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：	秦嘉俊
学　　院：	计算机学院
系：	
专　　业：	计算机科学与技术
学　　号：	3210106182
指导教师：	许海涛

2023 年 12 月 29 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：　　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

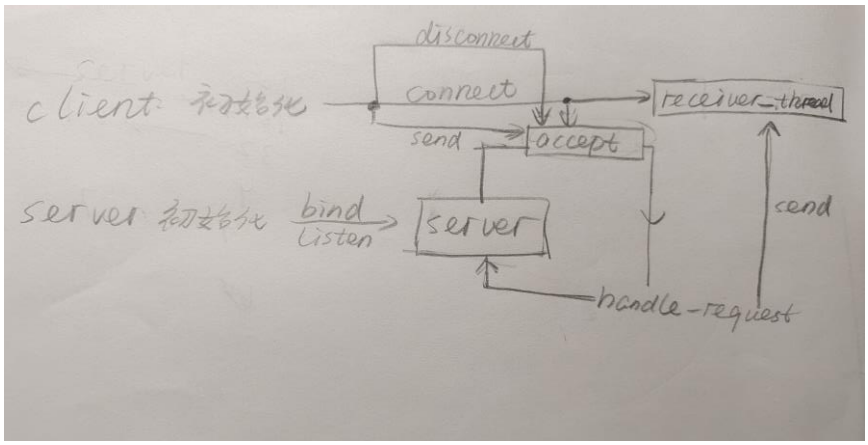
五、 实验数据记录和处理

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：

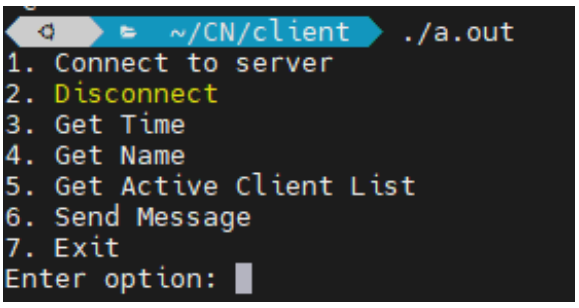
- **源代码：客户端和服务端的代码分别在一个目录**
- **可执行文件：可运行的.exe 文件或 Linux 可执行文件，客户端和服务端各一个**

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



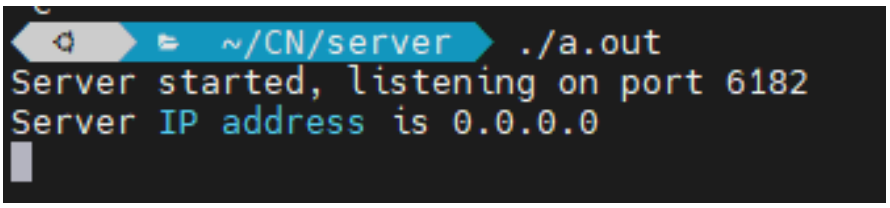
- 客户端的接收数据子线程循环关键代码截图（描述总体，省略细节部分）

  在成功连接到服务器之后，我们会通过 detach() 这个方法新建一个线程，这个子线程就负责执行下面的 receive_data() 函数，不断地接收服务器的数据。

这个函数里只要我们处于连接状态，就尝试从 sock 读取数据并放到 buffer 中，如果有数据被传过来就进行输出，一直循环，直到连接断开。

```
// 客户端的接收数据子线程循环
void receive_data() {
    char buffer[1024] = {0};
    while (connected) {
        int valread = read(sock, buffer, 1024);
        if (valread > 0) {
            lock_guard<mutex> lock(cout_mutex);
            cout << endl << "Server Message:\n" << buffer << endl;
            memset(buffer, 0, sizeof(buffer));
        }
    }
}
```

- 服务器初始运行后显示的界面



这里监听的端口为 6182，即学号后四位。

输出的 ip addr 为 0.0.0.0，表示本机的 IP 地址。

- 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）

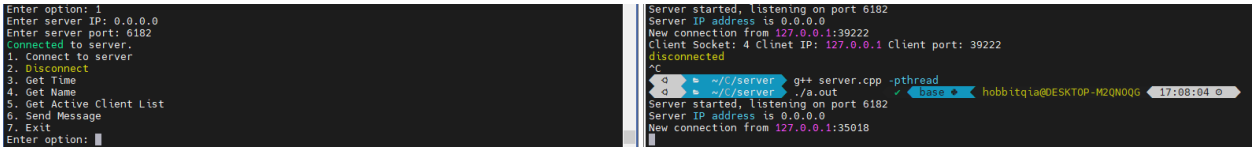
在服务端，我们已经成功建立连接后，就通过 detach() 新建一个子线程，调用 handle_client() 来处理用户的请求。

每次循环都尝试从 client_socket 读取数据并放到 buffer 中，随后根据请求的类型交由不同的函数具体处理。处理后要清空缓冲区。

客户端断开连接后，需要关闭 client_socket，并将其从我们服务端用来记录处于连接状态的客户信息删除掉（a 用来记录客户端信息，client_sockets 用来记录每个用户端的 socket 号）

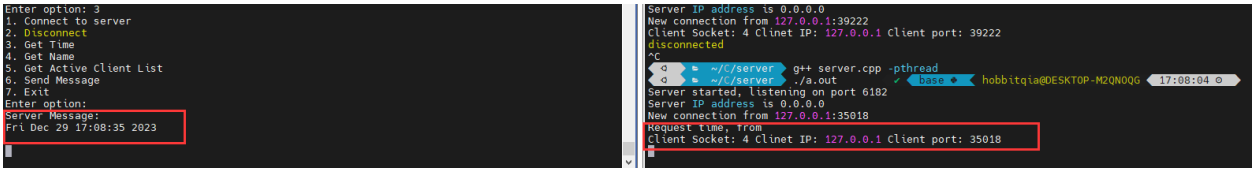
```cpp
// 服务器的客户端处理子线程循环
void handle_client(int client_socket) {
    char buffer[1024] = {0};

    while (true) {
        int valread = read(client_socket, buffer, 1024);
        if (valread <= 0) {
            break; // 客户端断开连接
        }

        string request(buffer);
        if (request == "time") send_current_time(client_socket);
        else if (request == "name") send_server_name(client_socket);
        else if (request == "clients") send_client_list(client_socket);
        else if (request.substr(0, 3) == "msg") {
            string msg = request.substr(4);
            int target_socket = buffer[3] - '0';
            msg += "\nMessage from socket: " + to_string(client_socket);
            send_client_msg(target_socket, client_socket, msg);
        }
        else send(client_socket, "Invalid request", 15, 0);

        memset(buffer, 0, sizeof(buffer));
    }

    // 客户端断开连接
    close(client_socket);
    clients_mutex.lock();
    client_sockets.erase(remove(client_sockets.begin(), client_sockets.end(), client_socket), client_sockets.end());
    clients_mutex.unlock();

    cout_mutex.lock();
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (a[i].socket == client_socket) {
            cout << "Client Socket: " << a[i].socket << " Clinet IP: "<< a[i].ip << " Client port: " << a[i].port << endl;
            cout << "disconnected" << endl;
            a[i].socket = -1;
            break;
        }
    }

    cout_mutex.unlock();
}
```

- 客户端选择连接功能时，客户端和服务端显示内容截图。

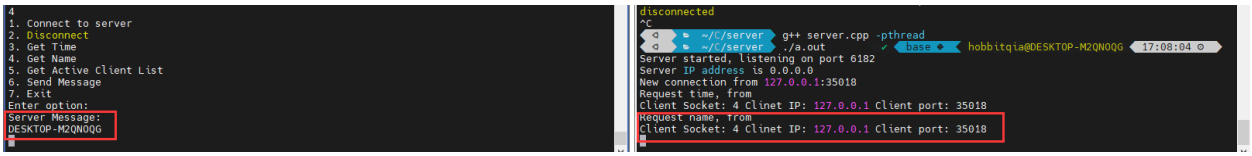


左为客户端，右为服务端。（下同，这里不再重复）

- 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。



- 客户端选择获取名字功能时，客户端和服务端显示内容截图。

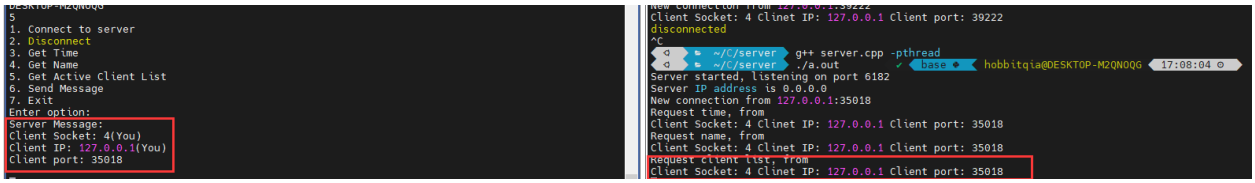


相关的服务器的处理代码片段：

这里我们首先输出提示信息，随后通过 gethostname 获得主机名称，并返回给客户端。、

```
void send_server_name(int client_socket) {
    cout << "Request name, from " << endl;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (a[i].socket == client_socket) {
            cout << "Client Socket: " << a[i].socket << " Clinet IP: "<< a[i].ip << " Client port: " << a[i].port << endl;
            break;
        }
    }
    char hostname[256];
    gethostname(hostname, sizeof(hostname));
    send(client_socket, hostname, strlen(hostname), 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。

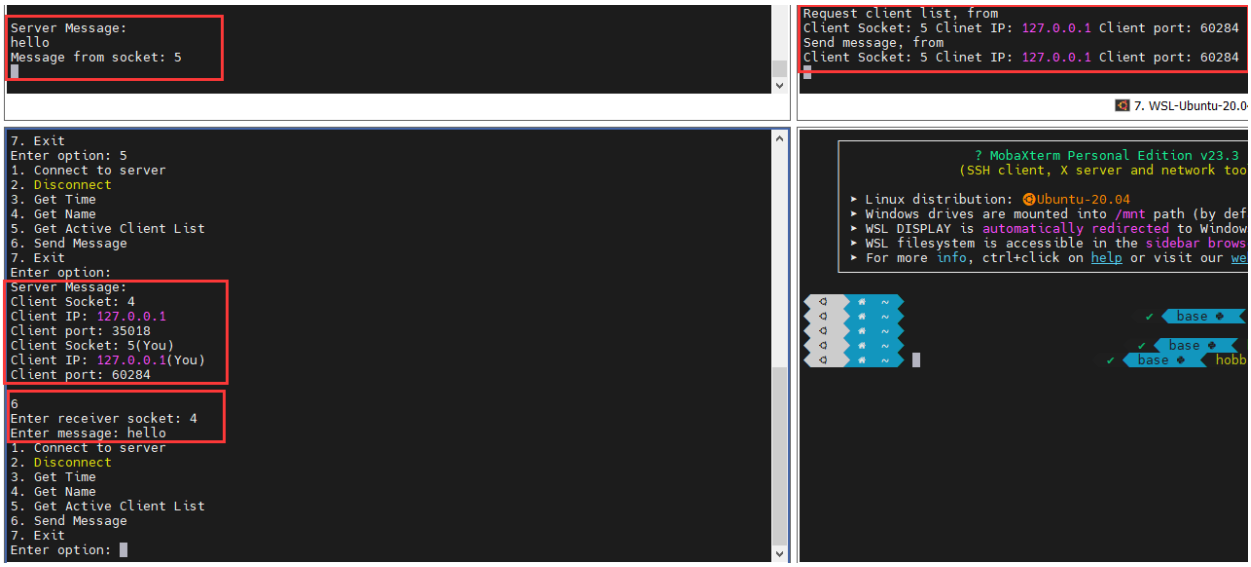


相关的服务器的处理代码片段：

这里我们先输出提示信息，随后从 client_sockets 中找到目前连接在服务器上的 socket 句柄，并从用户信息结构体数组（a）中找到对应的结构体，随后记录信息，最后发送回客户端即可。

```
void send_client_list(int client_socket) {
    cout << "Request client list, from " << endl;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (a[i].socket == client_socket) {
            cout << "Client Socket: " << a[i].socket << " Clinet IP: "<< a[i].ip << " Client port: " << a[i].port << endl;
            break;
        }
    }
    lock_guard<mutex> lock(clients_mutex);
    string clients_info;
    for (int socket : client_sockets) {
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (a[i].socket == socket) {
                clients_info += "Client Socket: " + to_string(socket);
                if (socket == client_socket) clients_info += "(You)\n";
                else clients_info += "\n";
                clients_info += "Client IP: " + a[i].ip;
                if (socket == client_socket) clients_info += "(You)\n";
                else clients_info += "\n";
                clients_info += "Client port: " + to_string(a[i].port) + "\n";
                break;
            }
        }
    }
    send(client_socket, clients_info.c_str(), clients_info.length(), 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：

服务器端（可选）：

接收消息的客户端：

左上角（socket=4）为接收消息的客户端，右上角为服务端，左下角为发送消息的客户端（socket=5）

这里我们在发送消息之前，先查看了活跃的用户列表，随后再发送。

相关的服务器的处理代码片段：

根据客户端传过来的 msg，提取出来要发送到的客户端。随后添加提示信息（即消息是从哪个客户发过来的）并在服务端也输出提示信息，随后直接发送即可。

```
else if (request.substr(0, 3) == "msg") {
    string msg = request.substr(4);
    int target_socket = buffer[3] - '0';
    msg += "\nMessage from socket: " + to_string(client_socket);
    send_client_msg(target_socket, client_socket, msg);
}
```

```
void send_client_msg(int client_socket, int from_socket, const string& msg) {
    cout << "Send message, from " << endl;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (a[i].socket == from_socket) {
            cout << "Client Socket: " << a[i].socket << " Clinet IP: "<< a[i].ip << " Client port: " << a[i].port << endl;
            break;
        }
    }
    lock_guard<mutex> lock(clients_mutex);
    string clients_info = msg;
    send(client_socket, clients_info.c_str(), clients_info.length(), 0);
}
```

（再 send_client_msg 中 client_socket 是接收者，from_socket 是发送者）

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 (option == 6) {
    cout << "Enter receiver socket: ";
    cin.ignore();
    string receiver;
    getline(cin, receiver);
    cout << "Enter message: ";
    string msg;
    getline(cin, msg);
    msg = "msg" + receiver + msg;
    send_message(msg);
}
```

```
// 发送消息到服务器
void send_message(const string& message) {
    if (connected) {
        send(sock, message.c_str(), message.size(), 0);
    } else {
        cout << "Not connected to the server" << endl;
    }
}
```

接收消息部分同前 receive_data()函数

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？

  不需要。

  其端口是在连接服务器端时由操作系统随机分配的，分配时要与已经分配的端口不同。

  每一次均会变化。

- 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？

  可以。服务端调用 listen 后 connect 函数就可以正常返回，此时客户端的连接处于半连接状态，也会输出成功连接的提示信息。

- 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？

struct sockaddr_in address;其中 sockaddr_in 里有协议族、IP 地址、端口号这三个成员变量，可以唯一标识一个客户端。

- 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）

  上为连接时，下为某个客户端断开后。

```
 ~ # netstat -an | grep 6182
tcp        0      0 0.0.0.0:6182            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:40934         127.0.0.1:6182          ESTABLISHED
tcp        0      0 127.0.0.1:41590         127.0.0.1:6182          ESTABLISHED
tcp        0      0 127.0.0.1:6182          127.0.0.1:41590         ESTABLISHED
tcp        0      0 127.0.0.1:6182          127.0.0.1:40934         ESTABLISHED
unix  2      [ ]         DGRAM      CONNECTED     1461829
 ~ # netstat -an | grep 6182
tcp        0      0 0.0.0.0:6182            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:40934         127.0.0.1:6182          ESTABLISHED
tcp        0      0 127.0.0.1:41590         127.0.0.1:6182          TIME_WAIT
tcp        0      0 127.0.0.1:6182          127.0.0.1:40934         ESTABLISHED
unix  2      [ ]         DGRAM      CONNECTED     1461829
```

  这个状态会一直持续，直到服务器关闭。

- 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？

TCP 连接状态会变为上图的状态 （即从 ESTABLISHED 变为 TIME_WAIT）服务器可以通过向客户端发送消息来检查连接是否还有效。

七、 讨论、心得

本次实验是计网最后一个实验，相比之前的确更容易了许多。VSCode 的界面就是比 GNS3 的界面亲切（确信），而且本身内容也和课程本身结合更紧密，体验还是可以的。这里也非常感谢老师和助教姐姐去掉了本来的 GNS3 实验~~（永别了！牢笼）~~。每当我看到其他班的同学因为还剩好几个计网实验而焦头烂额，我就突然释怀的笑。“你们有这样的老师和助教吗？！”伟大，无需多言！

这学期的学习也要进入尾声了，感谢助教姐姐的帮助。难得相遇，终要别离。江湖路远，后会有期！