

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机网络基础 |
| 实验名称： | 基于Socket接口实现自定义协议通信 |
| 姓 名： |  |
| 学 院： | 计算机学院 |
| 系： | 计算机系 |
| 专 业： | 计算机科学与技术 |
| 学 号： |  |
| 指导教师： | 陆魁军 |

| **姓名** | **学号** | **分工** | **工作量** |
| --- | --- | --- | --- |
| 徐铭 | 3210102037 | Server端所有内容 | 50% |
| 程天乐 | 3210102097 | Client端所有内容 | 50% |

2023年 10月 9日

**浙江大学实验报告**

实验名称： 基于Socket接口实现自定义协议通信 实验类型： 编程实验

同组学生： 徐铭、程天乐 实验地点： 计算机网络实验室

# 实验目的

* 学习如何设计网络应用协议
* 掌握Socket编程接口编写基本的网络应用软件

# 实验内容

根据自定义的协议规范，使用Socket编程接口编写基本的网络应用软件。

* 掌握C语言形式的Socket编程接口用法，能够正确发送和接收网络数据包
* 开发一个客户端，实现人机交互界面和与服务器的通信
* 开发一个服务端，实现并发处理多个客户端的请求
* 程序界面不做要求，使用命令行或最简单的窗体即可
* 功能要求如下：

1. 运输层协议采用TCP
2. 客户端采用交互菜单形式，用户可以选择以下功能：
   1. 连接：请求连接到指定地址和端口的服务端
   2. 断开连接：断开与服务端的连接
   3. 获取时间: 请求服务端给出当前时间
   4. 获取名字：请求服务端给出其机器的名称
   5. 活动连接列表：请求服务端给出当前连接的所有客户端信息（编号、IP地址、端口等）
   6. 发消息：请求服务端把消息转发给对应编号的客户端，该客户端收到后显示在屏幕上
   7. 退出：断开连接并退出客户端程序
3. 服务端接收到客户端请求后，根据客户端传过来的指令完成特定任务：
   1. 向客户端传送服务端所在机器的当前时间
   2. 向客户端传送服务端所在机器的名称
   3. 向客户端传送当前连接的所有客户端信息
   4. 将某客户端发送过来的内容转发给指定编号的其他客户端
   5. 采用异步多线程编程模式，正确处理多个客户端同时连接，同时发送消息的情况

* 根据上述功能要求，设计一个客户端和服务端之间的应用通信协议
* **本实验涉及到网络数据包发送部分不能使用任何的Socket封装类，只能使用最底层的C语言形式的Socket API**
* 本实验可组成小组，服务端和客户端可由不同人来完成

# 主要仪器设备

* 联网的PC机、Wireshark软件
* Visual C++、gcc等C++集成开发环境。

# 操作方法与实验步骤

* 设计请求、指示（服务器主动发给客户端的）、响应数据包的格式，至少要考虑如下问题：

1. 定义两个数据包的边界如何识别
2. 定义数据包的请求、指示、响应类型字段
3. 定义数据包的长度字段或者结尾标记
4. 定义数据包内数据字段的格式（特别是考虑客户端列表数据如何表达）

* 小组分工：1人负责编写服务端，1人负责编写客户端
* 客户端编写步骤（**需要采用多线程模式**）

1. 运行初始化，调用socket()，向操作系统申请socket句柄
2. 编写一个菜单功能，列出7个选项
3. 等待用户选择
4. 根据用户选择，做出相应的动作（未连接时，只能选连接功能和退出功能）
5. 选择连接功能：请用户输入服务器IP和端口，然后调用connect()，等待返回结果并打印。连接成功后设置连接状态为已连接。**然后创建一个接收数据的子线程，循环调用receive()，如果收到了一个完整的响应数据包，就通过线程间通信（如消息队列）发送给主线程，然后继续调用receive()，直至收到主线程通知退出。**
6. 选择断开功能：调用close()，并设置连接状态为未连接。通知并等待子线程关闭。
7. 选择获取时间功能：组装请求数据包，类型设置为时间请求，然后调用send()将数据发送给服务器，**接着等待接收数据的子线程返回结果**，并根据响应数据包的内容，打印时间信息。
8. 选择获取名字功能：组装请求数据包，类型设置为名字请求，然后调用send()将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印名字信息。
9. 选择获取客户端列表功能：组装请求数据包，类型设置为列表请求，然后调用send()将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印客户端列表信息（编号、IP地址、端口等）。
10. 选择发送消息功能（选择前需要先获得客户端列表）：请用户输入客户端的列表编号和要发送的内容，然后组装请求数据包，类型设置为消息请求，然后调用send()将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印消息发送结果（是否成功送达另一个客户端）。
11. 选择退出功能：判断连接状态是否为已连接，是则先调用断开功能，然后再退出程序。否则，直接退出程序。
12. 主线程除了在等待用户的输入外，还在处理子线程的消息队列，如果有消息到达，则进行处理，如果是响应消息，则打印响应消息的数据内容（比如时间、名字、客户端列表等）；如果是指示消息，则打印指示消息的内容（比如服务器转发的别的客户端的消息内容、发送者编号、IP地址、端口等）。

* 服务端编写步骤（**需要采用多线程模式**）

1. 运行初始化，调用socket()，向操作系统申请socket句柄
2. 调用bind()，绑定监听端口（**请使用学号的后4位作为服务器的监听端口**），接着调用listen()，设置连接等待队列长度
3. 主线程循环调用accept()，直到返回一个有效的socket句柄，在客户端列表中增加一个新客户端的项目，并记录下该客户端句柄和连接状态、端口。然后创建一个子线程后继续调用accept()。该子线程的主要步骤是（**刚获得的句柄要传递给子线程，子线程内部要使用该句柄发送和接收数据**）：

* 调用send()，发送一个hello消息给客户端（可选）
* 循环调用receive()，如果收到了一个完整的请求数据包，根据请求类型做相应的动作：

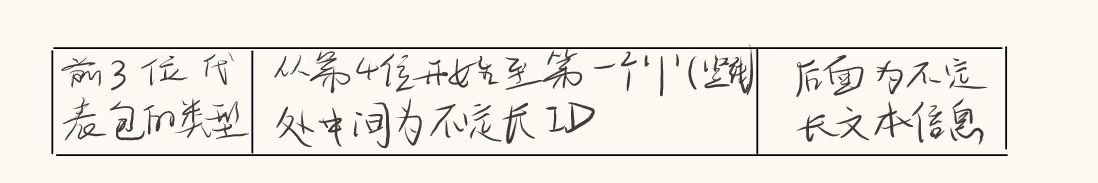
1. 请求类型为获取时间：调用time()获取本地时间，然后将时间数据组装进响应数据包，调用send()发给客户端
2. 请求类型为获取名字：将服务器的名字组装进响应数据包，调用send()发给客户端
3. 请求类型为获取客户端列表：读取客户端列表数据，将编号、IP地址、端口等数据组装进响应数据包，调用send()发给客户端
4. 请求类型为发送消息：根据编号读取客户端列表数据，如果编号不存在，将错误代码和出错描述信息组装进响应数据包，调用send()发回源客户端；如果编号存在并且状态是已连接，则将要转发的消息组装进指示数据包。调用send()发给接收客户端（使用接收客户端的socket句柄），发送成功后组装转发成功的响应数据包，调用send()发回源客户端。
5. 主线程还负责检测退出指令（如用户按退出键或者收到退出信号），检测到后即通知并等待各子线程退出。最后关闭Socket，主程序退出。

* 编程结束后，双方程序运行，检查是否实现功能要求，如果有问题，查找原因，并修改，直至满足功能要求
* 使用多个客户端同时连接服务端，检查并发性
* 使用Wireshark抓取每个功能的交互数据包

# 实验数据记录和处理

**请将以下内容和本实验报告一起打包成一个压缩文件上传：**

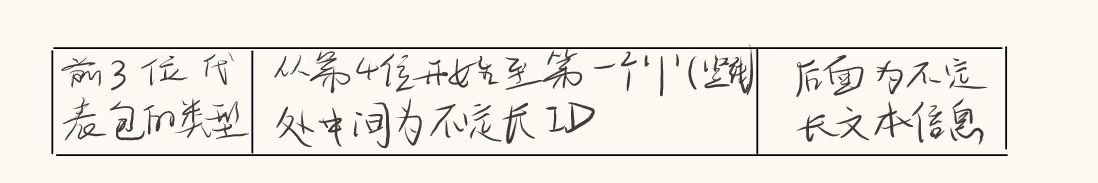
* **源代码：客户端和服务端的代码分别在一个目录**
* **可执行文件：可运行的.exe文件或Linux可执行文件，客户端和服务端各一个**
* **描述请求数据包的格式（画图说明），请求类型的定义**



前三位表示请求类型：

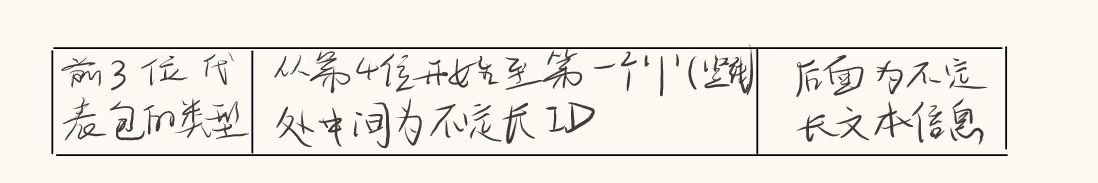
000:连接  
001:断开连接  
010:获取时间  
011:获取名字  
100:获取所有用户信息  
101:发消息

* **描述响应数据包的格式（画图说明），响应类型的定义**



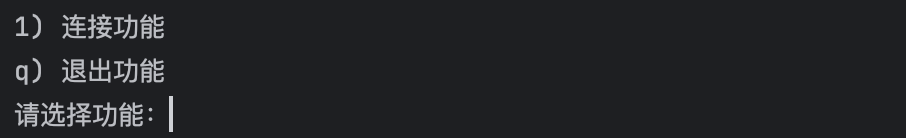
010:返回时间  
011:返回名字  
100:返回所有用户信息  
101:返回发消息是否成功（发送客户端）或发送的消息（接受客户端）

* **描述指示数据包的格式（画图说明），指示类型的定义**

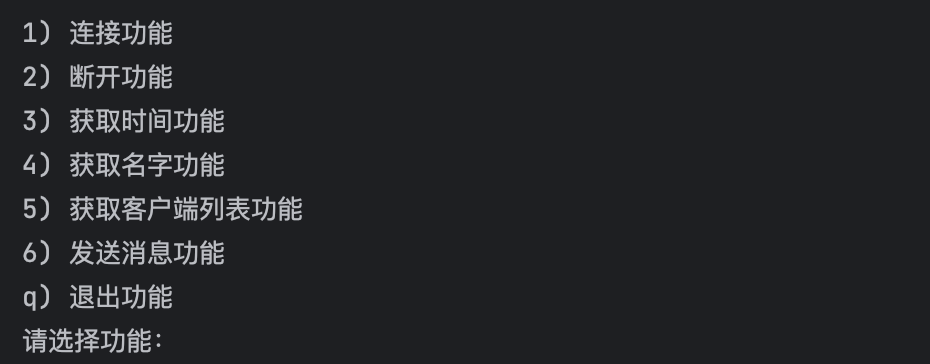


000:连接信息

* **客户端初始运行后显示的菜单选项**

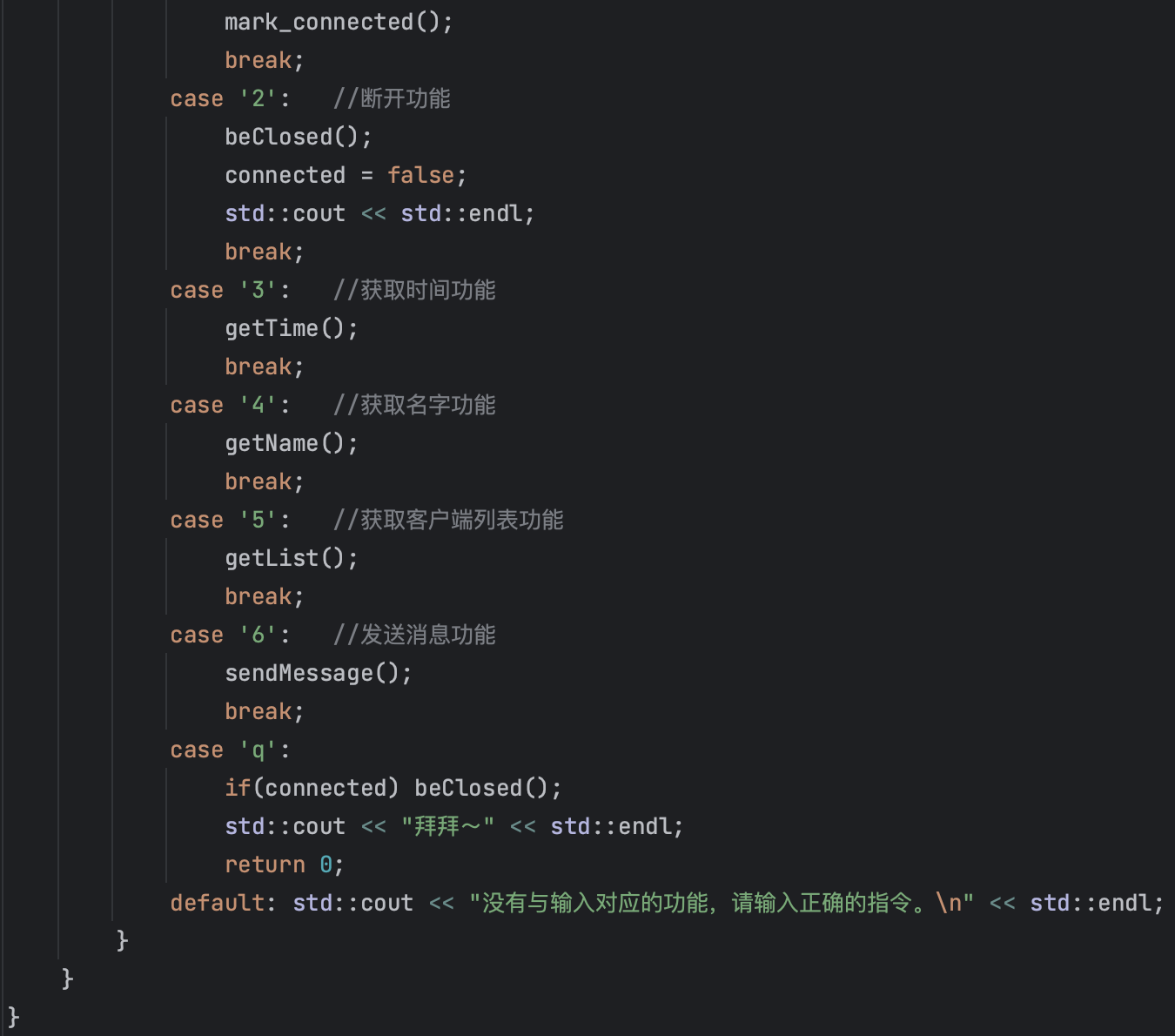


在连接服务器后菜单选项会变为：



* **客户端的主线程循环关键代码截图（描述总体，省略细节部分）**





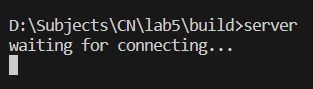
根据用户选择的功能调用对应的功能函数，直到用户输入q退出为止。

* **客户端的接收数据子线程循环关键代码截图（描述总体，省略细节部分）**

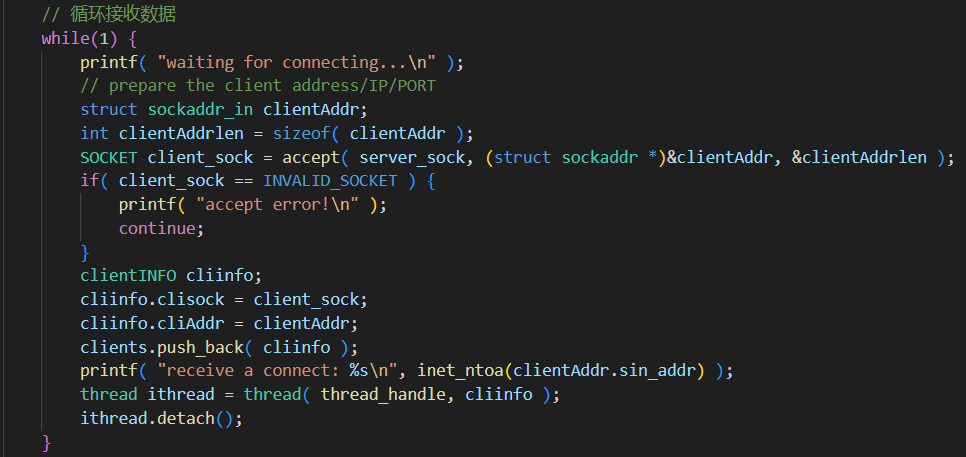


将接收到的字符串信息解析为定义的包类型，然后根据包的内容，输出对应的信息。

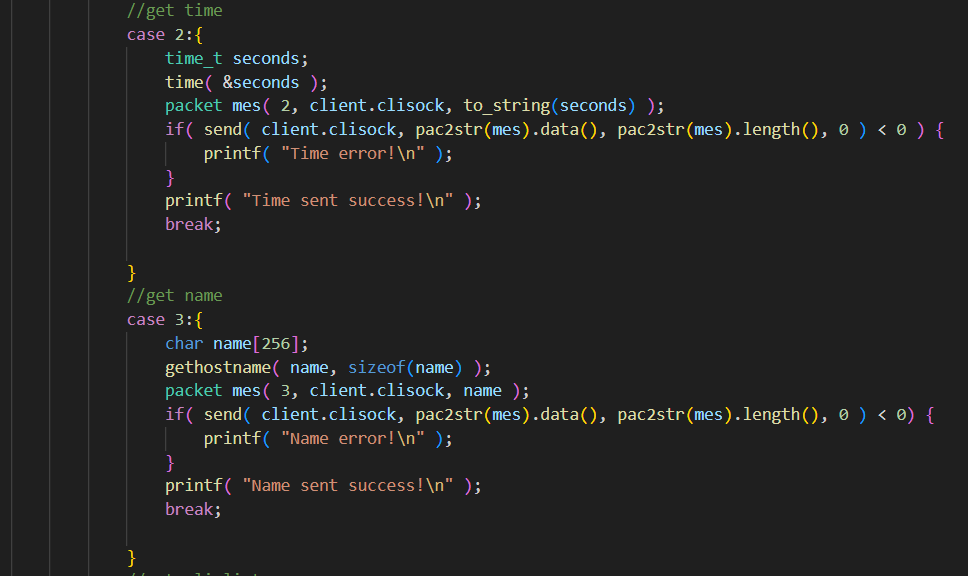
* **服务器初始运行后显示的界面**



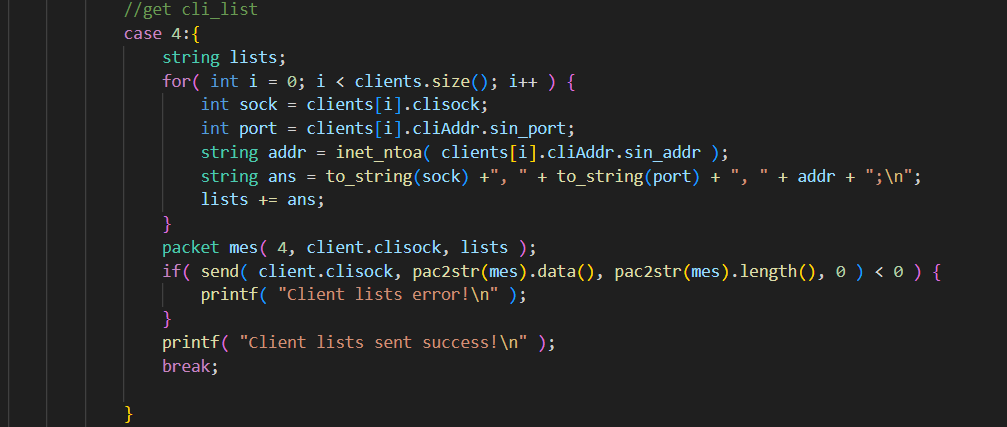
* **服务器的主线程循环关键代码截图（描述总体，省略细节部分）**

如上图所示，主进程开启后，在一个死循环内每接受到一个client连接就创建一个子线程去处理这个连接，并在主进程中通过线程池去管理这些子线程；

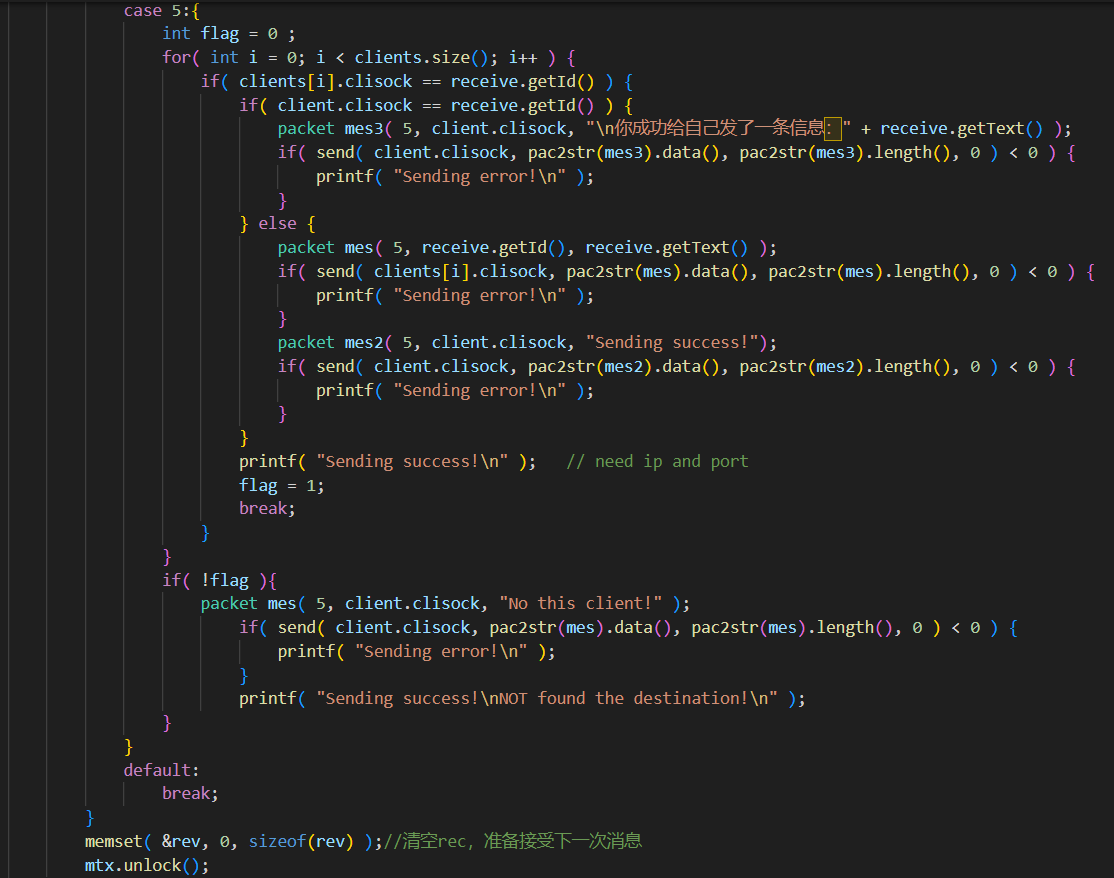
* **服务器的客户端处理子线程循环关键代码截图（描述总体，省略细节部分）**



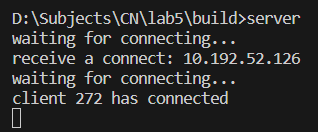
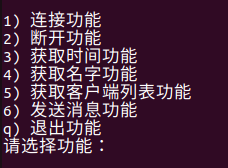
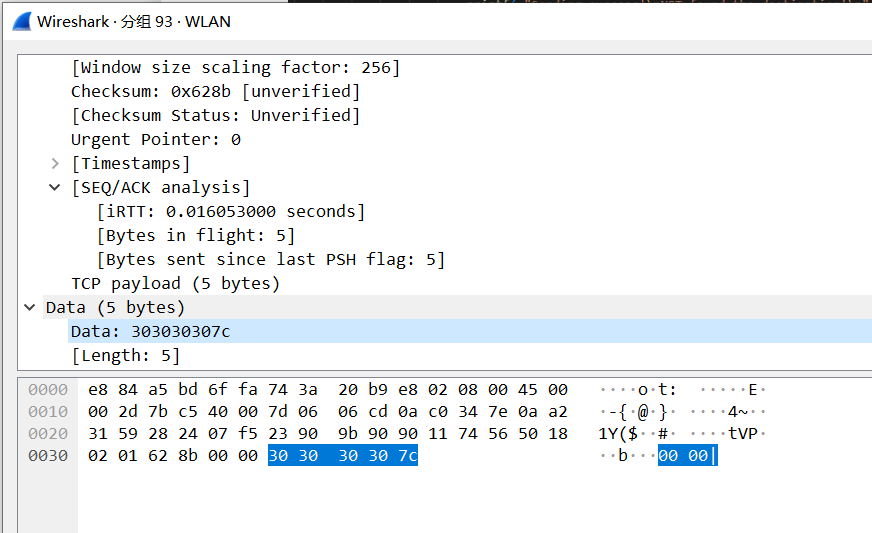
子线程处理每个连接请求的过程也是一个循环，只有当client和服务端处于连接状态时会持续循环，否则直接结束；



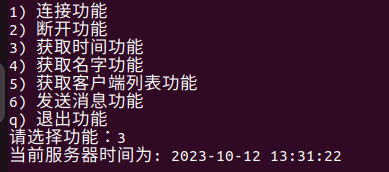
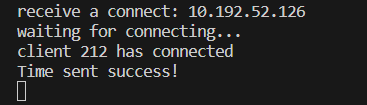
如果客户端请求时间、名字、用户列表，这里将会进行一次信息的组装，用上面规定的协议组成相应数据包，将内容通过 send发回。而如果是向其他用户发送消息，则会解析出编号、IP 和发送内容，找到已经在连接里的这个客户，并向他发送一条消息；



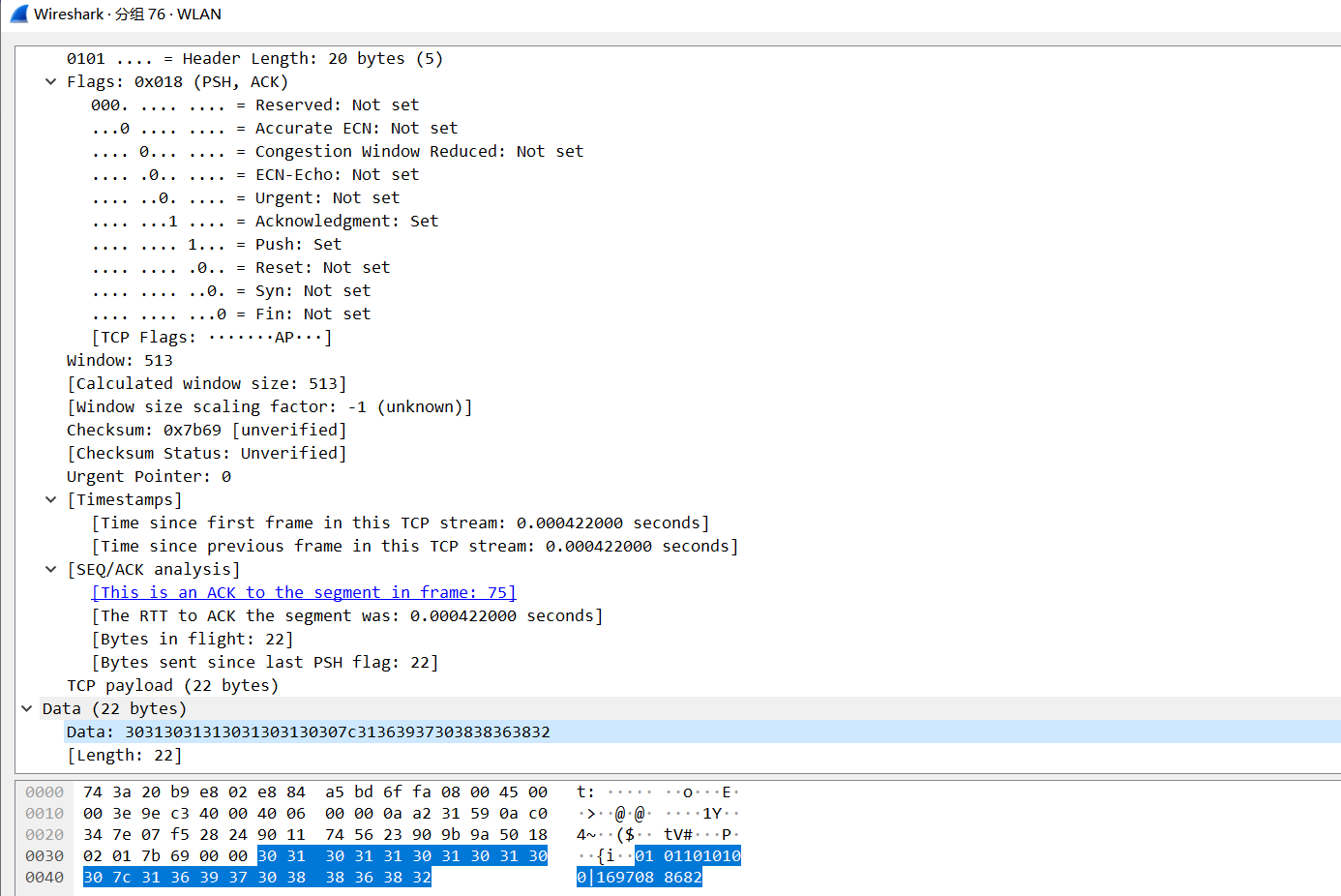
* **客户端选择连接功能时，客户端和服务端显示内容截图。**

 Wireshark抓取的数据包截图：

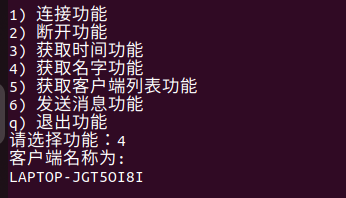
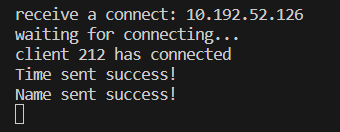
* **客户端选择获取时间功能时，客户端和服务端显示内容截图。**



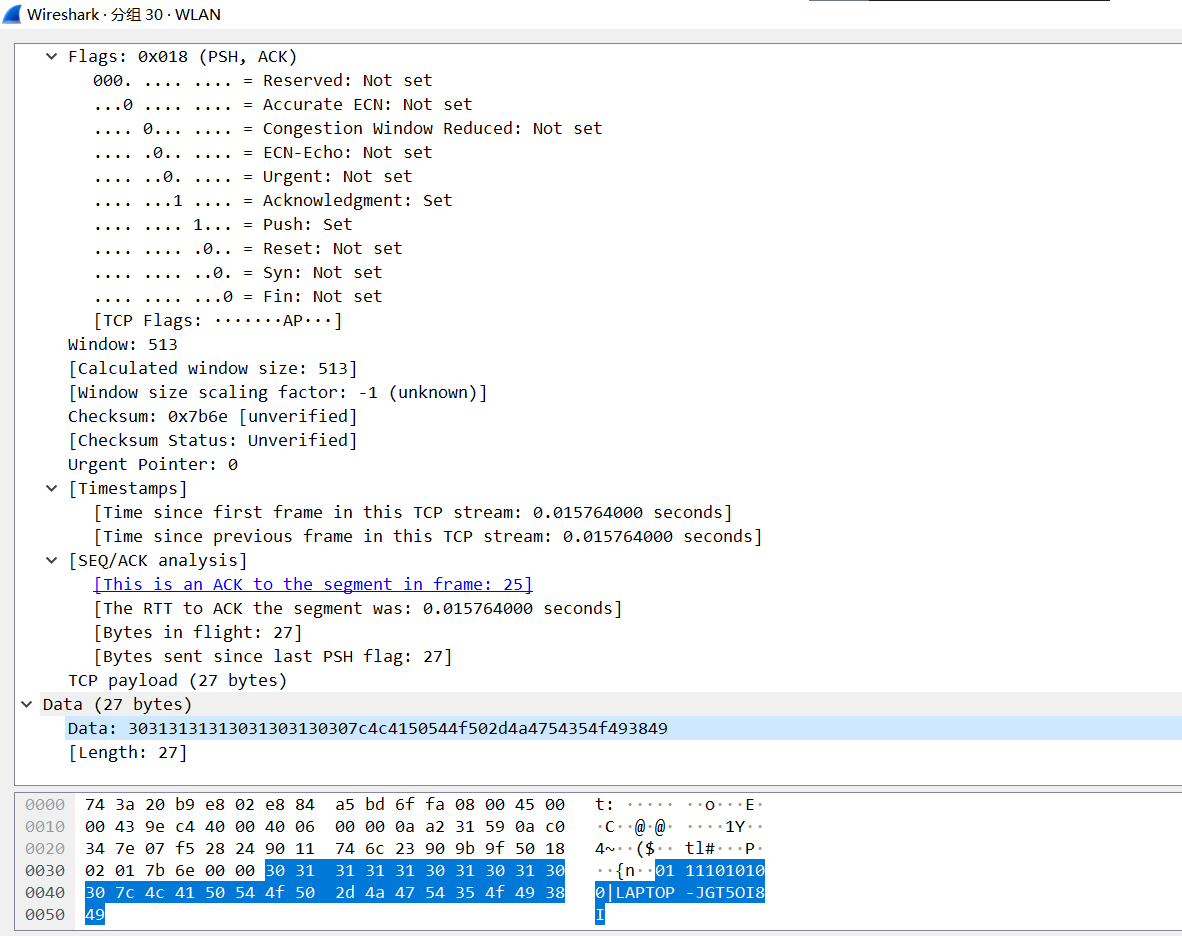
Wireshark抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的时间数据对应的位置）：



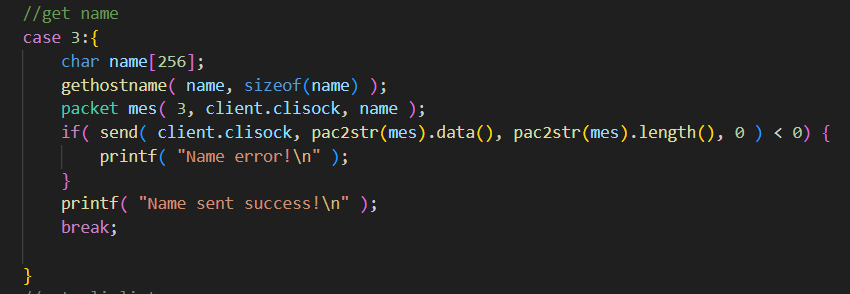
* **客户端选择获取名字功能时，客户端和服务端显示内容截图。**



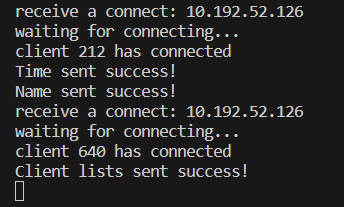
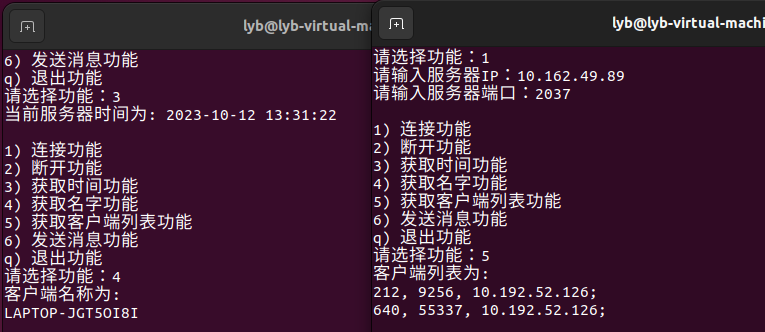
Wireshark抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的名字数据对应的位置）：

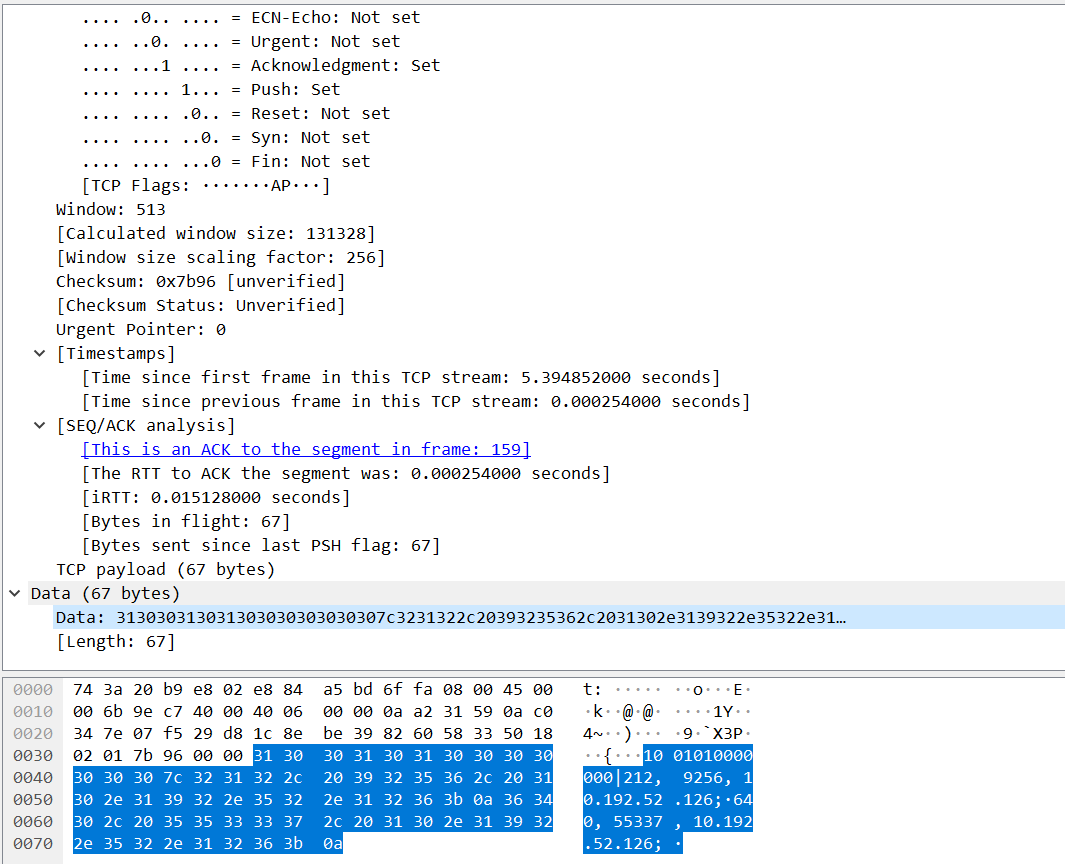


相关的服务器的处理代码片段：

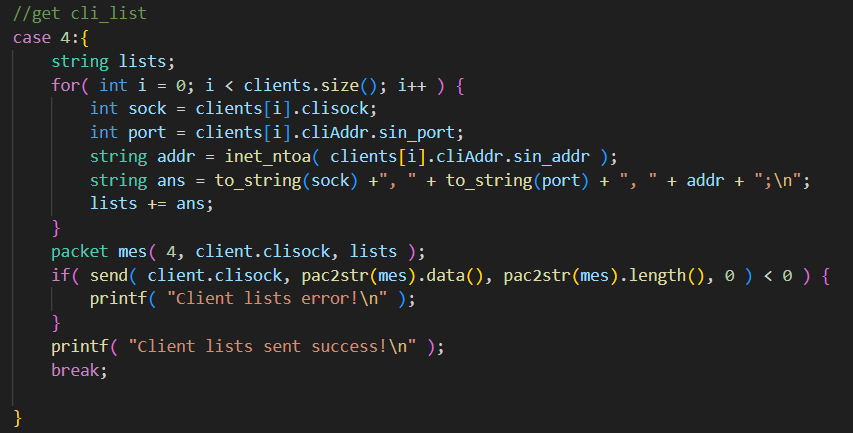


* **客户端选择获取客户端列表功能时，客户端和服务端显示内容截图。**

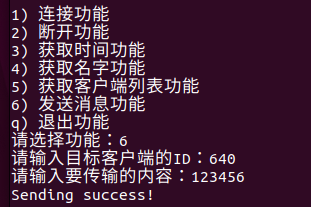


Wireshark抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的客户端列表数据对应的位置）：

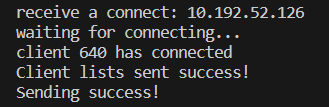
相关的服务器的处理代码片段：



* **客户端选择发送消息功能时，客户端和服务端显示内容截图。**

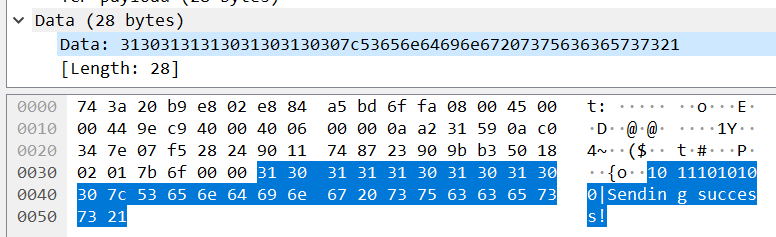
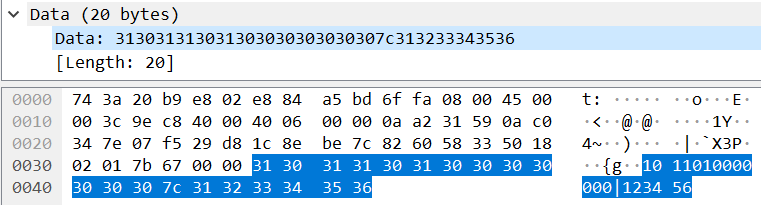
发送消息的客户端：

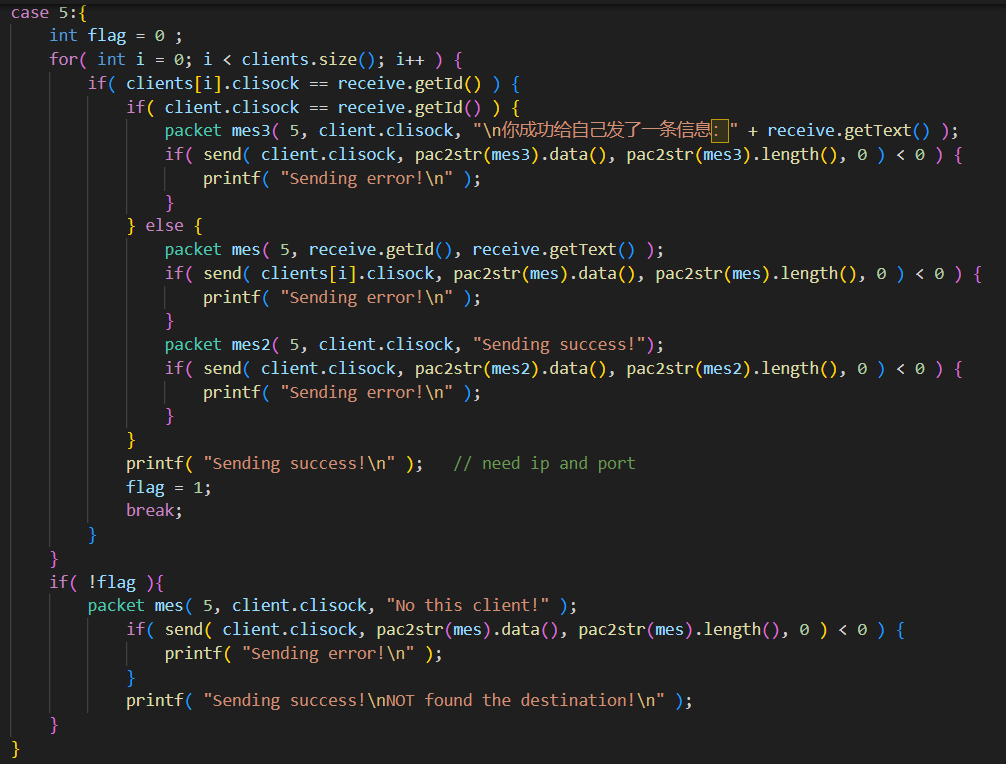
服务器：

接收消息的客户端：

c0432e8802f2877c1f89a78dc498a0e

Wireshark抓取的数据包截图（发送和接收分别标记）：



相关的服务器的处理代码片段：

相关的客户端（发送和接收消息）处理代码片段：



通过waitingStatus来判断是发送端还是接受端：如果waitingStatus为true，则说明是发送端，输出得到的包中的反馈信息；如果waitingStatus为false，则说明是接收端，输出收到的信息。

* **拔掉客户端的网线，然后退出客户端程序。观察客户端的TCP连接状态，并使用Wireshark观察客户端是否发出了TCP连接释放的消息。同时观察服务端的TCP连接状态在较长时间内（10分钟以上）是否发生变化。**

答：从 wireshark 抓包观察得到，客户端并没有释放 TCP 连接释放的消息，但一段时间之后，

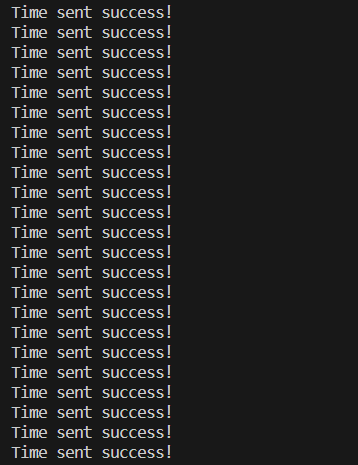
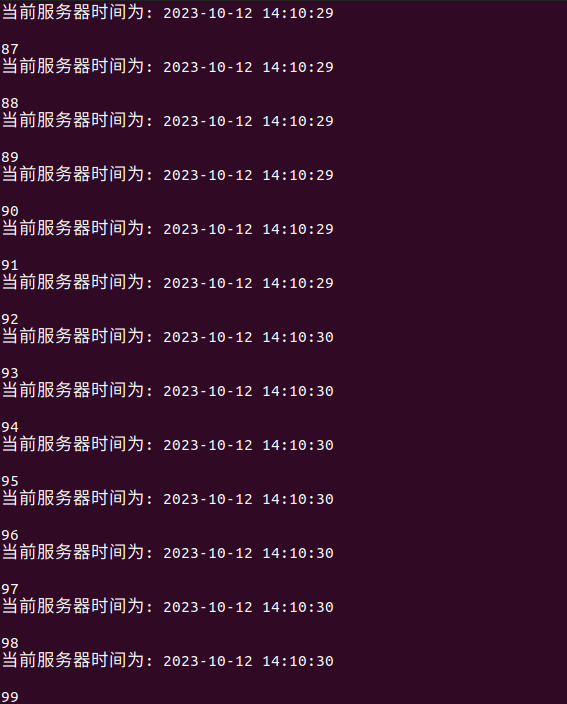
服务器的连接断开

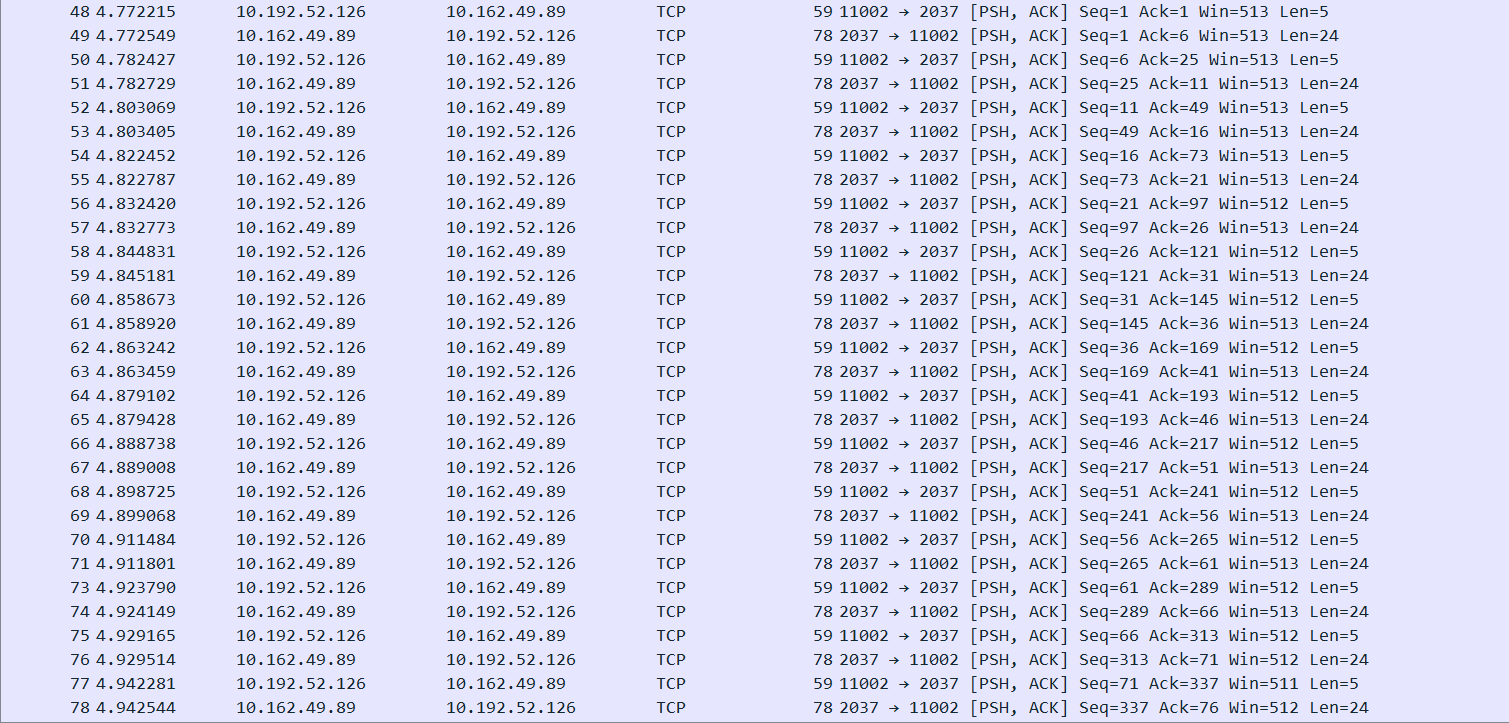
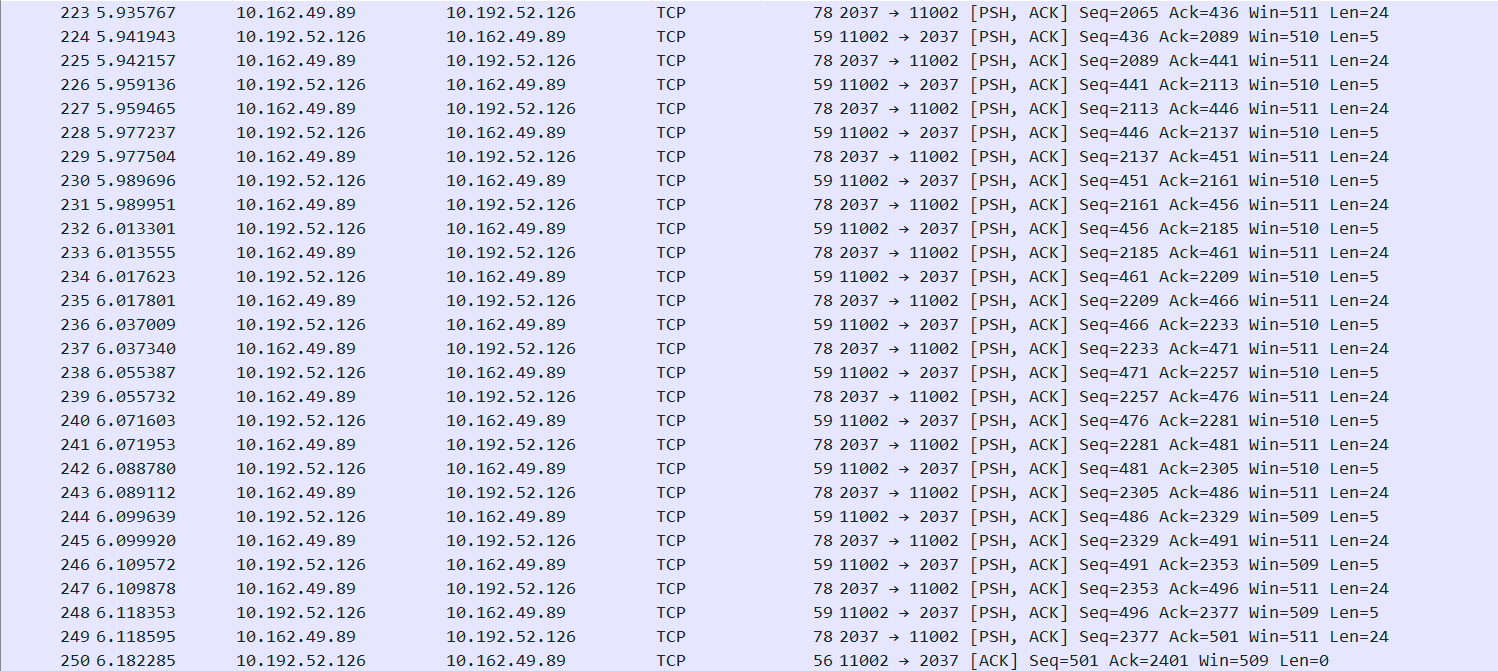
* **再次连上客户端的网线，重新运行客户端程序。选择连接功能，连上后选择获取客户端列表功能，查看之前异常退出的连接是否还在。选择给这个之前异常退出的客户端连接发送消息，出现了什么情况？**

答：从客户端列表中发现之前异常退出的的连接并不存在，而且无法给其发消息，因为服务器

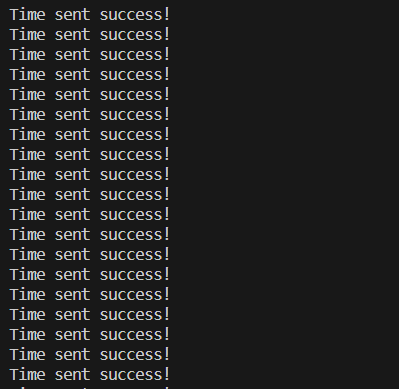
已经断掉了与其的连接

* **修改获取时间功能，改为用户选择1次，程序内自动发送100次请求。服务器是否正常处理了100次请求，截取客户端收到的响应（通过程序计数一下是否有100个响应回来），并使用Wireshark抓取数据包，观察实际发出的数据包个数。**





* **多个客户端同时连接服务器，同时发送时间请求（程序内自动连续调用100次send），服务器和客户端的运行截图**



# 实验结果与分析

* **客户端是否需要调用bind操作？它的源端口是如何产生的？每一次调用connect时客户端的端口是否都保持不变？**

不需要。

在客户端建立与服务器的连接时，客户端的操作系统会为该连接分配一个临时的源端口号。

是会发生变化的。

* **假设在服务端调用listen和调用accept之间设了一个调试断点，暂停在此断点时，此时客户端调用connect后是否马上能连接成功？**

答：能够连接成功

* **连续快速send多次数据后，通过Wireshark抓包看到的发送的Tcp Segment次数是否和send的次数完全一致？**

答：不完全一致

* **服务器在同一个端口接收多个客户端的数据，如何能区分数据包是属于哪个客户端的？**

答：如果IP地址不同，则通过IP地址区分；如果IP地址相同，则通过客户端产生的 Socket 描述符或者连接时间的先后加以区分。

* **客户端主动断开连接后，当时的TCP连接状态是什么？这个状态保持了多久？（可以使用netstat -an查看）**

答：TCP 的连接状态为 TIME\_WAIT，维持了大约70s。

* **客户端断网后异常退出，服务器的TCP连接状态有什么变化吗？服务器该如何检测连接是否继续有效？**

答：服务端的连接状态没有变化；可以尝试向客户端发送一个数据包。观察是否有回应或是设定超时时间，当客户端间隔一定时间没有进行操作时，服 务端自动断开连接。

# 讨论、心得

本次实验中客户端部分比较难的部分就是多线程的处理。因为是第一次写涉及多线程的程序，所以写起来有一定的难度。通过不断地查询相关资料以及和同学之间的交流，最终还是解决了这些问题，成功地按要求完成了这次实验。