一、网络聊天程序的设计与实现

课程设计题目

了解Socket通信的原理，在此基础上编写一个聊题程序。

课程设计内容

在计算机通信领域，socket 被翻译为“套接字”，它是计算机之间进行通信的一种约定或一种方式。通过 socket 这种约定，一台计算机可以接收其他计算机的数据，也可以向其他计算机发送数据。

socket 的典型应用就是 Web 服务器和浏览器：浏览器获取用户输入的URL，向服务器发起请求，服务器分析接收到的URL，将对应的网页内容返回给浏览器，浏览器再经过解析和渲染，就将文字、图片、视频等元素呈现给用户。

TCP /IP协议包含的范围非常的广，它是一种四层协议，包含了各种硬件、软件需求的定义。TCP /IP协议确切的说法应该是TCP/UDP/IP协议。UDP协议是一种保护消息边界的，不保障可靠数据的传输。TCP协议是一种流传输的协议。它提供可靠的、有序的、双向的、面向链接的传输。

保护消息边界，就是指传输协议把数据当作一条独立的消息在网上传输，接收端只能接收独立的消息。也就是说存在保护消息边界，接收端一次只能接收发送端发出的一个数据包。

而面向流则是指无保护消息边界的，如果发送端连续发送数据，接收端有可能在以此接收动作中，会接收两个或者更多的数据包。

课程设计步骤

注意：以下代码均在**Linux**下运行，与**Windows**不兼容。

Server.cpp:

1. 通过 socket() 函数创建一个套接字，参数 AF_INET 表示使用 IPv4 地址，SOCK_STREAM 表示使用面向连接的数据传输方式，IPPROTO_TCP 表示使用 TCP 协议。在 Linux 中，socket 也是一种文件，有文件描述符，可以使用 write() / read() 函数进行 I/O 操作。
2. 通过 bind() 函数将套接字 serv_sock 与特定的IP地址和端口绑定，IP地址和端口都保存在 sockaddr_in 结构体中。
3. socket() 函数确定了套接字的各种属性，bind() 函数让套接字与特定的IP地址和端口对应起来，这样客户端才能连接到该套接字。
4. 套接字处于被动监听状态。所谓被动监听，是指套接字一直处于“睡眠”中，直到客户端发起请求才会被“唤醒”。
5. accept() 函数用来接收客户端的请求。程序一旦执行到 accept() 就会被阻塞（暂停运行），直到客户端发起请求。
6. write() 函数用来向套接字文件中写入数据，也就是向客户端发送数据。
7. 和普通文件一样，socket 在使用完毕后也要用 close() 关闭

Client.cpp:

`Client.cpp`代码中除了以下两处其余部分与`Server.cpp`大致相同。

1. 通过 connect() 向服务器发起请求，服务器的IP地址和端口号保存在 sockaddr_in 结构体中。直到服务器传回数据后，connect() 才运行结束。
2. 通过 read() 从套接字文件中读取数据。

课程设计结果及结果分析 心得体会

IP地址（IP Address）

计算机分布在世界各地，要想和它们通信，必须要知道确切的位置。确定计算机位置的方式有多种，IP 地址是最常用的，例如，114.114.114.114 是国内第一个、全球第三个开放的 DNS 服务地址，127.0.0.1 是本机地址。

其实，我们的计算机并不知道 IP 地址对应的地理位置，当要通信时，只是将 IP 地址封装到要发送的数据包中，交给路由器去处理。路由器有非常智能和高效的算法，很快就会找到目标计算机，并将数据包传递给它，完成一次单向通信。

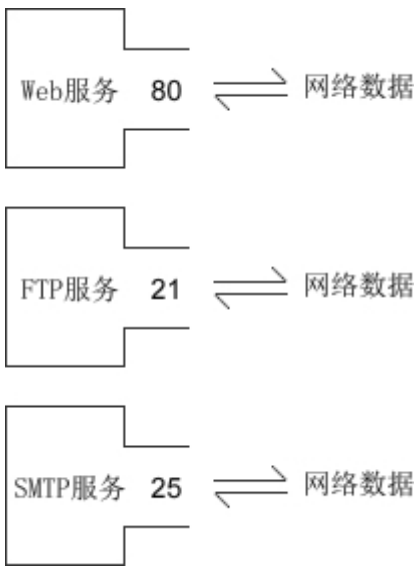
目前大部分软件使用 IPv4 地址，但 IPv6 也正在被人们接受，尤其是在教育网中，已经大量使用。

端口（Port）

有了 IP 地址，虽然可以找到目标计算机，但仍然不能进行通信。一台计算机可以同时提供多种网络服务，例如 Web服务、FTP服务（文件传输服务）、SMTP服务（邮箱服务）等，仅有 IP 地址，计算机虽然可以正确接收到数据包，但是却不知道要将数据包交给哪个网络程序来处理，所以通信失败。

为了区分不同的网络程序，计算机会为每个网络程序分配一个独一无二的端口号（Port Number），例如，Web服务的端口号是 80，FTP 服务的端口号是 21，SMTP 服务的端口号是 25。

端口（Port）是一个虚拟的、逻辑上的概念。可以将端口理解为一道门，数据通过这道门流入流出，每道门有不同的编号，就是端口号。如下图所示：



协议（Protocol）

协议（Protocol）就是网络通信的约定，通信的双方必须都遵守才能正常收发数据。协议有很多种，例如 TCP、UDP、IP 等，通信的双方必须使用同一协议才能通信。协议是一种规范，由计算机组织制定，规定了很多细节，例如，如何建立连接，如何相互识别等。

> 协议仅仅是一种规范，必须由计算机软件来实现。例如 IP 协议规定了如何找到目标计算机，那么各个开发商在开发自己的软件时就必须遵守该协议，不能另起炉灶。

所谓协议族（Protocol Family），就是一组协议（多个协议）的统称。最常用的是 TCP/IP 协议族，它包含了 TCP、IP、UDP、Telnet、FTP、SMTP 等上百个互为关联的协议，由于 TCP、IP 是两种常用的底层协议，所以把它们统称为 TCP/IP 协议族。

数据传输方式

计算机之间有很多数据传输方式，各有优缺点，常用的有两种：SOCK_STREAM 和 SOCK_DGRAM。

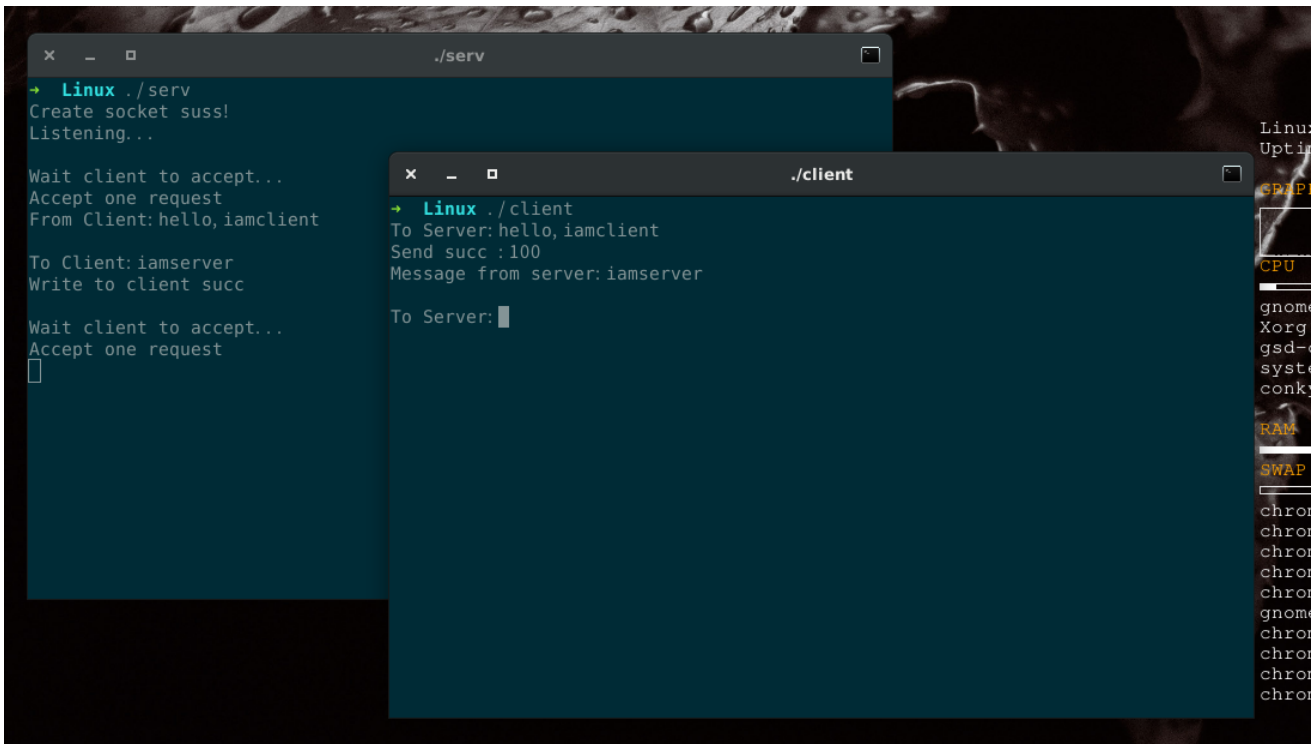
1) SOCK_STREAM 表示面向连接的数据传输方式。数据可以准确无误地到达另一台计算机，如果损坏或丢失，可以重新发送，但效率相对较慢。常见的 http 协议就使用 SOCK_STREAM 传输数据，因为要确保数据的正确性，否则网页不能正常解析。

2) SOCK_DGRAM 表示无连接的数据传输方式。计算机只管传输数据，不作数据校验，如果数据在传输中损坏，或者没有到达另一台计算机，是没有办法补救的。也就是说，数据错了就错了，无法重传。因为 SOCK_DGRAM 所做的校验工作少，所以效率比 SOCK_STREAM 高。

QQ 视频聊天和语音聊天就使用 SOCK_DGRAM 传输数据，因为首先要保证通信的效率，尽量减小延迟，而数据的正确性是次要的，即使丢失很小的一部分数据，视频和音频也可以正常解析，最多出现噪点或杂音，不会对通信质量有实质的影响。

程序源文件主要代码展示

程序运行截图



`Server.cpp`

```
// Server.cpp

// create socket
int serv_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// bind socket with ip & port
```

```
    struct sockaddr_in serv_addr;
    // fill with 0 each byte
    memset(&serv_addr, 0, sizeof(serv_addr));
    // use IPV4
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = htons(1234);
    bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    cout << "Create socket suss!" << endl;

    // listen
    listen(serv_sock, 20);

    cout << "Listening..." << endl << endl;

    // accept client's requests
    struct sockaddr_in clnt_addr;
    socklen_t clnt_addr_size = sizeof(clnt_addr);

    while (1)
    {
        // accept can pause code to run, until receive a request
        cout << "Wait client to accept..." << endl;
        int clnt_sock = accept(serv_sock, (struct sockaddr * )&clnt_addr,
&clnt_addr_size);
        cout << "Accept one request" << endl;

        // send msg to client
        char buffer[100] = {0};
        int strLen = read(clnt_sock, buffer, sizeof(buffer) - 1);
        // int w_l = write(clnt_sock, buffer, sizeof(buffer));
        // cout << "Write " << w_l << " bytes" << endl << endl;
        cout << "From Client:" << buffer << endl << endl;
        cout << "To Client:";
        cin >> buffer;
        write(clnt_sock, buffer, sizeof(buffer));
        cout << "Write to client succ" << endl << endl;

        // close client socket
        close(clnt_sock);
        // clean buffer
        memset(buffer, 0, sizeof(buffer));

    }
    // close serv sock
    close(serv_sock);
```

`Client.cpp`

```
    // Client.cpp

    // request to serv (ip & port)
```

```
    struct sockaddr_in serv_addr;
    // fill with 0 each byte
    memset(&serv_addr, 0, sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = htons(1234);

    /*
    char usr_name[10];
    cout << "Please input your name:" << endl;
    cin >> usr_name;
    cout << "User_Name:" << usr_name << endl;
    */

    char str_smthing[100] = {0};
    char buffer[100] = {0};

    while (1)
    {
        // create socket
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

        // input smthing to send to serv
        cout << "To Server:";
        // cin >> str_smthing;
        string data;
        getline(cin, data);
        strcpy(str_smthing, data.c_str());
        // write to serv
        // int w_state = write(sock, msg, sizeof(msg));
        int w_state = write(sock, str_smthing, sizeof(str_smthing));
        cout << "Send succ :" << w_state << endl;

        // read msg from serv
        read(sock, buffer, sizeof(buffer) - 1);
        cout << "Message from server:" << buffer << endl << endl;

        // clean buffer
        memset(str_smthing, 0, sizeof(str_smthing));
        memset(buffer, 0, sizeof(buffer));
        // close sock
        close(sock);
    }
```