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：	蔡佳伟
学　　院：	计算机科学与技术学院
系：	
专　　业：	软件工程
学　　号：	3220104519
指导教师：	高艺

2024 年 12 月 21 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：无　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

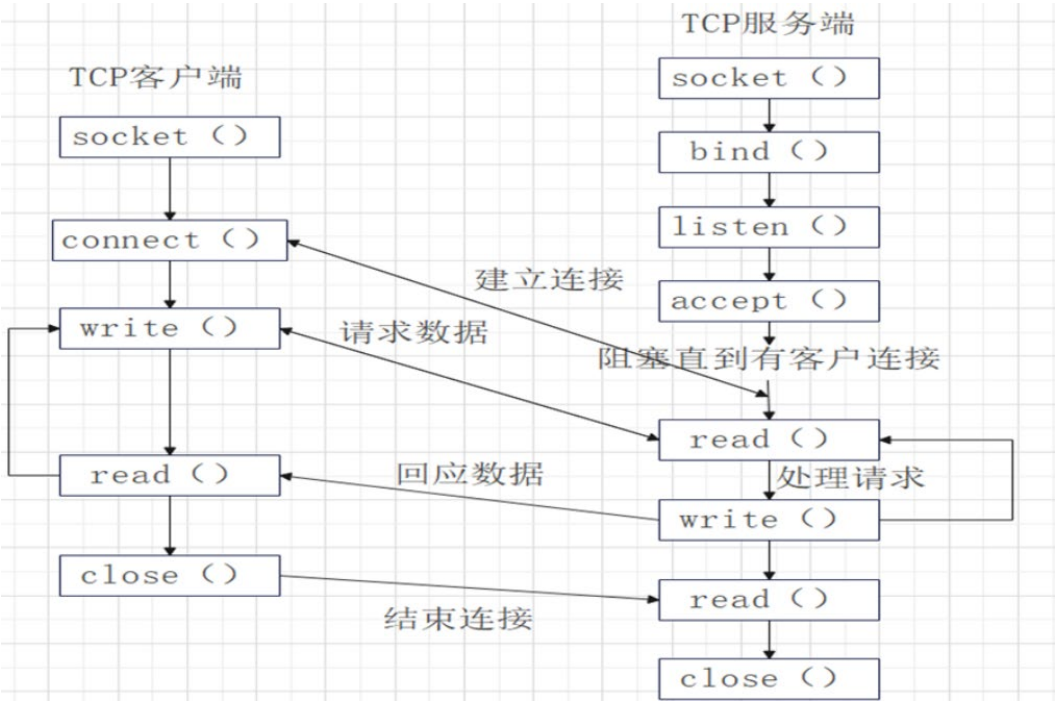
请将以下内容和本实验报告一起打包成一个压缩文件上传：

- **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**

上传的代码在 socket 目录下，为 client.c 和 socket.c。我是在 linux 环境下操作的。在 socket 目录下，使用 gcc server.c -o server –lpthread 命令可以生成 server 可执行文件，使用 gcc client.c -o client -lpthread –pthread 命令可以生成 client 文件，然后使用 ./server 命令可以运行服务器，使用./client 命令可以运行客户端。我也一并上传了可执行文件。

以下实验记录均需结合屏幕截图（截取源代码或运行结果），进行文字标注（看完请删除本句）。

- 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

```
Connection success!! Port: 4519
This program is developed by Jiawei Cai, 3220104519,
College of Computer Science and Technology, Zhejiang University.
This is a simple socket program.

Please choose an option:
1) Connect to server
2) Disconnect from server
3) Get server time
4) Get server name
5) Get client list
6) Send message to client
7) Exit
Enter your choice: 4
Not connected to any server.

Please choose an option:
1) Connect to server
2) Disconnect from server
3) Get server time
4) Get server name
5) Get client list
6) Send message to client
7) Exit
Enter your choice:
```

- 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）

```c
static void Data_handle(void *sock_fd)
{
    char data_recv[BUFFER_LENGTH];
    int i_recvBytes, fd = *((int *)sock_fd);

    while (1)
    {
        memset(data_recv, 0, BUFFER_LENGTH);

        i_recvBytes = recv(fd, data_recv, BUFFER_LENGTH, 0);

        if (i_recvBytes > 0)
        {
            if (data_recv[0] == '#')
            {
                printf("\nThe data from another client:\n%s\n", data_recv);
            }
            else
            {
                printf("The information from the server:\n%s\n", data_recv);
            }
        }

        if (strcmp(data_recv, "connection closed") == 0)
        {
            break;
        }
    }
```

```c
    int ret = shutdown(fd, SHUT_WR);
    assert(ret != -1);

    exit(0);
}
```

这个函数在一个独立的线程中运行，负责从服务器或其他客户端接收数据

进入一个无限循环，通过 recv 函数接收数据

根据接收到的数据，判断数据内容并打印输出

如果接收到的消息是 connection closed，则退出循环，表示连接已经关闭

- 服务器初始运行后显示的界面

```
cjwjiwang@cjwjiwang-virtual-machine:~/socket$ ./server
waiting for new connection...
A new connection occurs!
waiting for new connection...
waiting for request...
```

服务器初始运行成功

- 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）

```
ver.c > ...
static void Data_handle(void *sock_fd)
    while (1)
        if (data_recv[0] == '#' && data_recv[1] == '#' && data_recv[2] == '#' && data_recv[3] ==
            printf("The data sent:\n%s", data_send);
        }
        else
        {
            int target_number = 0, source_number = 0;
            int index = 0;
            for (;; index++)
            {
                if (data_recv[index] != ' ')
                {
                    target_number *= 10;
                    target_number += data_recv[index] - '0';
                }
                else
                {
                    break;
                }
            }
            printf("The information that client want to say to client number %d is:\n%s\n", targ

            for (int i = 0; i < connection_number; i++)
            {
                if (fd == client_list[i].fd)
                {
                    source_number = client_list[i].number;
                    break;
                }
            }
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

```
Please choose an option:
1) Connect to server
2) Disconnect from server
3) Get server time
4) Get server name
5) Get client list
6) Send message to client
7) Exit
Enter your choice: 1
Please input the server IP address: 127.0.0.1
Please input the server port: 4519
Connected to server 127.0.0.1:4519
```

```
cjwjiwang@cjwjiwang-virtual-machine:~/socket$ ./server
waiting for new connection...
A new connection occurs!
waiting for new connection...
waiting for request...
```

客户端和服务器成功建立连接

- 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。

```
2) Disconnect from server
3) Get server time
4) Get server name
5) Get client list
6) Send message to client
7) Exit
Enter your choice: The information from the server:
now datetime: 2025-1-2 22:10:36
```

```
read from client : #### get time ####
The data sent:
now datetime: 2024-12-21 14:55:17
waiting for request...
```

选择时间功能后，客户端和服务器均显示时间

- 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。

```
4

Please choose an option:
1) Connect to server
2) Disconnect from server
3) Get server time
4) Get server name
5) Get client list
6) Send message to client
7) Exit
Enter your choice: The information from the server:
The server host name: weichen-VMware-Virtual-Platform
```

```
read from client : #### get name ####
cjwjiwang-virtual-machineThe data sent:
The server host name: cjwjiwang-virtual-machinewaiting for reque
st...
```

选择获取名字功能后，客户端和服务器均显示 server host name

相关的服务器的处理代码片段：

```
else if (strcmp(infomation, "name") == 0)
{
    char host_name[1024] = {0};
    gethostname(host_name, 1024);
    sprintf(data_send, "The server host name: %s", host_name);
    printf("%s", host_name);
}
```

当接收到 get name 请求时，服务器通过 gethostname 函数获取服务器的主机名，然后将主机名格式化并发送给客户端

- 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。

```
5

Please choose an option:
1) Connect to server
2) Disconnect from server
3) Get server time
The information from the server:
tid                    ip      port    number
1677723328      127.0.0.1       42769   0
And your client's number is: 0

4) Get server name
5) Get client list
6) Send message to client
7) Exit
Enter your choice:
```

```
read from client : #### get list ####
The data sent:
tid                    ip      port    number
2864711232      127.0.0.1       42769   0
And your client's number is: 0
waiting for request...
```

选择获取列表后，客户端和服务器均显示客户端列表

相关的服务器的处理代码片段：

```
else
{
    int current_number = 0;
    strcpy(data_send, "tid\t\t\tip\tport\tnumber\n");

    for (int i = 0; i < connection_number; i++)
    {
        char temp[50] = {0};
        sprintf(temp, "%u\t%s\t%d\t%d\n", client_list[i].tid, client_list[i].ip, client_list[i].port, client_list[i].number);
        strcat(data_send, temp);

        if (client_list[i].tid == pthread_self())
        {
            current_number = client_list[i].number;
        }
    }

    char temp[50];
    sprintf(temp, "And your client's number is: %d\n", current_number);
    strcat(data_send, temp);
}
```

服务器遍历所有连接的客户端并将每个客户端的信息（线程 ID，IP，端口，客户端编号）添加到返回的字符串中

最后，还会将当前客户端的编号添加到返回的信息中，该信息通过 send 返回给客户端

- 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：

```
Now please input the number of command: 2
Please input the client's number you want to: 1
Please input something you want to send to the server:
cjwsocket
The information from the server:
The server has received your message.
```

服务器端（可选）：

```
waiting for new connection...
waiting for request...
read from client : 1 send cjwsocket
The information that client want to say to client number 1 is:
cjwsocket
waiting for request...
```

接收消息的客户端：

```
The data from another client:
#### The client number 0 sends you a message:
cjwsocket
```

客户端 0 发送消息给客户端 1 后，服务器看到了客户端 0 发送的消息和目的地，客户端 1 也收到了客户端 0 发送的消息

相关的服务器的处理代码片段：

```
else
{
    int target_number = 0, source_number = 0;
    int index = 0;
    for (;; index++)
    {
        if (data_recv[index] != ' ')
        {
            target_number *= 10;
            target_number += data_recv[index] - '0';
        }
        else
        {
            break;
        }
    }
    printf("The information that client want to say to client number %d is:\n%s\n", target_number, data_recv + index + 6);

    for (int i = 0; i < connection_number; i++)
    {
        if (fd == client_list[i].fd)
        {
            source_number = client_list[i].number;
            break;
        }
    }
```

```
char temp[BUFFER_LENGTH] = {0};
sprintf(temp, "#### The client number %d sends you a message:\n%s", source_number, data_recv + index + 6);
if (send(client_list[target_number].fd, temp, strlen(temp), 0) == -1)
{
    strcpy(data_send, "Sending error!\n");
}
else
{
    strcpy(data_send, "The server has received your message.\n");
}
```

服务器首先解析客户端发送的消息，获取目标客户端的编号和消息内容，然后遍历客户端列表，将消息发送给目标客户端。

如果消息发送成功，服务器会通知客户端“消息已被接收”，如果发送失败，会返回发送错误。

相关的客户端（发送和接收消息）处理代码片段：

发送消息：

```
else if (choice == 2)
{
    int client_number;
    printf("Please input the client's number you want to: ");
    scanf("%d", &client_number);
    printf("Please input something you want to send to the server:\n");
    getchar();
    char temp[BUFFER_LENGTH - 5] = {0};
    fgets(temp, sizeof(temp), stdin);
    temp[strcspn(temp, "\n")] = '\0';
    snprintf(data_send, sizeof(data_send)+sizeof(char)*12, "%d send %s", client_number, temp);
}
```

用户选择 2 来发送消息时，首先输入目标客户端的编号 client_number，然后输入要发送的消息内容，并存储在 temp 字符数组中。

最后 snprintf 格式化并准备将消息发送到服务器，消息的格式是：客户端编号 send 消息内容

接收消息：

```
if (i_recvBytes > 0)
{
    if (data_recv[0] == '#')
    {
        printf("\nThe data from another client:\n%s\n", data_recv);
    }
    else
    {
        printf("The information from the server:\n%s\n", data_recv);
    }
}
```

在接收到数据后，首先判断数据的内容，如果消息的第一个字符是#，则认为是其他客户端发送过来的数据，打印，否则认为是服务器发来的消息，打印服务器信息。

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？

  TCP 模式下，客户端不调用 bind，当没有绑定端口时，操作系统会自动为客户端分配一个源端口，每次调用 connect 时，客户端的端口通常会改变。

  UDP 模式下，客户端同样不调用 bind，当没有绑定端口时，操作系统也会自动为客户端分配一个源端口。在这种情况下，客户端多次发送 UDP 数据包时，端口通常保持不变，直到该套接字关闭为止。

- 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？

  连接不会立即成功。Connect 的成功取决于服务端在 accept 状态下监听连接请求。如果服务端在调用 accept 之前暂时停止，客户端会进入阻塞状态，等待服务端的相响应。只有当服务端继续执行 accept 后，连接才会完成。

- 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？

  可以通过 IP 地址和端口号区分。可以看到上面的例子是有信息的。

```
waiting for new connection...
waiting for request...
read from client : 1 send cjwsocket
The information that client want to say to client number 1 is:
cjwsocket
waiting for request...
```

- 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）

  TCP 连接状态会进入 FIN_WAIT_1 状态，然后经过 FIN_WAIT_2 和 TIME_WAIT 状态，最终完全关闭连接。在接收到客户端的 FIN 后，服务器进入 CLOSE_WAIT 状态，这个状态保持时间大约为一分钟。

- 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？

如果客户端断网后异常退出，服务器的 TCP 连接状态会先保持不变，但它会定时检测连接状态，如果客户端没有回应服务器，TCP 连接状态就会设置为断开。

七、 讨论、心得

本次实验一开始还是比较陌生的，在晚上学习了一下 socket 的写法，用底层的 c 语言来实现。在 debug 的过程中，要确保每个客户端连接都有独立线程的实现，很有难度。我大量使用了 print 中间变量的方法来完成这个实验。还有 snprintf 这句话一直都有 bug，也修改了很长时间。总而言之，这个实验让我明确了 Socket 编程接口编写基本的网络应用软件的基本操作，作为最后一个实验收获还是不小的。