浙江大学

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：	张佳瑶
学　　院：	计算机学院
系：	计算机科学与技术学院
专　　业：	软件工程专业
学　　号：	3170103240
指导教师：	高艺

2019 年 10 月 20 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：贺婷婷　实验地点：计算机网络实验室

一、 实验目的

- 掌握 Socket 编程接口编写基本的网络应用软件

二、 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：
     a) 连接：请求连接到指定地址和端口的服务端
     b) 断开连接：断开与服务端的连接
     c) 获取时间: 请求服务端给出当前时间
     d) 获取名字：请求服务端给出其机器的名称
     e) 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）
     f) 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
     g) 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：
     a) 向客户端传送服务端所在机器的当前时间
     b) 向客户端传送服务端所在机器的名称
     c) 向客户端传送当前连接的所有客户端信息
     d) 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
     e) 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
- **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

三、 主要仪器设备

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

四、 操作方法与实验步骤

- 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**）

a) 运行初始化，调用 socket()，向操作系统申请 socket 句柄

b) 编写一个菜单功能，列出 7 个选项

c) 等待用户选择

d) 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）

1. 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()，直至收到主线程通知退出。**
2. 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。
3. 选择获取时间功能：调用 send()将获取时间请求发送给服务器，**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。
4. 选择获取名字功能：调用 send()将获取名字请求发送给服务器，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。
5. 选择获取客户端列表功能：调用 send()将获取客户端列表信息请求发送给服务器，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）。
6. 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，然后调用 send()将数据发送给服务器，观察另外一个客户端是否收到数据。
7. 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。
8. 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**）

a) 运行初始化，调用 socket()，向操作系统申请 socket 句柄

b) 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

c) 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，并调用 send()发给客户端
2. 请求类型为获取名字：调用 GetComputerName 获取本机名，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、端口等数据通过调用 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，将要转发的消息组

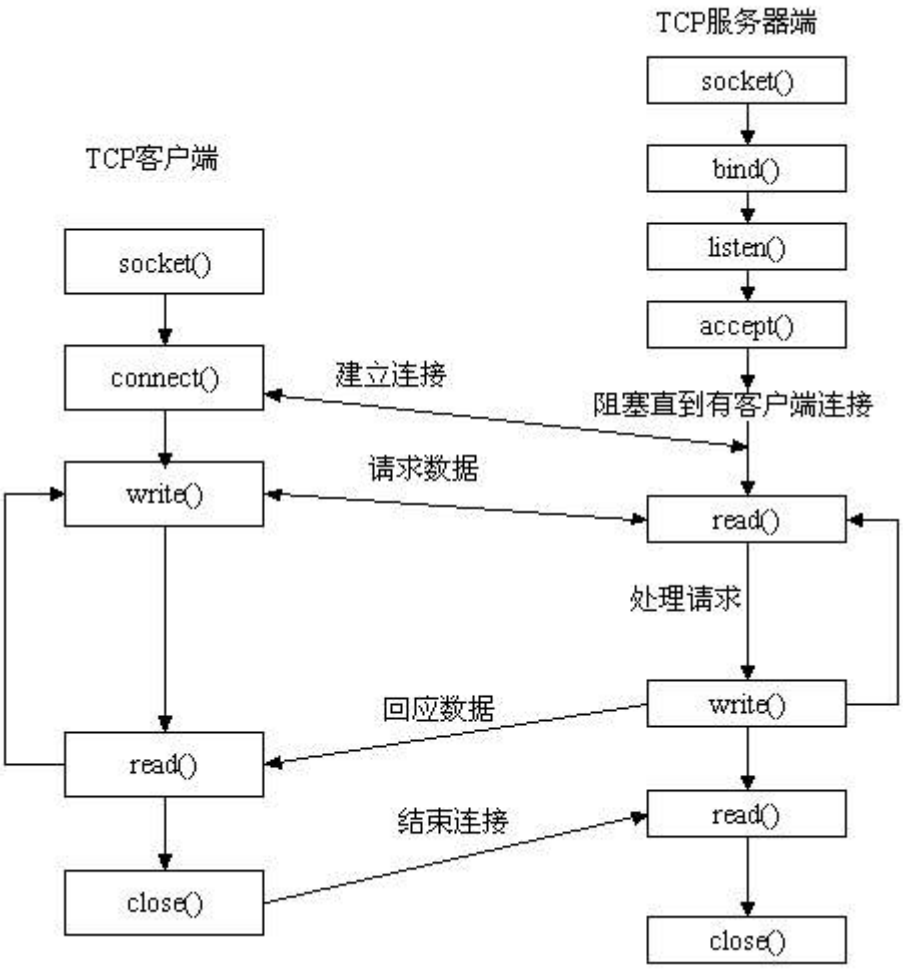
装通过调用 send()发给接收客户端（使用接收客户端的 socket 句柄）。

- 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

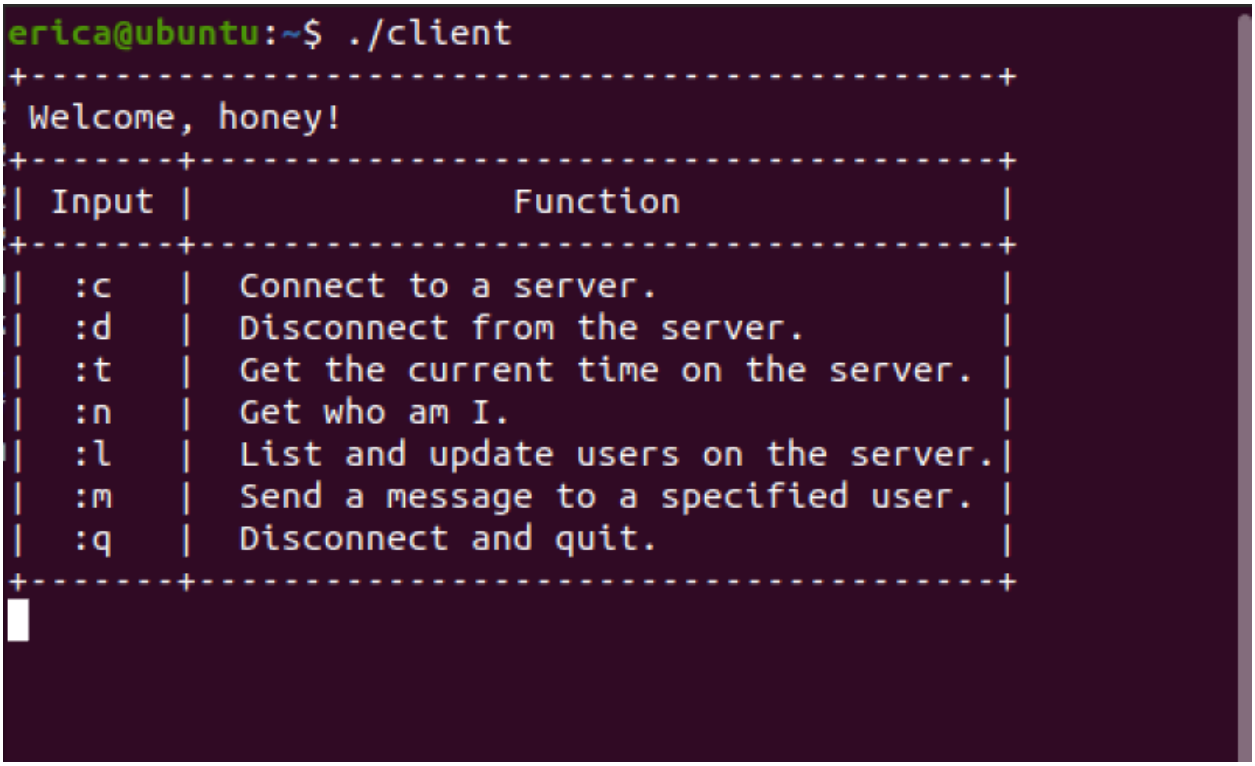
五、 实验数据记录和处理

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**
- 客户端和服务器端框架图（用流程图表示）



- 客户端初始运行后显示的菜单选项



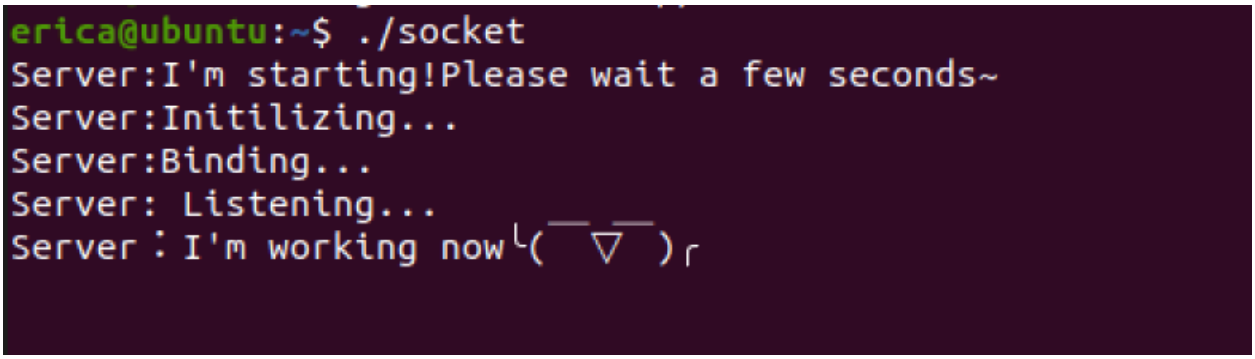
- 客户端的接收数据子线程循环关键代码截图（描述总体，省略细节部分）

```
void *receiveMsg(void *)
{
    char msg[BUFFER_SIZE];
    int strLen;

    while (true)
    {
        if (quit)
        {
            printf("Message receive thread quit...\n")
;
```

```
            pthread_exit(NULL);
        }
        else
        {
            //deal with the message
            strLen = recv(serverSocketFD, msg, BUFFER_
SIZE, 0);
            msg[strLen] = 0;
            interpretMsg(msg);
        }
    }
    return NULL;
}
```

连接成功后，创建接收信息的子线程，开始循环接收信息，解析接收到的信息。如果检测到连接退出指令 quit，就关闭子线程。

- 服务器初始运行后显示的界面



- 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）

```
while (true)
    {
        //get the conn of client
        struct sockaddr_in clientAddr;
        socklen_t addrLength = sizeof(clientAddr);
        //conn- the TCP connection socket descriptor
        int conn = accept(sockfd, (struct sockaddr*)&c
lientAddr, &addrLength);
        if (-1 == conn)
        {
            fprintf(stderr, "Server: I can't connect t
he client, is there something wrong?\n");
            continue;//ignore this client and continue
 the loop
        }
        printf("Server: Now we've got a new client[%d]
!A thread prepared for it is creating.\n", conn);
        pthread_t pid;
        pthread_t thread;
        //create a thread for the client to handle its
 requests
```

```
        if (pthread_create(&thread, NULL, clientHandle
Thread, &conn) != 0)//创建子线程
        {
            fprintf(stderr, "Server: There's somthing
wrong while creating the thread!\n");
            break;
        }
        //get the ip of the client to create a client
list
        Item a;
        a.conn = conn;
        strcpy(a.ip, inet_ntoa(clientAddr.sin_addr));
        a.port = ntohs(clientAddr.sin_port);
        clientList.push_back(a);
    }
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

  客户端：

```
+-------+------------------------------------
:c
Enter IP address:
127.0.0.1
Enter port number:
4341
connect successful
```

  服务端：

```
Server: Now we've got a new client[4]!A thread prepared fo
r it is creating.
Client[4]: Create thread successfully.
```

- 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。

  客户端：

```
:t
Server Time: Wed Dec 31 16:00:00 1969
```

  服务端：

```
Client[5]: Respond client request for time.
```

- 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。

  客户端：

```
:n
Host Name: ubuntu
```

  服务端：

```
Client[5]: Respond client request for hostname.
```

  相关的服务器的处理代码片段：

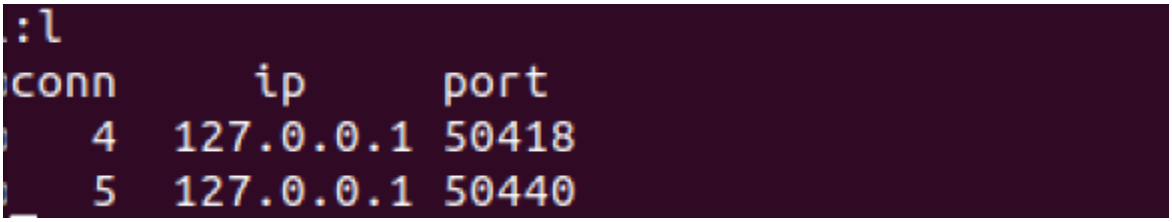
```
//analyse the message,decide use which function to use
 according to the format
int analyseMessage(int conn, char * data)
{
    time_t time;
    int i;
    char hostname[HOSTNAME_LENGTH];
```

```
    char message[MESSAGE_LENGTH];
    char temp[100];
    bool toClientExist = false;
    int toClientConn = -1;
    //message sent by client
    char clientMessage[MESSAGE_LENGTH];
    //Wrong format
    if (data == NULL || data[0] != ':' || data[1] == N
ULL)
    {
        return -1;
    }
    switch (data[1])
    {
          case 't':
           case 'n':
        //format: :n hostname
        strcpy(message, ":n ");
        gethostname(hostname, HOSTNAME_LENGTH);
        strcat(message, hostname);
        strcat(message,"\n");
        sendMessage(conn, message);
```

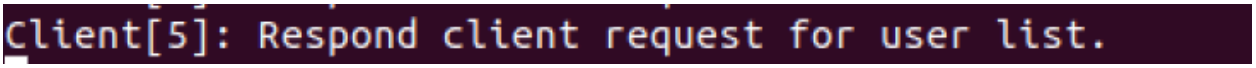
```
        printf("Client[%d]: Respond client request for
 hostname.\n", conn);
        break;


    case 'c':
    case 'm':
    }
    return 0;
}
```

- 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。

  客户端：



  服务端：



  相关的服务器的处理代码片段：

```
//analyse the message,decide use which function to use
 according to the format
int analyseMessage(int conn, char * data)
{
    time_t time;
```

```
    int i;
    char hostname[HOSTNAME_LENGTH];
    char message[MESSAGE_LENGTH];
    char temp[100];
    bool toClientExist = false;
    int toClientConn = -1;
    //message sent by client
    char clientMessage[MESSAGE_LENGTH];
    //Wrong format
    if (data == NULL || data[0] != ':' || data[1] == N
ULL)
    {
        return -1;
    }
    switch (data[1])
    {
    case 't':
    case 'n':
        //ask for client lists - traverse the client l
ist and send the id,IP,port data to this client.
    case 'c':
        strcpy(message, ":c ");
```

```
        i=0;
        for (vector<Item>::iterator it = clientList.be
gin(); it != clientList.end(); it++, i++)
        {
            //format :c conn|ip|port;conn|ip|port;...
            sprintf(temp, "%4d|%10s|%4d;", (*it).conn,
 (*it).ip, (*it).port);
            strcat(message, temp);
        }
        sendMessage(conn, message);
        printf("Client[%d]: Respond client request for
 user list.\n", conn);
        //send message - find whether the receiver cli
ent exists, if exists, send() the message to it and te
ll it the sender is this client.
        break;
    case 'm':
    }
    return 0;
}
```

- 客户端选择发送消息功能时，两个客户端和服务端（如果有的话）显示内容截图。

  发送消息的客户端：

```
:m
Enter receiver's conn:5
Enter send message:htt
Message: Transfer Success: The message has been successful
ly sent to Client[5]
```

接收消息的客户端：

```
Message: Message from Client[    6]: htt
```

相关的服务器的处理代码片段：

```
//analyse the message,decide use which function to use
 according to the format
int analyseMessage(int conn, char * data)
{
    time_t time;
    int i;
    char hostname[HOSTNAME_LENGTH];
    char message[MESSAGE_LENGTH];
    char temp[100];
    bool toClientExist = false;
    int toClientConn = -1;
    //message sent by client
    char clientMessage[MESSAGE_LENGTH];
    //Wrong format
    if (data == NULL || data[0] != ':' || data[1] == N
ULL)
```

```
    {
        return -1;
    }
    switch (data[1])
    {
    case 't':
    case 'n':
    case 'c':
    case 'm':
        //get the toClientConn
        sscanf(data + 3, "%d", &toClientConn);
        //get the message from client
        for (i = 3; i < strlen(data); i++)
        {
            if (data[i] == ' ')break;
        }
        sprintf(clientMessage,":m Message from Client[
%5d]:",conn);
        strcat(clientMessage, data + i);
        //find whether the toClient exists
        for (vector<Item>::iterator it = clientList.be
gin(); it != clientList.end(); it++)
```

```
        {
            if ((*it).conn == toClientConn)
            {
                toClientExist = true;
                break;
            }
        }
        if (toClientExist = false)
        {
            sprintf(message, ":m Transfer Fail: Client
[%d] you want to send message to is not on this server
.Please check the conn.\n", toClientConn);
            sendMessage(conn, message);
        }
        else
        {

            if (-1 == sendMessage(toClientConn, client
Message))
            {
```

```
                sprintf(message, ":m Transfer Fail: So
mething happens when sending message to Client[%d],ple
ase try again.\n", toClientConn);
                sendMessage(conn, message);
            }
            else
            {
                sprintf(message, ":m Transfer Success:
 The message has been successfully sent to Client[%d]\
n", toClientConn);
                sendMessage(conn, message);
            }

        }
    }
    return 0;
}
```

相关的客户端（发送和接收消息）处理代码片段：

```
int main()
{
    bool state = false;
    char instruction[3];
```

```
    char name[NAME_LENGTH];
    initialize();
    while (true)
    {
        scanf("%s", instruction);
        if (instruction[0] != ':')
        {
            printf("Instruction format error\n");
            continue;
        }
        //on the unconnected state, only connect and q
uit can be choosed.
        if (!state)
        {
            switch (instruction[1])
            {
            case 'c':
                state = startConnect();
                break;
            case 'q':
                quit = true;
                printf("Client server quit...\n");
```

```
                pthread_exit(NULL);
            default:
                printf("Can't choice before connect!\n
");
                break;
            }
        }
        else
        {
            char msg[BUFFER_SIZE] = {0};
            switch (instruction[1])
            {
            case 'c':
                state = startConnect();
                sprintf(msg, "::");
                break;
            case 'd':
                state = quitConnect();
                sprintf(msg, "::");
                break;
            case 't':
                sprintf(msg, ":t");
```

```
                break;
            case 'n':
                sprintf(msg, ":n");
                break;
            case 'l':
                sprintf(msg, ":c");
                break;
            case 'm':
                char recvConn[BUFFER_SIZE];
                char sendMessage[BUFFER_SIZE];

                printf("Enter receiver's conn:");
                scanf("%s", recvConn);
                printf("Enter send message:");
                scanf("%s", sendMessage);
                sprintf(msg, ":m %s %s", recvConn, sen
dMessage);
                break;
            case 'q':
                state = quitConnect(); //close socket
                quit = true;           //quit receive
thread
```

```
                pthread_exit(NULL);     //quit send thr
ead
            default:
                sprintf(msg, "::");
                break;
            }
            if (send(serverSocketFD, msg, BUFFER_SIZE,
 0) == -1)
                printf("Send fail");
        }
    }
}
```

循环接收用户输入的指令，有连接、获得时间、列出用户、发送数据、退出连接功能。

六、 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？

答：不需要。

客户端向服务器发送请求时，服务器随机分配一个未被占用的端口给客户端。

不会保持不变，因为每一次 connect 请求时服务器都会**随机**分配一个空闲端口，保持不变的概率较小。

- 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：可以。

先说明三个函数的作用：listen 函数将套接字(sockfd)变成被动的连接监听套接字（被动等待客户端的连接），accept 函数从处于监听状态的流套接字 s 的客户连接请求队列中取出排在最前的一个客户请求，并且创建一个新的套接字来与客户套接字创建连接通道。connect 函数为客户端主动连接服务器，通过三次握手建立连接。需要注意的是，连接的过程是由内核完成而不是这个函数完成的，函数的作用是通知 Linux 内核，让 Linux 内核自动完成 TCP 三次握手连接。

当有一个客户端主动连接（connect()），Linux 内核就自动完成 TCP 三次握手，将建立好的链接自动存储到队列中，所以只要 TCP 服务器调用了 listen()，客户端就可以通过 connect() 和服务器建立连接，而这个连接的过程是由内核完成。

- 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：

不同应用进程间的网络通信和连接可以通过 3 个参数来区分：通信的目的 IP 地址、使用的传输层协议(TCP 或 UDP)和使用的端口号,Socket 唯一绑定三个参数,client 客户端在发送时传递的 sockfd 参数可以用来区分。

- 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：

如果客户端主动断开，在应用程序中调用 close，于是其 TCP 会发出 FIN M 请求主动关闭连接，然后客户端 TCP 进入 PIN_WAIT1 状态。当服务端接收到 FIN M 后，执行被动关闭，对接收到的 FIN M 进行确认，返回客户端 ACK。随后服务端进入 CLOSE_WAIT 状态，客户端进入 FIN_WAIT 状态。FIN 的接收意味着应用进程在该连接上无法再接受数据，于是也会被作为文件结束符传递给应用进程。一段时间后，服务端检测到客户端关闭操作，接收到文件结束符，调用 close 关闭 socket。这导致服务端的 TCP 发送 FIN N。客户端接收到这个 FIN N 后，客服端的 TCP 进入 TIME_WAIT 状态，向服务端再次发送 ACK 确认，2MSL 后进入 CLOSE 状态。

- 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：

  客户端断网后异常退出，就说明客户端未主动发送 close 就断开连接，即客户端发送的 FIN 丢失或未发送。服务端会维持 ESTABLISHED 状态，但此时的连接是一个“死连接”。可以通过心跳机制检测客户端和服务端的连接是否是死连接。带心跳机制的客户端，以一定频率发送报文段不含有任何数据的数据包。服务端用 recv 函数不会接收到值，但是会阻塞等待。心跳包发送的频率参数可以调节。服务端接收不到心跳包就可以判断客户端已经异常退出。

七、 讨论、心得

我在实验中完成的是客户端程序的编写。网络层用 IP 地址标识网络中的主机，传输中的“协议+端口”标识主机中的进程。Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层。客户端首先需要申请一个 socket 套接字，然后和服务端进行连接。连通后，就可以和服务端通信，发送和接收信息。起先将发送信息也作为一个子进程处理，然后发现是错误的做法，更改为用户选择发送信息时再发送。而接收信息是单独的子进程，一直在运行，直到主进程退出。