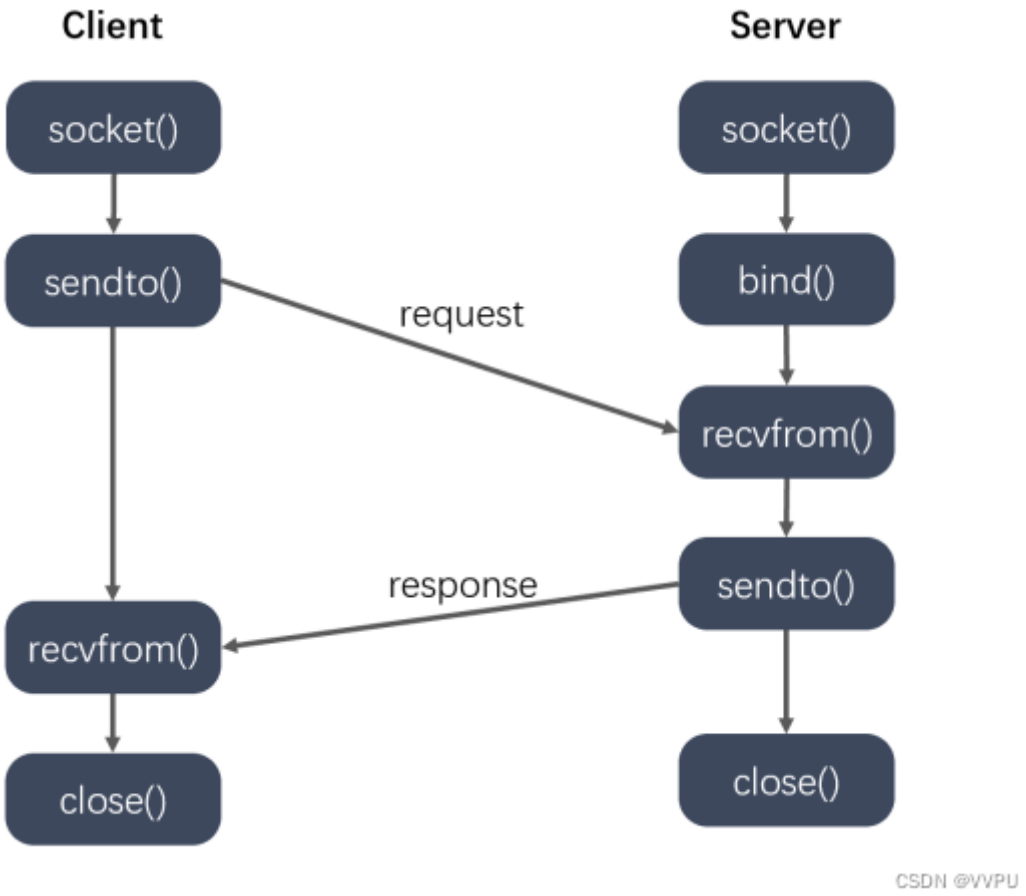
UDP通信

UDP 为应用程序提供了一种无需建立连接就可以发送封装的 IP 数据包的方法。

UDP通信（单播）



发送接收函数详解

```
    #include <sys/types.h>
    #include <sys/socket.h>
    ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const
struct sockaddr *dest_addr, socklen_t addrlen);
    - 参数:
        - sockfd : 通信的fd
        - buf : 要发送的数据
        - len : 发送数据的长度
        - flags : 使用0即可。
        - dest_addr : 通信的另外一端的地址信息
        - addrlen : 地址的内存大小

    ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct
sockaddr *src_addr, socklen_t *addrlen);
    - 参数:
        - sockfd : 通信的fd
        - buf : 接收数据的数组
        - len : 数组的大小
        - flags : 0。同sendto
        - src_addr : 用来保存另外一端的地址信息，不需要可以指定为NULL
```

```
        - addrlen : 地址的内存大小
```

代码实现

1、服务端

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

int main() {

    // 1.创建一个通信的socket
    int fd = socket(PF_INET, SOCK_DGRAM, 0);     //第二个参数表示使用UDP协议
    if(fd == -1) {
        perror("socket");
        exit(-1);
    }

    //服务端的socket地址信息
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(9999);
    addr.sin_addr.s_addr = INADDR_ANY;

    // 2.绑定socket地址信息
    int ret = bind(fd, (struct sockaddr *)&addr, sizeof(addr));
    if(ret == -1) {
        perror("bind");
        exit(-1);
    }

    // 3.通信。使用upd，即使不用多线程多进程并发，也可以让多个客户端与服务端通信。
    while(1) {
        char recvbuf[128];
        char ipbuf[16];

        struct sockaddr_in cliaddr;
        int len = sizeof(cliaddr);

        // 接收数据
        int num = recvfrom(fd, recvbuf, sizeof(recvbuf), 0, (struct sockaddr
*)&cliaddr, &len);

        //输出客户端的socket地址信息
        printf("client IP : %s, Port : %d\n",
            inet_ntop(AF_INET, &cliaddr.sin_addr.s_addr, ipbuf, sizeof(ipbuf)),
            ntohs(cliaddr.sin_port));

        printf("client say : %s\n", recvbuf);

        // 发送数据
        sendto(fd, recvbuf, strlen(recvbuf) + 1, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
```

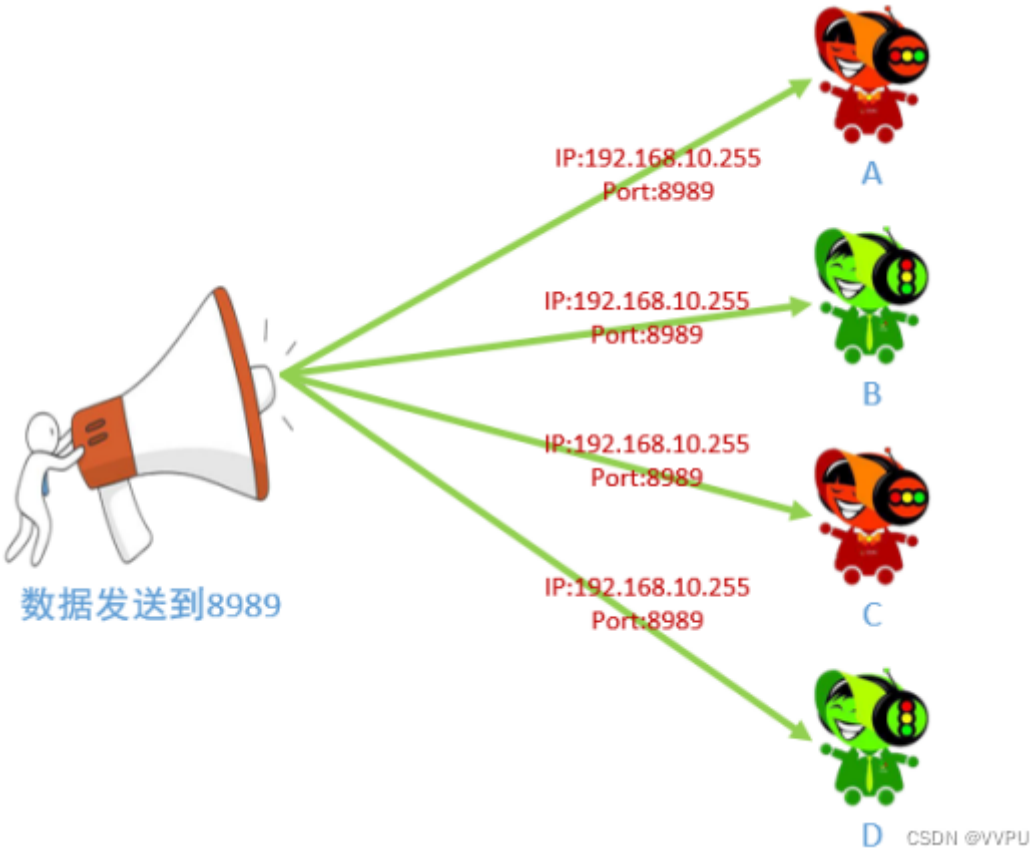
```
    }

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

2、客户端

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

int main() {

    // 1.创建一个通信的socket
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd == -1) {
        perror("socket");
        exit(-1);
    }

    // 封装服务器的地址信息
    struct sockaddr_in saddr;
    saddr.sin_family = AF_INET;
    saddr.sin_port = htons(9999);
    inet_pton(AF_INET, "192.168.43.131", &saddr.sin_addr.s_addr);

    int num = 0;
    // 3.通信
    while(1) {

        // 向服务端发送数据
        char sendBuf[128];
        sprintf(sendBuf, "hello , i am client %d \n", num++);
        sendto(fd, sendBuf, strlen(sendBuf) + 1, 0, (struct sockaddr *)&saddr,
sizeof(saddr));

        // 接收来自服务端的数据
        int num = recvfrom(fd, sendBuf, sizeof(sendBuf), 0, NULL, NULL);
        printf("server say : %s\n", sendBuf);

        sleep(1);
    }

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

UDP通信（广播）

向子网中多台计算机发送消息，并且子网中所有的计算机都可以接收到发送方发送的消息，每个广播消息都包含一个特殊的IP地址，这个IP中子网内主机标志部分的二进制全部为1。
a.只能在局域网中使用。
b.接收端需要绑定服务器广播使用的端口，才可以接收到广播消息。



广播属性设置函数详解

```
    // 设置广播属性的函数
    int setsockopt(int sockfd, int level, int optname,const void *optval,
socklen_t optlen);
        - sockfd : 文件描述符
        - level : SOL_SOCKET -- 被设置的选项的级别，如果想要在套接字级别上设置选项，就必须
把level设置为 SOL_SOCKET
        - optname : SO_BROADCAST -- 允许或禁止发送广播数据。
        - optval : int类型的值，为1表示允许广播
        - optlen : optval的大小
```

代码实现

1、发送端

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

int main() {

    // 1.创建一个通信的socket
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd == -1) {
        perror("socket");
        exit(-1);
    }

    // 2.设置广播属性
```

```
    int op = 1;
    setsockopt(fd, SOL_SOCKET, SO_BROADCAST, &op, sizeof(op));

    // 3.创建一个广播的地址。向这个广播局域网中的所有主机发送数据
    struct sockaddr_in cliaddr;
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_port = htons(9999);     //设置端口号，接收的主机需要绑定这个端口号
    inet_pton(AF_INET, "192.168.193.255", &cliaddr.sin_addr.s_addr);

    // 3.通信。服务端作为广播方，仅发送数据，不接受数据
    int num = 0;
    while(1) {

        char sendBuf[128];
        sprintf(sendBuf, "hello, client....%d\n", num++);
        // 发送数据
        sendto(fd, sendBuf, strlen(sendBuf) + 1, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        printf("广播的数据：%s\n", sendBuf);
        sleep(1);
    }

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

2、接收端

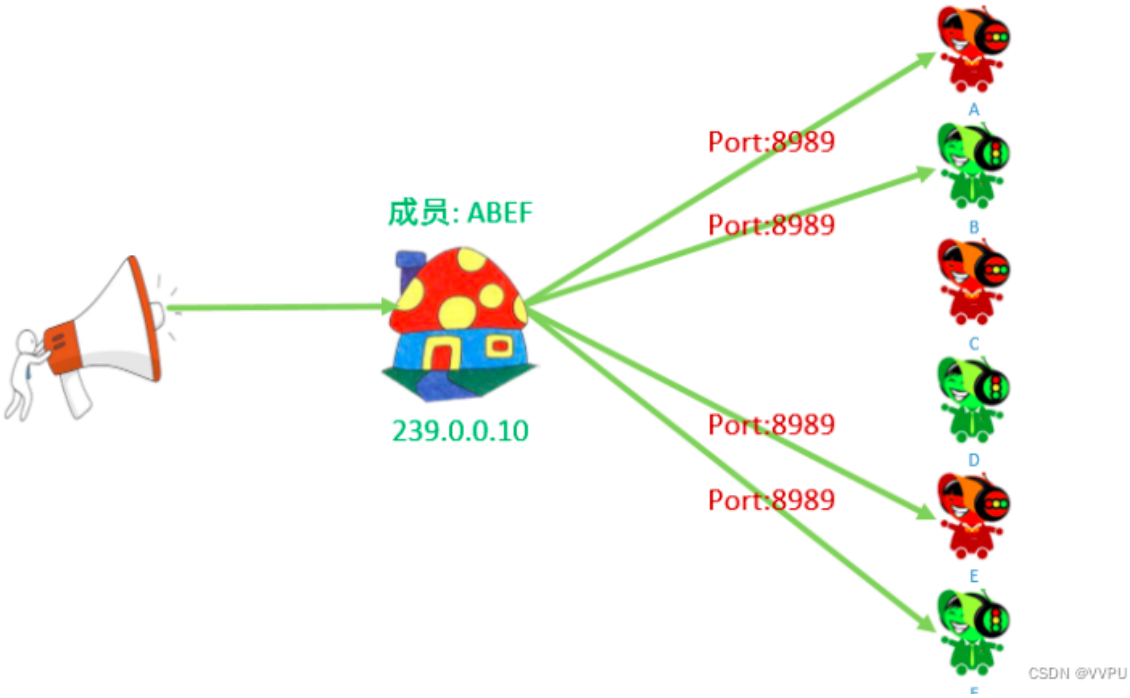
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>

int main() {

    // 1.创建一个通信的socket
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd == -1) {
        perror("socket");
        exit(-1);
    }

    // 2.客户端绑定本地的IP和端口
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(9999);
    addr.sin_addr.s_addr = INADDR_ANY;

    int ret = bind(fd, (struct sockaddr *)&addr, sizeof(addr));
    if(ret == -1) {
        perror("bind");
        exit(-1);
    }

    // 3.通信
    while(1) {
```

```
        char buf[128];
        // 接收数据
        int num = recvfrom(fd, buf, sizeof(buf), 0, NULL, NULL);
        printf("server say : %s\n", buf);

    }

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

UDP通信（组播/多播）

简介

单播地址标识单个 IP 接口，广播地址标识某个子网的所有 IP 接口，多播地址标识一组 IP 接口。
单播和广播是寻址方案的两个极端（要么单个要么全部），多播则意在两者之间提供一种折中方案。多播数据报只应该由对它感兴趣的接口接收，也就是说由运行相应多播会话应用系统的主机上的接口接收。另外，广播一般局限于局域网内使用，而多播则既可以用于局域网，也可以跨广域网使用。
a.组播既可以用于局域网，也可以用于广域网
b.客户端需要加入多播组，才能接收到多播的数据



组播地址

> IP 多播通信必须依赖于 IP 多播地址，在 IPv4 中它的范围从 `224.0.0.0` 到 `239.255.255.255`，并被划分为局部链接多播地址、预留多播地址和管理权限多播地址三类:

IP地址	说明
224.0.0.0~224.0.0.255	局部链接多播地址：是为路由协议和其它用途保留的地址，路由器并不转发属于此范围的IP包
224.0.1.0~224.0.1.255	预留多播地址：公用组播地址，可用于Internet；使用前需要申请
224.0.2.0~238.255.255.255	预留多播地址：用户可用组播地址(临时组地址)，全网范围内有效
239.0.0.0~239.255.255.255	本地管理组播地址，可供组织内部使用，类似于私有 IP 地址，不能用于 Internet，可限制多播范围

函数说明

```
int setsockopt(int sockfd, int level, int optname,const void *optval, socklen_t
optlen);
// 服务器设置多播的信息，外出接口
    - level : IPPROTO_IP
    - optname : IP_MULTICAST_IF
    - optval : struct in_addr
// 客户端加入到多播组：
    - level : IPPROTO_IP
    - optname : IP_ADD_MEMBERSHIP
    - optval : struct ip_mreq

struct ip_mreq
{
    /* IP multicast address of group. */
    struct in_addr imr_multiaddr; // 组播的IP地址
    /* Local IP address of interface. */
    struct in_addr imr_interface; // 本地的IP地址
};
```

```
typedef uint32_t in_addr_t;

struct in_addr
{
    in_addr_t s_addr;
};
```

代码实现

1、发送端

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

int main() {

    // 1.创建一个通信的socket
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd == -1) {
        perror("socket");
        exit(-1);
    }

    // 2.设置多播的属性，设置外出接口
    struct in_addr imr_multiaddr;
    // 初始化多播地址
    inet_pton(AF_INET, "239.0.0.10", &imr_multiaddr.s_addr);
    setsockopt(fd, IPPROTO_IP, IP_MULTICAST_IF, &imr_multiaddr,
sizeof(imr_multiaddr));

    // 3.初始化客户端的地址信息
    struct sockaddr_in cliaddr;
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_port = htons(9999);
    inet_pton(AF_INET, "239.0.0.10", &cliaddr.sin_addr.s_addr);

    // 3.通信
    int num = 0;
    while(1) {

        char sendBuf[128];
        sprintf(sendBuf, "hello, client....%d\n", num++);
        // 发送数据
        sendto(fd, sendBuf, strlen(sendBuf) + 1, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        printf("组播的数据：%s\n", sendBuf);
        sleep(1);
    }

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

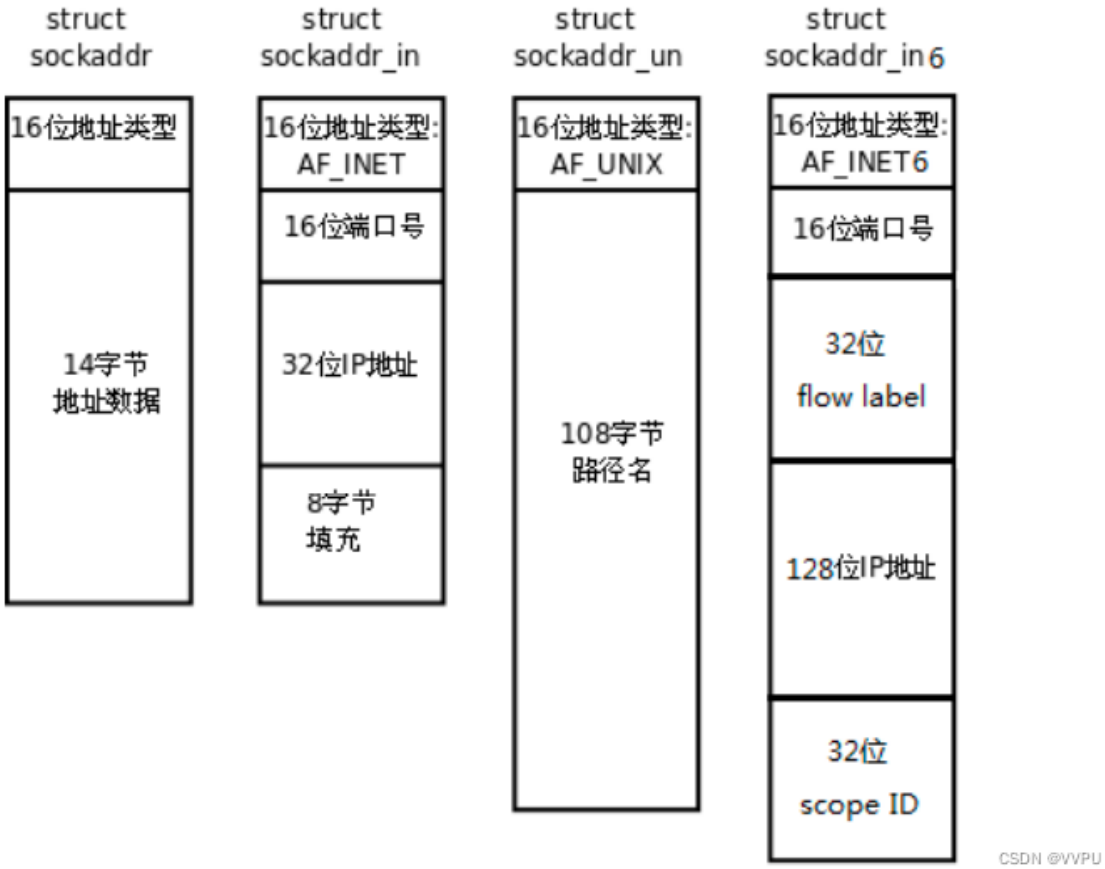
2、接收端

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

int main() {

    // 1.创建一个通信的socket
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd == -1) {
        perror("socket");
        exit(-1);
    }

    struct in_addr in;
    // 2.客户端绑定本地的IP和端口
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(9999);
    addr.sin_addr.s_addr = INADDR_ANY;

    int ret = bind(fd, (struct sockaddr *)&addr, sizeof(addr));
    if(ret == -1) {
        perror("bind");
        exit(-1);
    }

    struct ip_mreq op;
    inet_pton(AF_INET, "239.0.0.10", &op.imr_multiaddr.s_addr);
    op.imr_interface.s_addr = INADDR_ANY;

    // 加入到多播组
    setsockopt(fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &op, sizeof(op));

    // 3.通信
    while(1) {

        char buf[128];
        // 接收数据
        int num = recvfrom(fd, buf, sizeof(buf), 0, NULL, NULL);
        printf("server say : %s\n", buf);

    }

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

本地套接字

简介

本地套接字的作用：本地的进程间通信
有关系的进程间的通信
没有关系的进程间的通信
本地套接字实现流程和网络套接字类似，一般呢采用TCP的通信流程。



```
// 本地套接字通信的流程 - tcp

// 服务器端
1. 创建监听的套接字
    int lfd = socket(AF_UNIX/AF_LOCAL, SOCK_STREAM, 0);
2. 监听的套接字绑定本地的套接字文件 -> server端
    struct sockaddr_un addr;
    // 绑定成功之后，指定的sun_path中的套接字文件会自动生成。
    bind(lfd, addr, len);
3. 监听
    listen(lfd, 100);
4. 等待并接受连接请求
    struct sockaddr_un cliaddr;
    int cfd = accept(lfd, &cliaddr, len);
5. 通信
    接收数据：read/recv
    发送数据：write/send
6. 关闭连接
    close();


// 客户端的流程
1. 创建通信的套接字
    int fd = socket(AF_UNIX/AF_LOCAL, SOCK_STREAM, 0);
2. 监听的套接字绑定本地的IP 端口
    struct sockaddr_un addr;
    // 绑定成功之后，指定的sun_path中的套接字文件会自动生成。
    bind(lfd, addr, len);
3. 连接服务器
    struct sockaddr_un serveraddr;
    connect(fd, &serveraddr, sizeof(serveraddr));
4. 通信
    接收数据：read/recv
    发送数据：write/send
5. 关闭连接
    close();
```



```
// 头文件:  sys/un.h
#define UNIX_PATH_MAX 108
struct sockaddr_un {
    sa_family_t sun_family; // 地址族协议 af_local
    char sun_path[UNIX_PATH_MAX];    // 套接字文件的路径，这是一个伪文件，大小永远=0
};
```


代码实现

1、服务端

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

```c
#include <sys/un.h>

int main()
{

    unlink("server.sock");

    // 1.创建监听的套接字
    int lfd = socket(AF_LOCAL, SOCK_STREAM, 0);
    if (lfd == -1)
    {
        perror("socket");
        exit(-1);
    }

    // 2.绑定本地套接字文件
    struct sockaddr_un addr;
    addr.sun_family = AF_LOCAL;
    strcpy(addr.sun_path, "server.sock");  //执行时会生成这个文件，借助于这个文件进行双方通信
    int ret = bind(lfd, (struct sockaddr *)&addr, sizeof(addr));
    if (ret == -1)
    {
        perror("bind");
        exit(-1);
    }

    // 3.监听
    ret = listen(lfd, 100);
    if (ret == -1)
    {
        perror("listen");
        exit(-1);
    }

    // 4.等待客户端连接
    struct sockaddr_un cliaddr;
    int len = sizeof(cliaddr);
    int cfd = accept(lfd, (struct sockaddr *)&cliaddr, &len);
    if (cfd == -1)
    {
        perror("accept");
        exit(-1);
    }

    printf("client socket filename: %s\n", cliaddr.sun_path);

    // 5.通信
    while (1)
    {

        char buf[128];
        int len = recv(cfd, buf, sizeof(buf), 0);

        if (len == -1)
        {
            perror("recv");
            exit(-1);
```

```
        }
        else if (len == 0)
        {
            printf("client closed....\n");
            break;
        }
        else if (len > 0)
        {
            printf("client say : %s\n", buf);
            send(cfd, buf, len, 0);
        }
    }

    close(cfd);
    close(lfd);

    return 0;
}
```

2、客户端

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

int main()
{

    unlink("client.sock");

    // 1.创建套接字
    int cfd = socket(AF_LOCAL, SOCK_STREAM, 0);
    if (cfd == -1)
    {
        perror("socket");
        exit(-1);
    }

    // 2.绑定本地套接字文件
    struct sockaddr_un addr;
    addr.sun_family = AF_LOCAL;
    strcpy(addr.sun_path, "client.sock");
    int ret = bind(cfd, (struct sockaddr *)&addr, sizeof(addr));
    if (ret == -1)
    {
        perror("bind");
        exit(-1);
    }

    // 3.连接服务器
    struct sockaddr_un seraddr;
    seraddr.sun_family = AF_LOCAL;
    strcpy(seraddr.sun_path, "server.sock");
    ret = connect(cfd, (struct sockaddr *)&seraddr, sizeof(seraddr));
```

```
    if (ret == -1)
    {
        perror("connect");
        exit(-1);
    }

    // 4.通信
    int num = 0;
    while (1)
    {

        // 发送数据
        char buf[128];
        sprintf(buf, "hello, i am client %d\n", num++);
        send(cfd, buf, strlen(buf) + 1, 0);
        printf("client say : %s\n", buf);

        // 接收数据
        int len = recv(cfd, buf, sizeof(buf), 0);

        if (len == -1)
        {
            perror("recv");
            exit(-1);
        }
        else if (len == 0)
        {
            printf("server closed....\n");
            break;
        }
        else if (len > 0)
        {
            printf("server say : %s\n", buf);
        }

        sleep(1);
    }

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

执行后生成两个server.sock和client.sock两个文件，且文件大小始终是0.

```
kiko@Hkiko:~/lesson/network$ ls -al | grep .sock
srwxrwxr-x  1 kiko kiko     0 7月  29 08:28 client.sock
srwxrwxr-x  1 kiko kiko     0 7月  29 08:28 server.sock
-rw-rw-r--  1 kiko kiko  1371 7月  17 09:01 socketadd.c
```