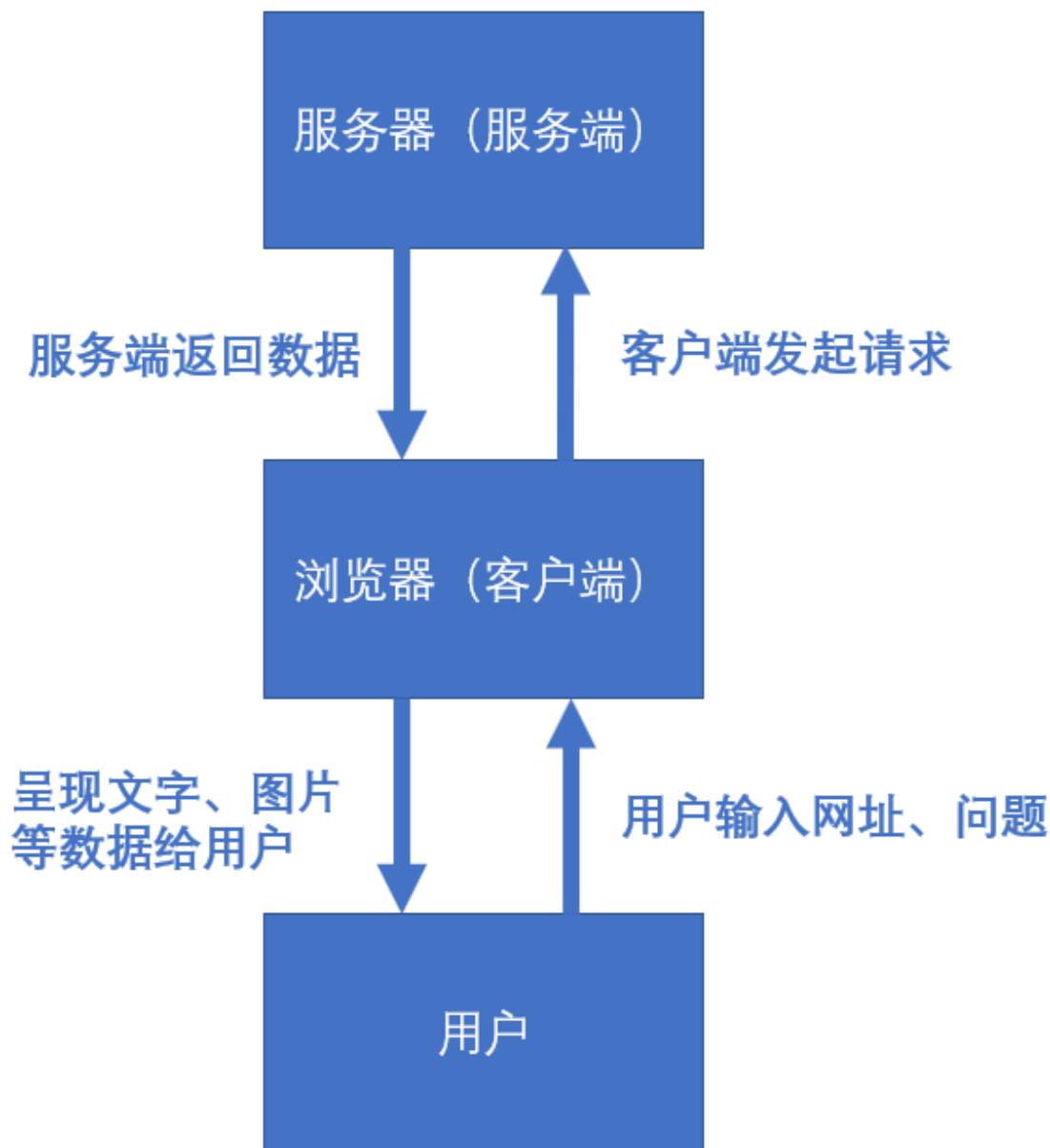


一、什么是socket？

Socket的英文原义是“孔”或“插座”。在编程中，Socket被称做 套接字，是网络通信中的一种约定。Socket编程的应用无处不在，我们平时用的QQ、微信、浏览器等程序，都与Socket编程有关。我们平时使用浏览器查资料，这个过程的技术原理是怎样的呢？



我们平时使用浏览器，大致就是这样的一个过程。这里有两个重要的名词：`服务端`与`客户端`。

Socket编程的目的就是如何实现这两端之间的通信。

1、Socket编程在嵌入式中也很重要

Socket编程不仅仅在互联网方面很重要，在我们的嵌入式方面也是非常重要，因为现在很多电子设备都趋向于联网。比如很多嵌入式工作的招聘要求都会有这一条要求：

任职要求：

- 1、大专以上学历，电子、通讯、计算机、自动化控制等相关专业；
- 2、三年以上工作经验，一年以上智能安防、智能家居行业软件开发工作经验；
- 3、熟练掌握C/C++等编程语言，了解汇编，具有良好的编程思想和规范，具有较强的调试和解决问题的能力；
- 4、了解51、STM8、MSP430等主流单片机开发，熟悉ARM7、cortex-M系列MCU的开发流程和方法；对单片机外围电路搭建和应用开发有较深入认识；
- 5、熟悉USART、SPI、IIC等单片机常用通信方式，有快速了解阅读中英文芯片手册的能力，对于新接触的IC能够尽快了解并编写驱动程序；
- 6、至少熟悉一种Zigbee芯片方案，对Zigbee协议栈、Zigbee调试方法有深入了解；对Zigbee标准协议有一定了解者优先；
- 7、熟悉网络通讯及Socket网络编程，对TCP/IP协议、HTTP协议有一定了解，有WIFI、蓝牙、GSM之一的开发工作经验，有与移动端应用通讯的单片机开发经验者优先；
- 8、具有较强的学习能力和问题分析能力，严谨的工作态度，乐观积极的工作心态，良好的团队合作精神、

任职资格:

- 1、2年以上嵌入式软件开发工作经验；
- 2、熟悉51、ARM等单片机开发，嵌入式linux系统开发；
- 3、熟悉网络协议，对TCP/IP协议有较深的理解；
- 4、具备良好的沟通能力，能耐心的做好与客户的技术沟通；

基本工作技能要求：

- 1、嵌入式软件开发经验，熟练使用C语言；
- 2、对新技术信息敏感，善于学习，有创新意识，工作态度积极主动；
- 3、良好的沟通协调能力和较强的工作责任心；

优选条件：

- 1、具备物联网方向实际项目开发经验者优先；
- 2、对物联网行业有一定了解，并有较为浓厚的兴趣者优先。”

说一点题外话，还在学校的朋友，如果感到很迷茫，不知道学什么的时候，可以上招聘网站上看看自己未来工作相关的职位的任职要求，这样就可以总结自己的一些不足、比较有针对性的去学习。

二、Socket编程中的几个重要概念

Socket编程用于解决我们客户端与服务端之间通信的问题。我们平时多多少少都有听过IP地址、端口、TCP协议、UDP协议等概念，这些都与Socket编程中相关，想要知道怎么用起来，当然得先了解它们的一些介绍。下面看一下这些专业术语的一些要点介绍：

1、什么是IP地址？

IP地址（Internet Protocol Address）是指互联网协议地址，又译为网际协议地址。IP地址被用来给Internet上的电脑一个编号。我们可以把“个人电脑”比作“一台电话”，那么“IP地址”就相当于“电话号码”。若计算机1知道计算机2的IP地址，则计算机1就能访问计算机2。

IP地址是一个32位的二进制数，通常被分割为4个“8位二进制数”（也就是4个字节）。IP地址通常用点分十进制表示成（a.b.c.d）的形式，其中，a,b,c,d都是0~255之间的十进制整数。例：点分十进IP地址（100.4.5.6），实际上是32位二进制数（01100100.00000100.00000101.00000110）。

IP地址有IPv4与IPv6之分，现在用得较多的是IPv4。其中，有一个特殊的IP地址需要我们记住：127.0.0.1，这是回送地址，即本地机，一般用来测试使用。后边我们的实例中会用到。

关于IP地址还有很多知识要点，但是对于在Socket编程中的应用，我们暂且知道这么多就可以。

2、什么是TCP/IP端口？

上一点中我们提到，若计算机1知道计算机2的IP地址，则计算机1就能访问计算机2。但是，我们要访问计算机2中的不同的应用软件，则还得需要一个信息：端口。端口使用16bit进行编号，即其范围为：0~65536。但0~1023的端口一般由系统分配给特定的服务程序，例如Web服务的端口号为80，FTP服务的端口号为21等。

3、什么是协议？

协议（Protocol）是通信双方进行数据交互的一种约定。如TCP、UDP协议：

（1）TCP协议

TCP（Transmission Control Protocol 传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，数据可以准确发送，数据丢失会重发。TCP协议常用于web应用中。

TCP连接（三次握手）

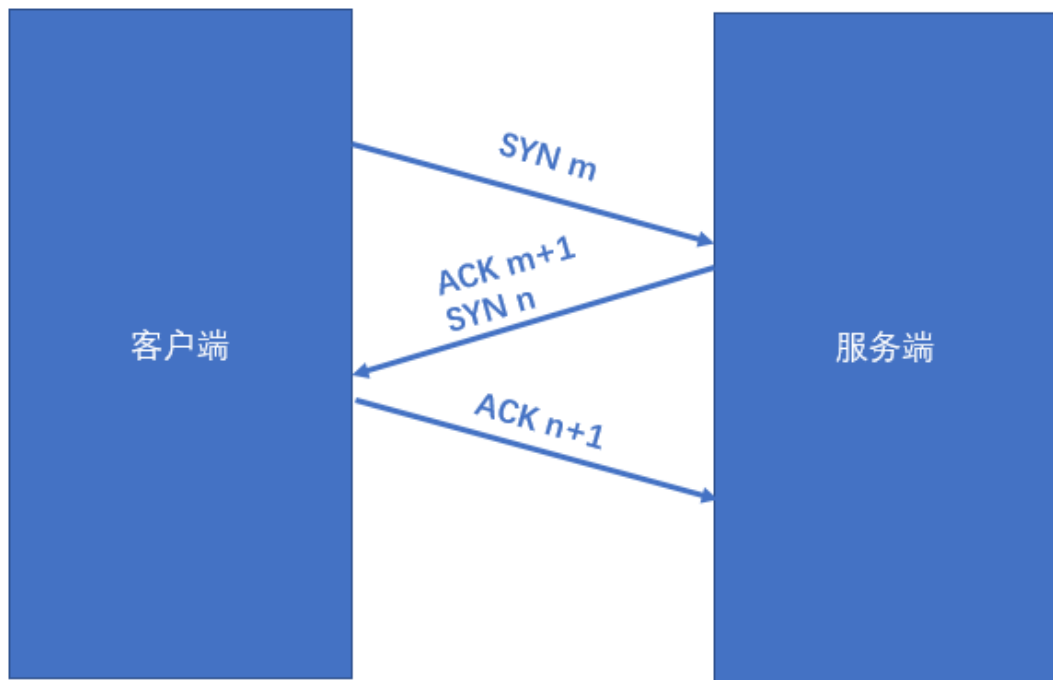
TCP传输起始时，客户端、服务端要完成三次数据交互工作才能建立连接，常称为**三次握手**。可形象比喻为如下对话：

客户端：服务端您好，我有数据要发给你，请求您开通访问权限。

服务端：客户端您好，已给您开通权限，您可以发送数据了。

客户端：收到，谢谢。

具体示意图为：



三次握手

这里的SYN和ACK都是标志位，其中SYN代表新建一个连接，ACK代表确认。其中m、n都是随机数。具体说明如：

- 第一次握手：SYN标志位被置位，客户端向服务端发送一个随机数m。
- 第二次握手：ACK、SYN标志位被置位。服务端向客户端发送m+1表示确认刚才收到的数据，同时向客户端发送一个随机数n。
- 第三次握手：ACK标志被置位。客户端向服务端发送n+1表示确认收到数据。

TCP断开（四次挥手）

TCP断开连接时，客户端、服务端要完成四次数据交互工作才能建立连接，常称为**四次挥手**。可形象比喻为如下对话：

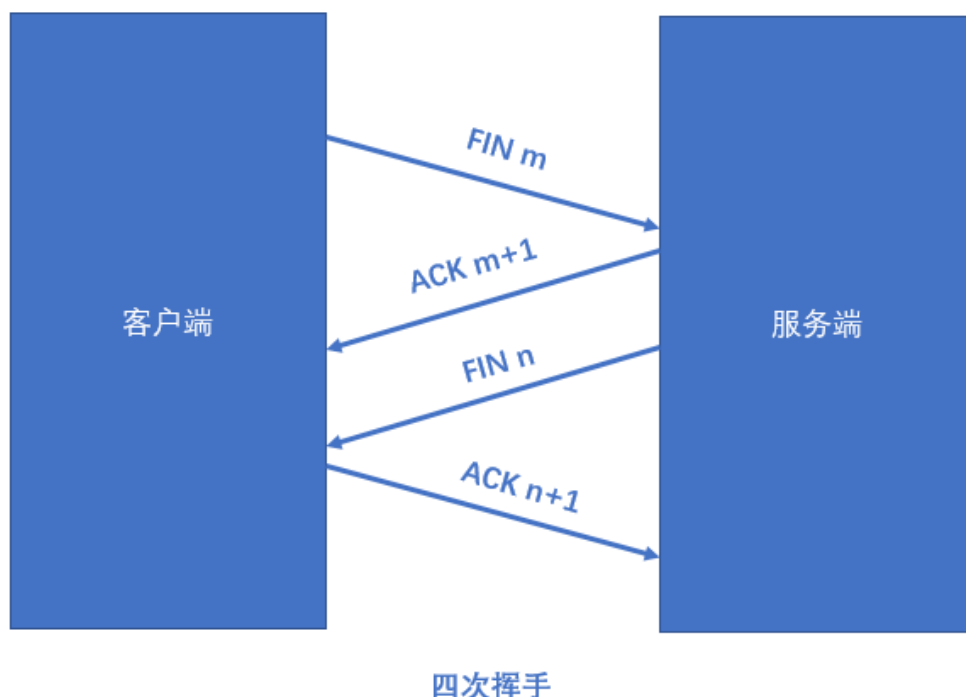
客户端：服务端您好，我发送数据完毕了，即将和您断开连接。

服务端：客户端您好，我稍稍准备一下，再给您断开

服务端：客户端您好，我准备好了，您可以断开连接了。

客户端：好的，合作愉快！

具体示意图为：



这里的FIN也是一个标志位，代表断开连接。具体说明类似**三次握手**。

为什么建立连接只需要三次数据交互，而断开连接需要四次呢？

建立连接时，服务端在监听状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。

而关闭连接时，当收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，己方也未必全部数据都发送给对方了，所以己方可以立即close，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送。

(2) UDP协议

UDP (User Datagram Protocol ， 用户数据报协议) 是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，可以保证通讯效率，传输延时小。例如视频聊天应用中用的就是UDP协议，这样可以保证及时丢失少量数据，视频的显示也不受很大影响。

4、什么是协议族？

协议族是多个协议的统称。比如我们的TCP/IP协议族，其不仅仅是TCP协议、IP协议，而是多个协议的集合，其包含IP、TCP、UDP、FTP、SMTP等协议。

三、socket编程的API接口

1、Linux下的socket API接口

(1) 创建socket : socket()函数

函数原型：

```
int socket(int af, int type, int protocol);
```

- af参数：af 为地址族（Address Family），也就是 IP 地址类型，常用的有 AF_INET 和 AF_INET6，其前缀也可以是 PF（Protocol Family），即 PF_INET 和 PF_INET6。
- type参数：type 为数据传输方式，常用的有 面向连接（SOCK_STREAM）方式（即 TCP）和无连接（SOCK_DGRAM）的方式（即 UDP）。
- protocol参数：protocol 表示传输协议，常用的有 IPPROTO_TCP 和 IPPROTO_UDP，分别表示 TCP 传输协议和 UDP 传输协议。

使用示例：

创建TCP套接字：

```
int tcp_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

创建UDP套接字：

```
int udp_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

（2）绑定套接字：bind()函数

函数原型：

```
int bind(int sock, struct sockaddr *addr, socklen_t addrlen);
```

- sock参数：sock 为 socket 文件描述符。
- addr参数：addr 为 sockaddr 结构体变量的指针。
- addrlen参数：addrlen 为 addr 变量的大小，可由 sizeof() 计算得出。

使用示例：

将创建的套接字 ServerSock 与本地IP 127.0.0.1、端口 1314 进行绑定：

```
/* 创建服务端socket */
int ServerSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

/* 设置服务端信息 */
struct sockaddr_in ServerSockAddr;
memset(&ServerSockAddr, 0, sizeof(ServerSockAddr)); // 给结构体ServerSockAddr
清零
ServerSockAddr.sin_family = PF_INET; // 使用IPv4地址
ServerSockAddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // 本机IP地址
ServerSockAddr.sin_port = htons(1314); // 端口

/* 绑定套接字 */
bind(ServerSock, (SOCKADDR*)&ServerSockAddr, sizeof(SOCKADDR));
```

其中 struct sockaddr_in 类型的结构体变量用于保存IPv4的IP信息。若是IPv6，则有对应的结构体：

```

struct sockaddr_in6
{
    sa_family_t sin6_family;    // 地址类型，取值为AF_INET6
    in_port_t sin6_port;        // 16位端口号
    uint32_t sin6_flowinfo;     // IPv6流信息
    struct in6_addr sin6_addr;   // 具体的IPv6地址
    uint32_t sin6_scope_id;     // 接口范围ID
};

```

(3) 建立连接 : connect()函数

函数原型 :

```

int connect(int sock, struct sockaddr *serv_addr, socklen_t addrlen);

```

参数与 bind() 的参数类似。

使用示例 :

```

int ClientSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
connect(ClientSock, (SOCKADDR*)&ServerSockAddr, sizeof(SOCKADDR));

```

(4) 监听 : listen()函数

函数原型 :

```

int listen(int sock, int backlog);

```

- sock参数 : sock 为需要进入监听状态的套接字。
- backlog参数 : backlog 为请求队列的最大长度。

使用示例 :

```

/* 进入监听状态 */
listen(ServerSock, 10);

```

(5) 接收请求 : accept()函数

函数原型 :

```

int accept(int sock, struct sockaddr *addr, socklen_t *addrlen);

```

- sock参数 : sock 为服务器端套接字。
- addr参数 : addr 为 sockaddr_in 结构体变量。
- addrlen参数 : addrlen 为参数 addr 的长度, 可由 sizeof() 求得。
- 返回值 : 一个新的套接字, 用于与客户端通信。

使用示例 :

```
/* 监听客户端请求，accept函数返回一个新的套接字，发送和接收都是用这个套接字 */  
int ClientSock = accept(ServerSock, (SOCKADDR*)&ClientAddr, &len);
```

(6) 关闭 : close()函数

函数原型 :

```
int close(int fd);
```

- fd : 要关闭的文件描述符。

使用示例 :

```
close(ServerSock);
```

(7) 数据的接收和发送

数据收发函数有几组 :

- read()/write()
- recv()/send()
- readv()/writev()
- recvmsg()/sendmsg()
- recvfrom()/sendto()

函数原型 :

```
ssize_t read(int fd, void *buf, size_t count);  
ssize_t write(int fd, const void *buf, size_t count);  
ssize_t send(int sockfd, const void *buf, size_t len, int flags);  
ssize_t recv(int sockfd, void *buf, size_t len, int flags);  
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,  
               const struct sockaddr *dest_addr, socklen_t addrlen);  
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,  
                struct sockaddr *src_addr, socklen_t *addrlen);  
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);  
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

这里介绍一下recv()/send()、recvfrom()/sendto()。

recv()函数 :

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- sockfd参数 : sockfd为要接收数据的套接字。
- buf参数 : buf 为要接收的数据的缓冲区地址。
- len参数 : len 为要接收的数据的字节数。
- flags参数 : flags 为接收数据时的选项，常设为0。

send()函数 :


```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

- sockfd参数：sockfd为要发送数据的套接字。
- buf参数：buf 为要发送的数据的缓冲区地址。
- len参数：len 为要发送的数据的字节数。
- flags参数：flags 为发送数据时的选项，常设为0。

recvfrom()函数：

```
ssize_t recvfrom(int sock, void *buf, size_t nbytes, int flags, struct sockaddr *from, socklen_t *addrlen);
```

- sock：用于接收UDP数据的套接字；
- buf：保存接收数据的缓冲区地址；
- nbytes：可接收的最大字节数（不能超过buf缓冲区的大小）；
- flags：可选项参数，若没有可传递0；
- from：存有发送端地址信息的sockaddr结构体变量的地址；
- addrlen：保存参数 from 的结构体变量长度的变量地址值。

sendto()函数：

```
ssize_t sendto(int sock, void *buf, size_t nbytes, int flags, struct sockaddr *to, socklen_t addrlen);
```

- sock：用于传输UDP数据的套接字；
- buf：保存待传输数据的缓冲区地址；
- nbytes：带传输数据的长度（以字节计）；
- flags：可选项参数，若没有可传递0；
- to：存有目标地址信息的 sockaddr 结构体变量的地址；
- addrlen：传递给参数 to 的地址值结构体变量的长度。

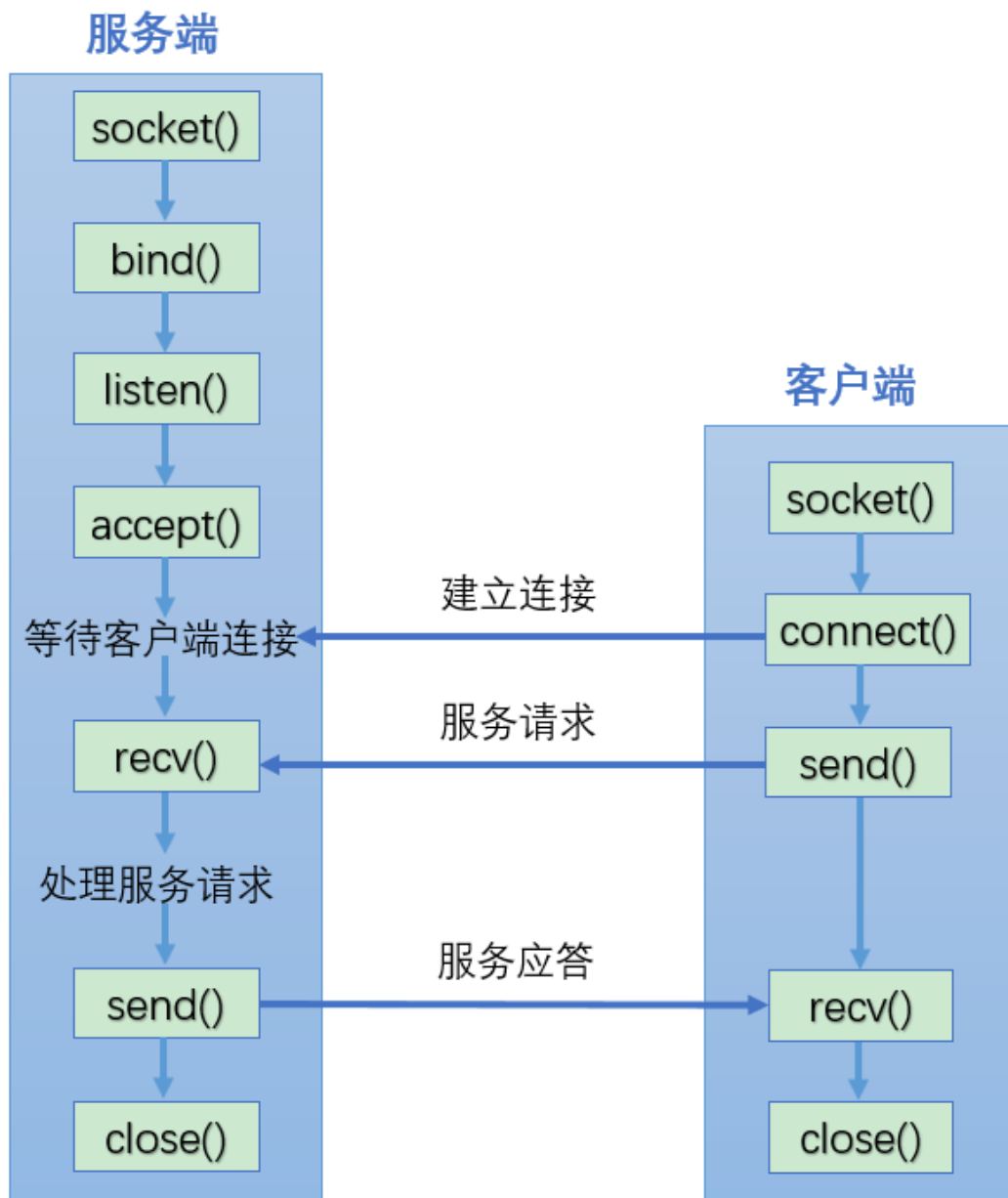
2、windows下的socket API接口

跟Linux下的差不多：

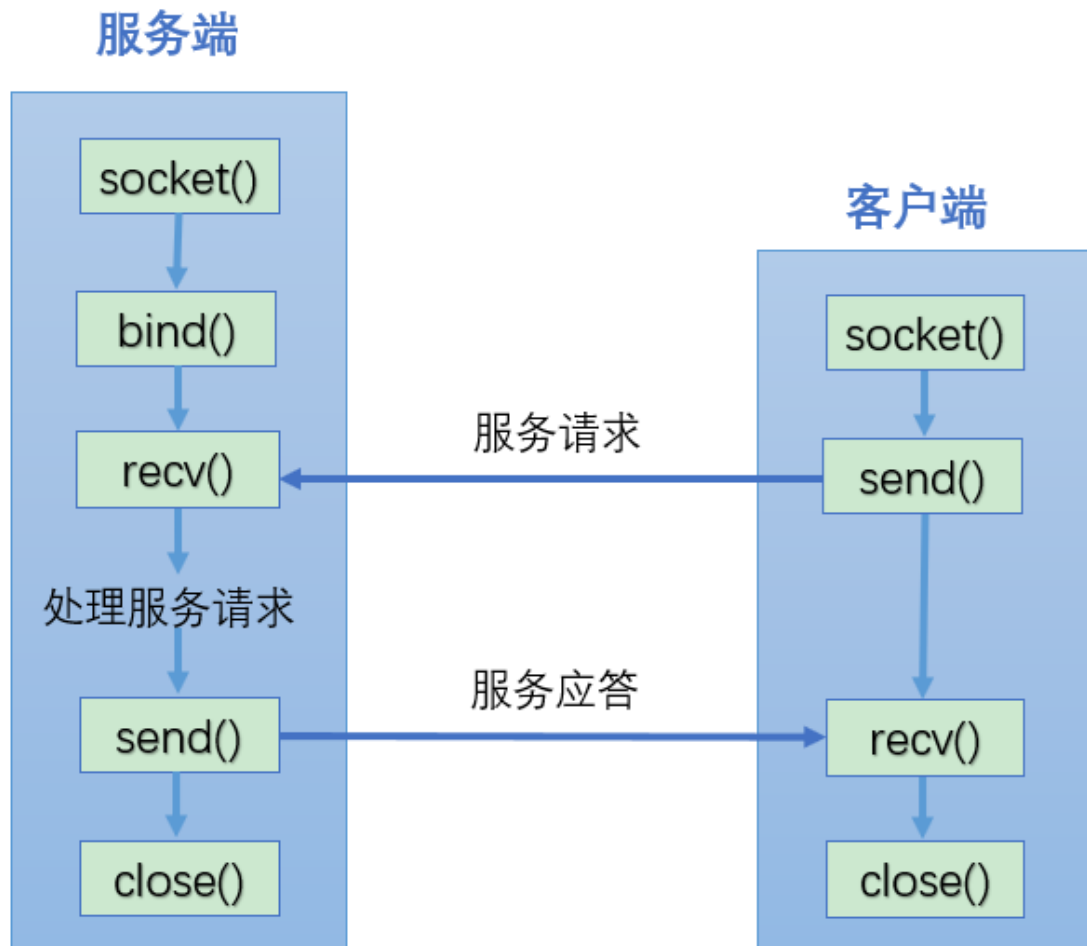
```
SOCKET socket(int af, int type, int protocol);  
int bind(SOCKET sock, const struct sockaddr *addr, int addrlen);  
int connect(SOCKET sock, const struct sockaddr *serv_addr, int addrlen);  
int listen(SOCKET sock, int backlog);  
SOCKET accept(SOCKET sock, struct sockaddr *addr, int *addrlen);  
int closesocket(SOCKET s);  
int send(SOCKET sock, const char *buf, int len, int flags);  
int recv(SOCKET sock, char *buf, int len, int flags);  
int recvfrom(SOCKET sock, char *buf, int nbytes, int flags, const struct sockaddr *from, int *addrlen);  
int sendto(SOCKET sock, const char *buf, int nbytes, int flags, const struct sockaddr *to, int addrlen);
```

3、TCP、UDP通信的socket编程过程图

(1) TCP通信socket编程过程



(2) UDP通信socket编程过程



四、socket的应用实例

1、基于TCP的本地客户端、服务端信息交互实例

本例的例子实现的功能为：本地TCP客户端往本地TCP服务端发送数据，TCP服务端收到数据则会打印输出，同时把原数据返回给TCP客户端。这个例子类似于我们在做单片机的串口实验时，串口上位机往我们的单片机发送数据，单片机收到数据则把该数据原样返回给上位机。

(1) windows的程序：

服务端程序tcp_server.c：

```
#include <stdio.h>
#include <winsock2.h>

#define BUF_LEN 100

int main(void)
{
    WSADATA wd;
    SOCKET ServerSock, ClientSock;
    char Buf[BUF_LEN] = {0};
    SOCKADDR ClientAddr;
    SOCKADDR_IN ServerSockAddr;
    int addr_size = 0, recv_len = 0;
```

```

/* 初始化操作sock需要的DLL */
WSAStartup(MAKEWORD(2,2), &wd);

/* 创建服务端socket */
if (-1 == (ServerSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)))
{
    printf("socket error!\n");
    exit(1);
}

/* 设置服务端信息 */
memset(&ServerSockAddr, 0, sizeof(ServerSockAddr)); // 给结构体
ServerSockAddr清零
ServerSockAddr.sin_family = AF_INET; // 使用IPv4地址
ServerSockAddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // 本机IP地址
ServerSockAddr.sin_port = htons(1314); // 端口

/* 绑定套接字 */
if (-1 == bind(ServerSock, (SOCKADDR*)&ServerSockAddr, sizeof(SOCKADDR)))
{
    printf("bind error!\n");
    exit(1);
}

/* 进入监听状态 */
if (-1 == listen(ServerSock, 10))
{
    printf("listen error!\n");
    exit(1);
}

addr_size = sizeof(SOCKADDR);

while (1)
{
    /* 监听客户端请求, accept函数返回一个新的套接字, 发送和接收都是用这个套接字 */
    if (-1 == (ClientSock = accept(ServerSock, (SOCKADDR*)&ClientAddr,
    &addr_size)))
    {
        printf("socket error!\n");
        exit(1);
    }

    /* 接受客户端的返回数据 */
    int recv_len = recv(ClientSock, Buf, BUF_LEN, 0);
    printf("客户端发送过来的数据为: %s\n", Buf);

    /* 发送数据到客户端 */
    send(ClientSock, Buf, recv_len, 0);

    /* 关闭客户端套接字 */
    closesocket(ClientSock);

    /* 清空缓冲区 */
    memset(Buf, 0, BUF_LEN);
}

```

```

/*如果有退出循环的条件，这里还需要清除对socket库的使用*/
/* 关闭服务端套接字 */
//closesocket(ServerSock);
/* WSACleanup();*/

return 0;
}

```

客户端程序tcp_client.c :

```

#include <stdio.h>
#include <winsock2.h>

#define BUF_LEN 100

int main(void)
{
    WSADATA wd;
    SOCKET ClientSock;
    char Buf[BUF_LEN] = {0};
    SOCKADDR_IN ServerSockAddr;

    /* 初始化操作sock需要的DLL */
    WSStartup(MAKEWORD(2,2), &wd);

    /* 向服务器发起请求 */
    memset(&ServerSockAddr, 0, sizeof(ServerSockAddr));
    ServerSockAddr.sin_family = AF_INET;
    ServerSockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    ServerSockAddr.sin_port = htons(1314);

    while (1)
    {
        /* 创建客户端socket */
        if (-1 == (ClientSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)))
        {
            printf("socket error!\n");
            exit(1);
        }
        if (-1 == connect(ClientSock, (SOCKADDR*)&ServerSockAddr,
sizeof(SOCKADDR)))
        {
            printf("connect error!\n");
            exit(1);
        }
        printf("请输入一个字符串，发送给服务端: ");
        gets(Buf);
        /* 发送数据到服务端 */
        send(ClientSock, Buf, strlen(Buf), 0);

        /* 接受服务端的返回数据 */
        recv(ClientSock, Buf, BUF_LEN, 0);
        printf("服务端发送过来的数据为: %s\n", Buf);

        memset(Buf, 0, BUF_LEN); // 重置缓冲区
    }
}

```

```

        closesocket(ClientSock);    // 关闭套接字
    }

    // WSACleanup(); /*如果有退出循环的条件，这里还需要清除对socket库的使用*/
    return 0;
}

```

我们上边的IP地址概念那一部分中，有强调 127.0.0.1 这个IP是一个特殊的IP地址，这是回送地址，即本地机，一般用来测试使用。这个例子中我们就用到了。此外，端口我们设置为 1314，这是随意设置的，只要范围在 1024~65536 之间就可以。

本文使用的是gcc编译器编译（关于gcc编译器的相关介绍可查看往期笔记：[【C语言笔记】使用notepad++、MinGW来开发C程序](#)及[【C语言笔记】windows命令行下编译C程序](#)），编译命令如下：

```

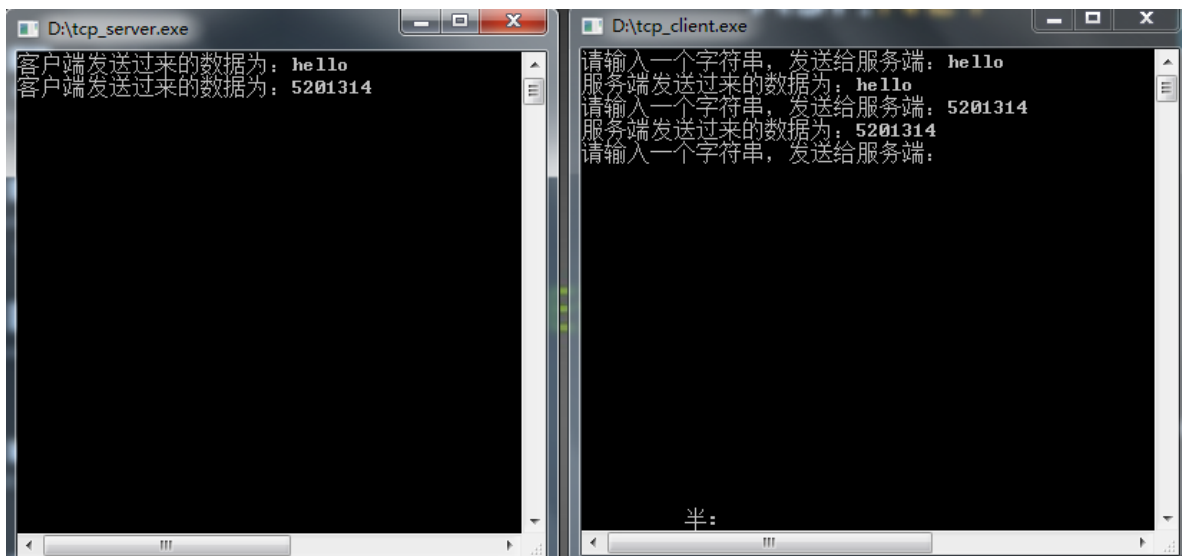
gcc tcp_client.c -o tcp_client.exe -lwsck32
gcc tcp_server.c -o tcp_server.exe -lwsck32

```

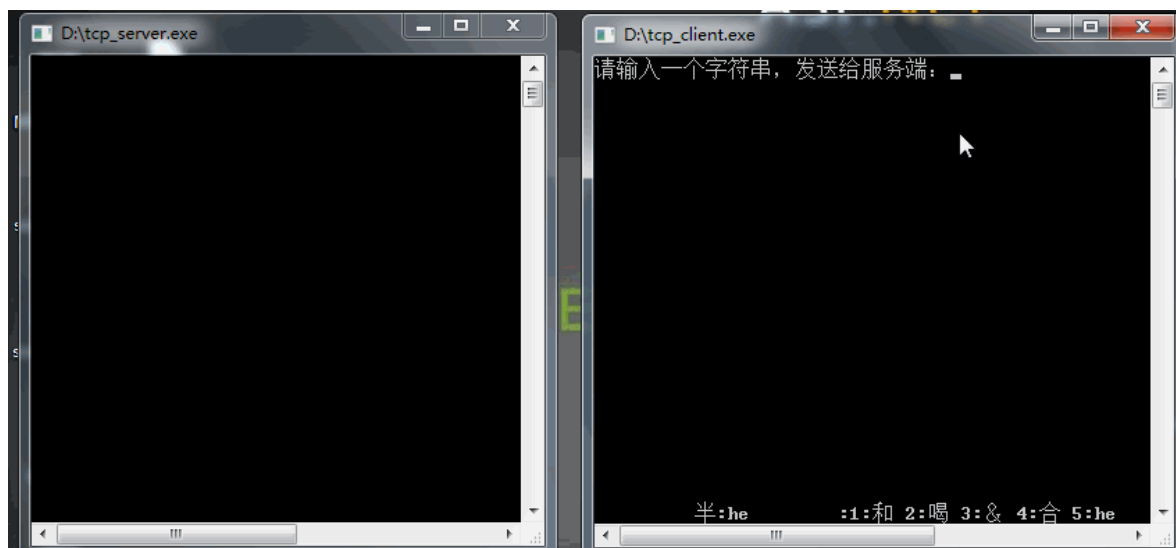
这里必须要加 -lwsck32 这个参数用于链接windows下socket编程必须的winsock2这个库。若是使用集成开发环境，则需要把 wsck32.lib 放在工程目录下，并在我们代码中 #include <winsock2.h> 下面加上一行 #pragma comment(lib, "ws2_32.lib") 代码（这种情况本人未验证，有兴趣的朋友可尝试）。

实验现象：

先启动服务端程序 tcp_server.exe，再启动客户端程序 tcp_client.exe，并在客户端中输入字符串，则当服务端会接收到字符串时会打印输出，与此同时也会往客户端返回相同的数据：



动图：



(2) Linux的程序：

在linux下，“一切都是文件”，所以这里我们的套接字也当做文件来看待。

服务端程序linux_tcp_server.c：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define BUF_LEN 100

int main(void)
{
    int ServerFd, ClientFd;
    char Buf[BUF_LEN] = {0};
    struct sockaddr ClientAddr;
    int addr_len = 0, recv_len = 0;
    struct sockaddr_in ServerSockAddr;
    int optval = 1;

    /* 创建服务端文件描述符 */
    if (-1 == (ServerFd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)))
    {
        printf("socket error!\n");
        exit(1);
    }

    /* 设置服务端信息 */

    memset(&ServerSockAddr, 0, sizeof(ServerSockAddr)); // 给结构体
    ServerSockAddr.sin_family = AF_INET; // 使用IPv4地址
    ServerSockAddr.sin_addr.s_addr = htonl(INADDR_ANY); // 自动获取IP地址
    ServerSockAddr.sin_port = htons(6666); // 端口
    ServerSockAddr.sin_len = sizeof(ServerSockAddr); // 结构体长度
    ServerSockAddr.sin_zero = '\0'; // 填充0

    if (bind(ServerFd, (struct sockaddr *)&ServerSockAddr, sizeof(ServerSockAddr)) < 0)
    {
        printf("bind error!\n");
        exit(1);
    }

    if (listen(ServerFd, 5) < 0)
    {
        printf("listen error!\n");
        exit(1);
    }

    while (1)
    {
        ClientFd = accept(ServerFd, (struct sockaddr *)&ClientAddr, &addr_len);
        if (ClientFd < 0)
        {
            printf("accept error!\n");
            continue;
        }

        recv_len = recv(ClientFd, Buf, BUF_LEN, 0);
        if (recv_len < 0)
        {
            printf("recv error!\n");
            continue;
        }

        printf("recv: %s\n", Buf);

        if (recv_len > 0)
        {
            send(ClientFd, Buf, recv_len, 0);
        }

        close(ClientFd);
    }

    close(ServerFd);
    return 0;
}
```

```

// 设置地址和端口号可以重复使用
if (setsockopt(ServerFd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval))
< 0)
{
    printf("setsockopt error!\n");
    exit(1);
}

/* 绑定操作, 绑定前加上上面的socket属性可重复使用地址 */
if (-1 == bind(ServerFd, (struct sockaddr*)&ServerSockAddr, sizeof(struct
sockaddr)))
{
    printf("bind error!\n");
    exit(1);
}

/* 进入监听状态 */
if (-1 == (listen(ServerFd, 10)))
{
    printf("listen error!\n");
    exit(1);
}

addr_len = sizeof(struct sockaddr);

while (1)
{
    /* 监听客户端请求, accept函数返回一个新的套接字, 发送和接收都是用这个套接字 */
    if (-1 == (ClientFd = accept(ServerFd, (struct sockaddr*)&ClientAddr,
&addr_len)))
    {
        printf("accept error!\n");
        exit(1);
    }

    /* 接受客户端的返回数据 */
    if ((recv_len = recv(ClientFd, Buf, BUF_LEN, 0)) < 0)
    {
        printf("recv error!\n");
        exit(1);
    }

    printf("客户端发送过来的数据为: %s\n", Buf);

    /* 发送数据到客户端 */
    send(ClientFd, Buf, recv_len, 0);

    /* 关闭客户端套接字 */
    close(ClientFd);

    /* 清空缓冲区 */
    memset(Buf, 0, BUF_LEN);
}

return 0;
}

```


客户端程序linux_tcp_client.c :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_LEN 100

int main(void)
{
    int ClientFd;
    char Buf[BUF_LEN] = {0};
    struct sockaddr_in ServerSockAddr;

    /* 向服务器发起请求 */
    memset(&ServerSockAddr, 0, sizeof(ServerSockAddr));
    ServerSockAddr.sin_family = AF_INET;
    ServerSockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    ServerSockAddr.sin_port = htons(6666);

    while (1)
    {
        /* 创建客户端socket */
        if (-1 == (ClientFd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)))
        {
            printf("socket error!\n");
            exit(1);
        }

        /* 连接 */
        if (-1 == connect(ClientFd, (struct sockaddr*)&ServerSockAddr,
            sizeof(ServerSockAddr)))
        {
            printf("connect error!\n");
            exit(1);
        }

        printf("请输入一个字符串，发送给服务端：");
        gets(Buf);
        /* 发送数据到服务端 */
        send(ClientFd, Buf, strlen(Buf), 0);
        memset(Buf, 0, BUF_LEN); // 重置缓冲区

        /* 接受服务端的返回数据 */
        recv(ClientFd, Buf, BUF_LEN, 0);
        printf("服务端发送过来的数据为: %s\n", Buf);

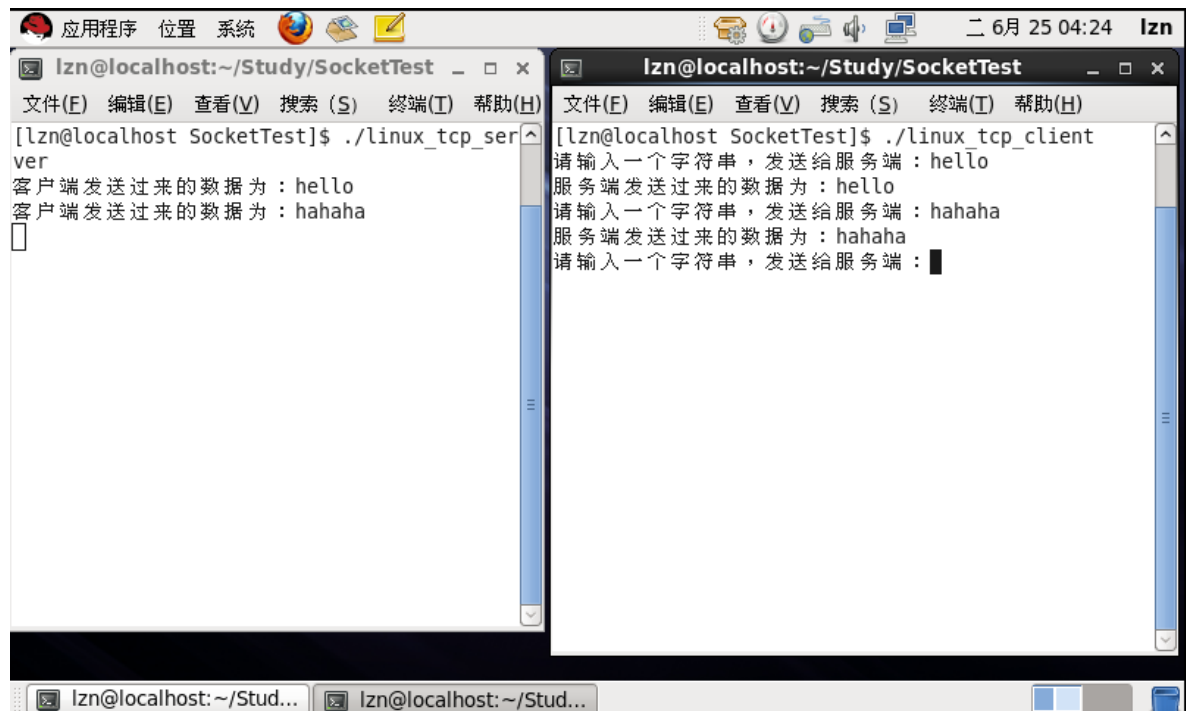
        memset(Buf, 0, BUF_LEN); // 重置缓冲区
        close(ClientFd); // 关闭套接字
    }
}
```

```
    return 0;
}
```

Linux下编译就不需要添加 `-lwsck32` 参数：

```
gcc linux_tcp_server.c -o linux_tcp_server
gcc linux_tcp_client.c -o linux_tcp_client
```

实验现象：



```
lzn@localhost:~/Study/SocketTest [lzn@localhost SocketTest]$ ./linux_tcp_server
客户端发送过来的数据为：hello
客户端发送过来的数据为：hahaha

lzn@localhost:~/Study/SocketTest [lzn@localhost SocketTest]$ ./linux_tcp_client
请输入一个字符串，发送给服务端：hello
服务端发送过来的数据为：hello
请输入一个字符串，发送给服务端：hahaha
服务端发送过来的数据为：hahaha
请输入一个字符串，发送给服务端：
```

在调试这份程序时，出现了绑定错误：



```
lzn@localhost:~/Study/SocketTest [lzn@localhost SocketTest]$ ./linux_tcp_server
bind error!
[lzn@localhost SocketTest]$
```

经上网查询发现是端口重复使用，可以在调用 `bind()` 函数之前调用 `setsockopt()` 函数以解决端口重复使用的问题：

```
// 设置地址和端口号可以重复使用
if (setsockopt(ServerFd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0)
{
    printf("setsockopt error!\n");
    exit(1);
}

/* 绑定操作，绑定前加上上面的socket属性可重复使用地址 */
if (-1 == bind(ServerFd, (struct sockaddr*)&ServerSockAddr, sizeof(struct sockaddr)))
{
    printf("bind error!\n");
    exit(1);
}
```

2、基于UDP的本地客户端、服务端信息交互实例

(1) windows的程序

服务端程序udp_server.c :

```
#include <stdio.h>
#include <winsock2.h>

#define BUF_LEN 100

int main(void)
{
    WSADATA wd;
    SOCKET ServerSock;
    char Buf[BUF_LEN] = {0};
    SOCKADDR ClientAddr;
    SOCKADDR_IN ServerSockAddr;
    int addr_size = 0;

    /* 初始化操作socket需要的DLL */
    WSStartup(MAKEWORD(2,2), &wd);

    /* 创建服务端socket */
    if(-1 == (ServerSock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)))
    {
        printf("socket error!\n");
        exit(1);
    }

    /* 设置服务端信息 */
    memset(&ServerSockAddr, 0, sizeof(ServerSockAddr)); // 给结构体
    ServerSockAddr.sin_family = AF_INET; // 使用IPv4地址
    ServerSockAddr.sin_addr.s_addr = htonl(INADDR_ANY); // 自动获取IP地址
    ServerSockAddr.sin_port = htons(1314); // 端口

    /* 绑定套接字 */

    if (-1 == (bind(ServerSock, (SOCKADDR*)&ServerSockAddr, sizeof(SOCKADDR))))
    {
        printf("bind error!\n");
        exit(1);
    }

    addr_size = sizeof(SOCKADDR);

    while (1)
    {
        /* 接受客户端的返回数据 */
        int str_len = recvfrom(ServerSock, Buf, BUF_LEN, 0, &ClientAddr,
        &addr_size);

        printf("客户端发送过来的数据为: %s\n", Buf);
    }
}
```

```

        /* 发送数据到客户端 */
        sendto(ServerSock, Buf, str_len, 0, &ClientAddr, addr_size);

        /* 清空缓冲区 */
        memset(Buf, 0, BUF_LEN);
    }

    /*如果有退出循环的条件，这里还需要清除对socket库的使用*/
    /* 关闭服务端套接字 */
    //closesocket(ServerSock);
    /* WSACleanup();*/

    return 0;
}

```

客户端程序udp_client.c :

```

#include <stdio.h>
#include <winsock2.h>

#define BUF_LEN 100

int main(void)
{
    WSADATA wd;
    SOCKET ClientSock;
    char Buf[BUF_LEN] = {0};
    SOCKADDR ServerAddr;
    SOCKADDR_IN ServerSockAddr;
    int ServerAddrLen = 0;

    /* 初始化操作sock需要的DLL */
    WSStartup(MAKEWORD(2,2), &wd);

    /* 创建客户端socket */
    if (-1 == (ClientSock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)))
    {
        printf("socket error!\n");
        exit(1);
    }

    /* 向服务器发起请求 */
    memset(&ServerSockAddr, 0, sizeof(ServerSockAddr));
    ServerSockAddr.sin_family = PF_INET;
    ServerSockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    ServerSockAddr.sin_port = htons(1314);

    ServerAddrLen = sizeof(ServerAddr);

    while (1)
    {
        printf("请输入一个字符串，发送给服务端: ");
        gets(Buf);
        /* 发送数据到服务端 */
    }
}

```

```

        sendto(ClientSock, Buf, strlen(Buf), 0, (struct
sockaddr*)&ServerSockAddr, sizeof(ServerSockAddr));

        /* 接受服务端的返回数据 */
        recvfrom(ClientSock, Buf, BUF_LEN, 0, &ServerAddr, &ServerAddrLen);
        printf("服务端发送过来的数据为: %s\n", Buf);

        memset(Buf, 0, BUF_LEN);    // 重置缓冲区
    }

    closesocket(ClientSock);    // 关闭套接字
    // WSACleanup(); /*如果有退出循环的条件, 这里还需要清除对socket库的使用*/
    return 0;
}

```

(2) Linux下的程序

服务端程序linux_udp_server.c :

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define BUF_LEN 100

int main(void)
{
    int ServerFd;
    char Buf[BUF_LEN] = {0};
    struct sockaddr ClientAddr;
    struct sockaddr_in ServerSockAddr;
    int addr_size = 0;
    int optval = 1;

    /* 创建服务端socket */
    if ( -1 == (ServerFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)))
    {
        printf("socket error!\n");
        exit(1);
    }

    /* 设置服务端信息 */
    memset(&ServerSockAddr, 0, sizeof(ServerSockAddr));    // 给结构体
ServerSockAddr清零
    ServerSockAddr.sin_family = AF_INET;    // 使用IPv4地址
    ServerSockAddr.sin_addr.s_addr = htonl(INADDR_ANY);    // 自动获取IP地址
    ServerSockAddr.sin_port = htons(1314);    // 端口

    // 设置地址和端口号可以重复使用
    if (setsockopt(ServerFd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval))
< 0)
    {

```

```

        printf("setsockopt error!\n");
        exit(1);
    }

    /* 绑定操作, 绑定前加上上面的socket属性可重复使用地址 */
    if (-1 == bind(ServerFd, (struct sockaddr*)&ServerSockAddr,
sizeof(ServerSockAddr)))
    {
        printf("bind error!\n");
        exit(1);
    }

    addr_size = sizeof(ClientAddr);

    while (1)
    {
        /* 接受客户端的返回数据 */
        int str_len = recvfrom(ServerFd, Buf, BUF_LEN, 0, &ClientAddr,
&addr_size);

        printf("客户端发送过来的数据为: %s\n", Buf);

        /* 发送数据到客户端 */
        sendto(ServerFd, Buf, str_len, 0, &ClientAddr, addr_size);

        /* 清空缓冲区 */
        memset(Buf, 0, BUF_LEN);
    }

    close(ServerFd);

    return 0;
}

```

客户端程序linux_udp_client.c :

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_LEN 100

int main(void)
{
    int ClientFd;
    char Buf[BUF_LEN] = {0};
    struct sockaddr ServerAddr;
    int addr_size = 0;
    struct sockaddr_in ServersockAddr;

    /* 创建客户端socket */
    if (-1 == (ClientFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)))

```

```

{
    printf("socket error!\n");
    exit(1);
}

/* 向服务器发起请求 */
memset(&ServerSockAddr, 0, sizeof(ServerSockAddr));
ServerSockAddr.sin_family = PF_INET;
ServerSockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
ServerSockAddr.sin_port = htons(1314);

addr_size = sizeof(ServerAddr);

while (1)
{
    printf("请输入一个字符串，发送给服务端：");
    gets(Buf);
    /* 发送数据到服务端 */
    sendto(ClientFd, Buf, strlen(Buf), 0, (struct sockaddr*)&ServerSockAddr,
    sizeof(ServerSockAddr));

    /* 接受服务端的返回数据 */
    recvfrom(ClientFd, Buf, BUF_LEN, 0, &ServerAddr, &addr_size);
    printf("服务端发送过来的数据为：%s\n", Buf);

    memset(Buf, 0, BUF_LEN);    // 重置缓冲区
}

close(ClientFd);    // 关闭套接字

return 0;
}

```

实验现象：

实验现象如实例1。

五、总结

本笔记简单介绍了一些与socket编程相关的一些知识点：IP地址，什么是端口，协议等。重点介绍了TCP、UDP通信的一些原理及其API接口的用法，并给出了windows和linux下的TCP、UDP通信实例。以上就是关于socket编程的一些总结，如有错误，欢迎指出！

我的微信公众号：

设为星标 ★



嵌入式大杂烩

📍 扫码关注不迷路

嵌入式 · 物联网 · 编程笔记 · 实用技巧 · 资源分享