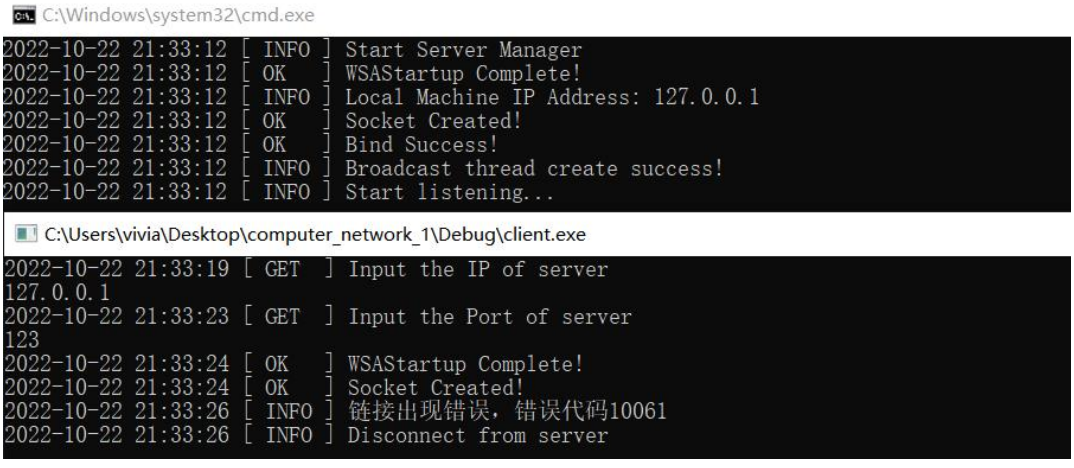
用户 1 和用户 2 开始聊天



用户 1 下线

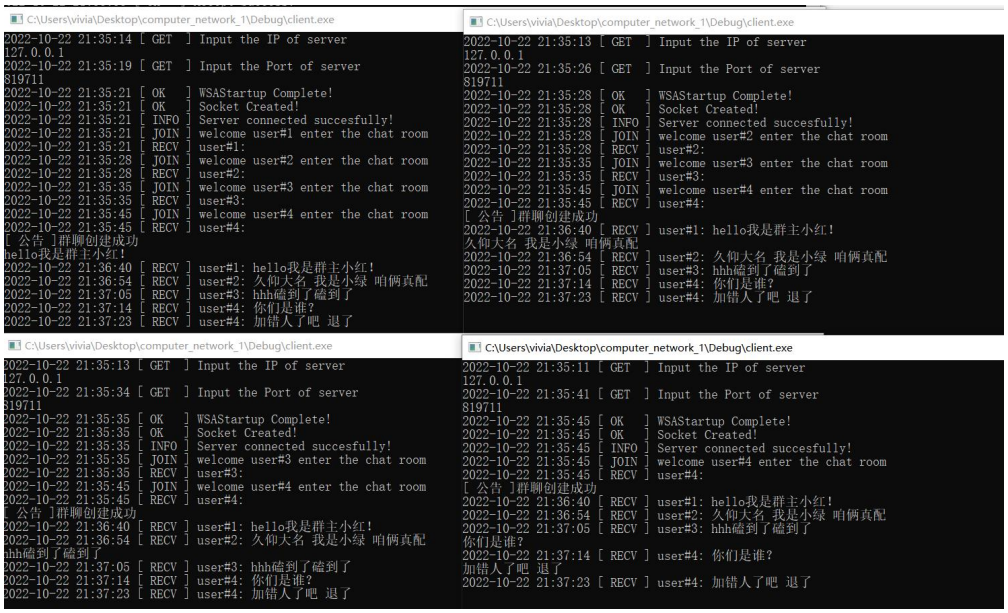


用户 3 输入错误端口号，与服务器端连接失败

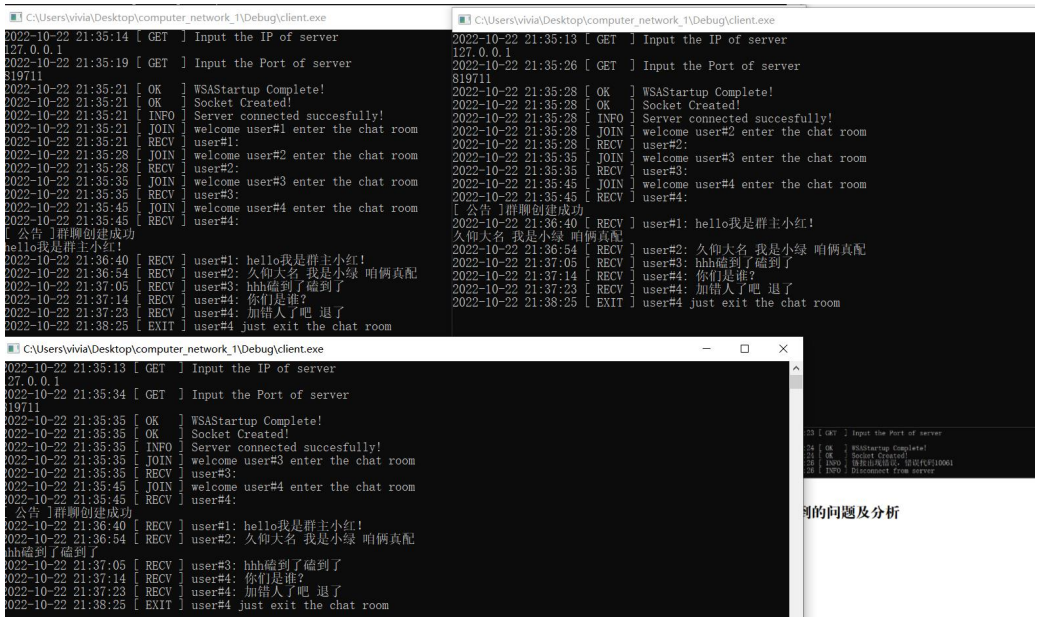


4.2 多人群聊模式

四个人与服务器连接成功，服务器广播“群聊创建成功”。四个人开始了他们的聊天:



用户 4 进错了群，退出了群聊



5 实验反思

在本次实验中，经过反思我认为有以下几点可以进一步改进：

- 可以进一步扩展多线程的用途，让每一个客户端都能在服务器端的一个线程上处理 accept,close 和 recv 消息
- 可以在群聊中加入功能，实现三人以上群聊中，可以实现两人间的私密消息（类似于腾讯会议里面的功能）
- 可以在一个新的客户端加入时，让用户输入关于这个用户的基本信息，并且群聊中的其他用户可以查看彼此的基本信息。