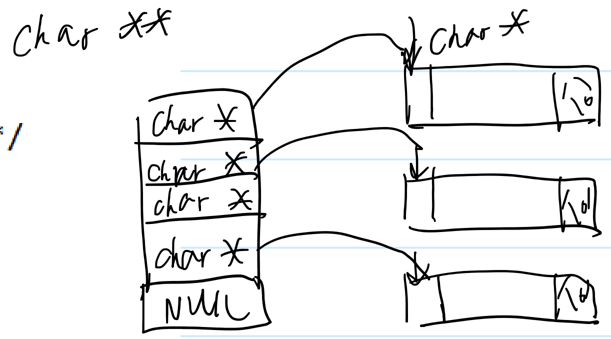
# gethostbyname

2023年5月8日 9:32

```
struct hostent {
    char  *h_name;            /* official name of host */
    char **h_aliases;         /* alias list */
    int    h_addrtype;        /* host address type */
    int    h_length;          /* length of address */
    char **h_addr_list;       /* list of addresses */
}
#define h_addr h_addr_list[0] /* for backward compatibility */
```



AF_INET / AF_INET6 ...

地址长度 4/16

ip地址列表（二进制）

2023年5月8日　　9:53

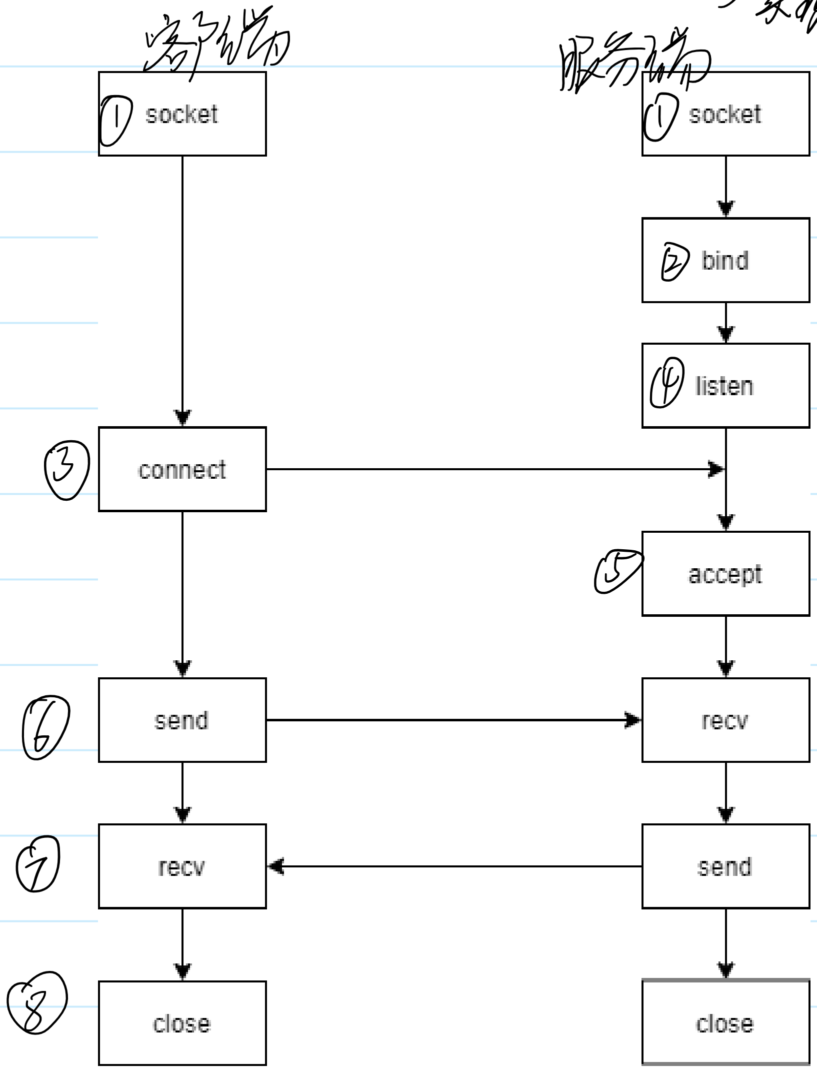
```c
int main(int argc, char *argv[])
{
    // ./0_gethostbyname www.baidu.com
    ARGS_CHECK(argc,2);
    struct hostent * phost = gethostbyname(argv[1]);
    if(phost == NULL){
        herror("gethostbyname");
        return -1;
    }
    printf("official name = %s\n", phost->h_name);
    for(int i = 0; phost->h_aliases[i] != NULL; ++i){
        printf("alias name = %s\n", phost->h_aliases[i]);
    }
    printf("addr type = %d\n",phost->h_addrtype);
    printf("addr length = %d\n",phost->h_length);
    for(int i = 0; phost->h_addr_list[i] != NULL; ++i){
        char buf[1024] = {0};
        inet_ntop(phost->h_addrtype,phost->h_addr_list[i],buf,1024);
        printf("ip = %s\n",buf);
    }
    return 0;
}
```

# tcp通信

2023年5月8日 9:54

用户编写应用层代码，使用了socket库.

→ 实现 传输层 以及更下层.



# 网络也是一种文件 socket

2023年5月8日 9:58

```
NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

地址类型

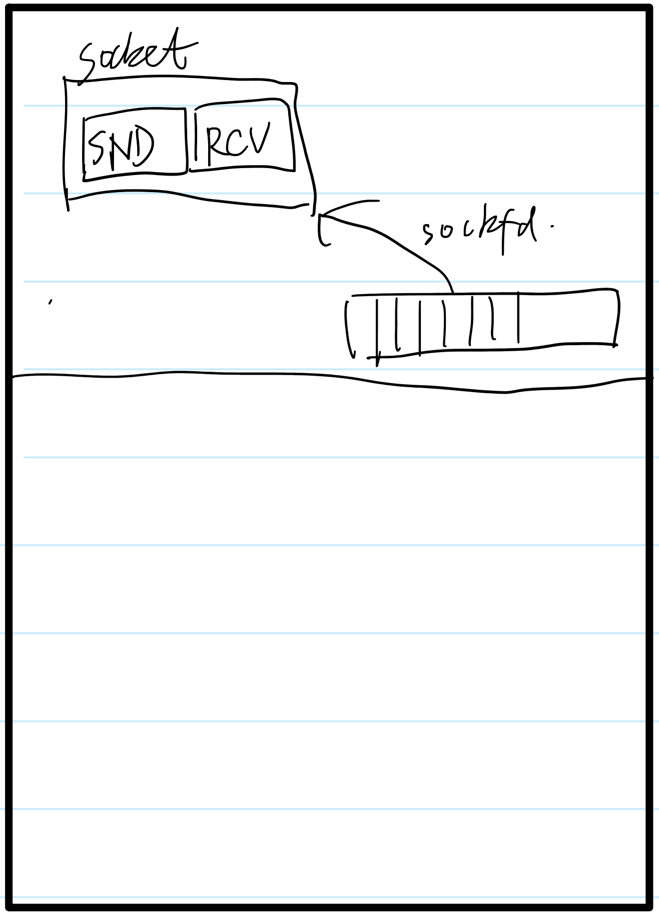
0

```
       AF_UNIX      Local communication
       AF_INET      IPv4 Internet protocols
       AF_INET6     IPv6 Internet protocols
```

TCP

```
           SOCK_STREAM     Provides sequenced, reliable, two-way, connection-based byte streams.
       SOCK_DGRAM      Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
```

UDP

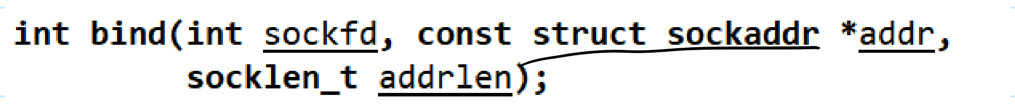


# bind 绑定地址

2023年5月8日　　10:07

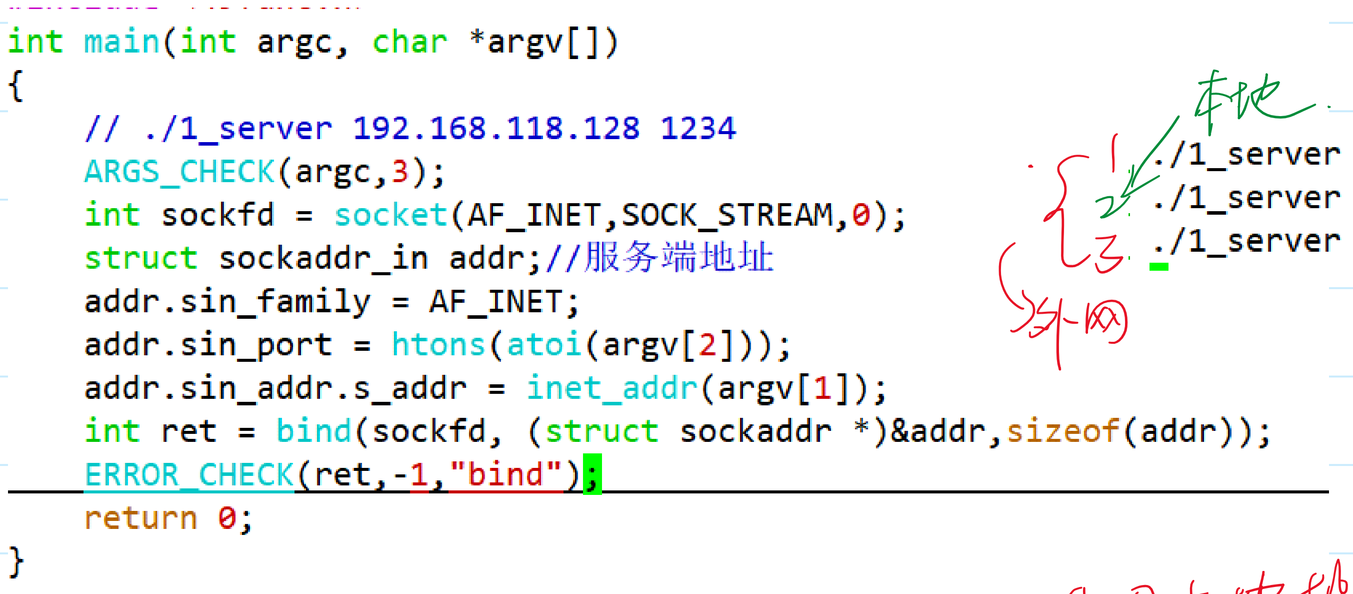
客户端 可以bind 不建议bind.

服务端 必须bind



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

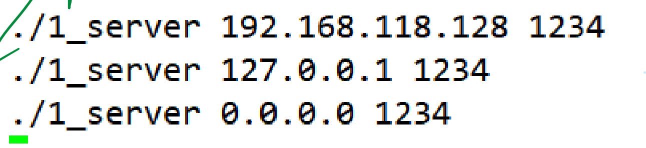
先创建 struct sockaddr_in
设置好内容
传参 & 再类型转换.



```
int main(int argc, char *argv[])
{
    // ./1_server 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_STREAM,0);
    struct sockaddr_in addr;//服务端地址
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = bind(sockfd, (struct sockaddr *)&addr,sizeof(addr));
    ERROR_CHECK(ret,-1,"bind");
    return 0;
}
```

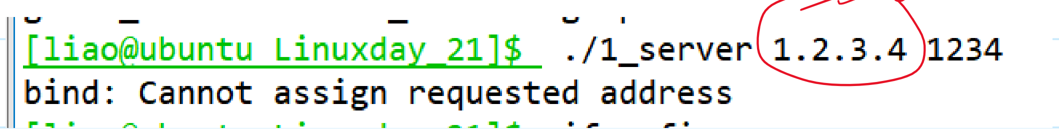
本地.

外网



```
1 ./1_server 192.168.118.128 1234
2 ./1_server 127.0.0.1 1234
3 ./1_server 0.0.0.0 1234
```

只能用本地地址



```
[liao@ubuntu Linuxday_21]$ ./1_server 1.2.3.4 1234
bind: Cannot assign requested address
```

# connect 建立连接

2023年5月8日 10:22

目标的ip端口

```
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

```
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./1_client 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_STREAM,0);
    struct sockaddr_in addr;//服务端地址
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = connect(sockfd,(struct sockaddr *)&addr,sizeof(addr));
    ERROR_CHECK(ret,-1,"connect");
    return 0;
}
```

```
int main(int argc, char *argv[])
{
    // ./1_server 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_STREAM,0);
    struct sockaddr_in addr;//服务端地址
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = bind(sockfd, (struct sockaddr *)&addr,sizeof(addr));
    ERROR_CHECK(ret,-1,"bind");
    sleep(100);
    return 0;
}
```

# tcpdump

2023年5月8日 10:27

Ncap ↓ windows抓包

Wireshark ↓ 筛选/分析

tcpdump

Linux抓包

```
tcpdump -n -i lo port 1234 -w /home/liao/49test.cap
```

$\underline{\text{port 1234}} \to$ 筛选表达式

① su 切到root

② -i 指定网卡 -i any

③ -w. 保存文件.

# 解决网络问题的一般流程

2023年5月8日 10:52

1 netstat命令 netstat -an 观察连接的状态

2 tcpdump 抓包 -w 保存抓包数据

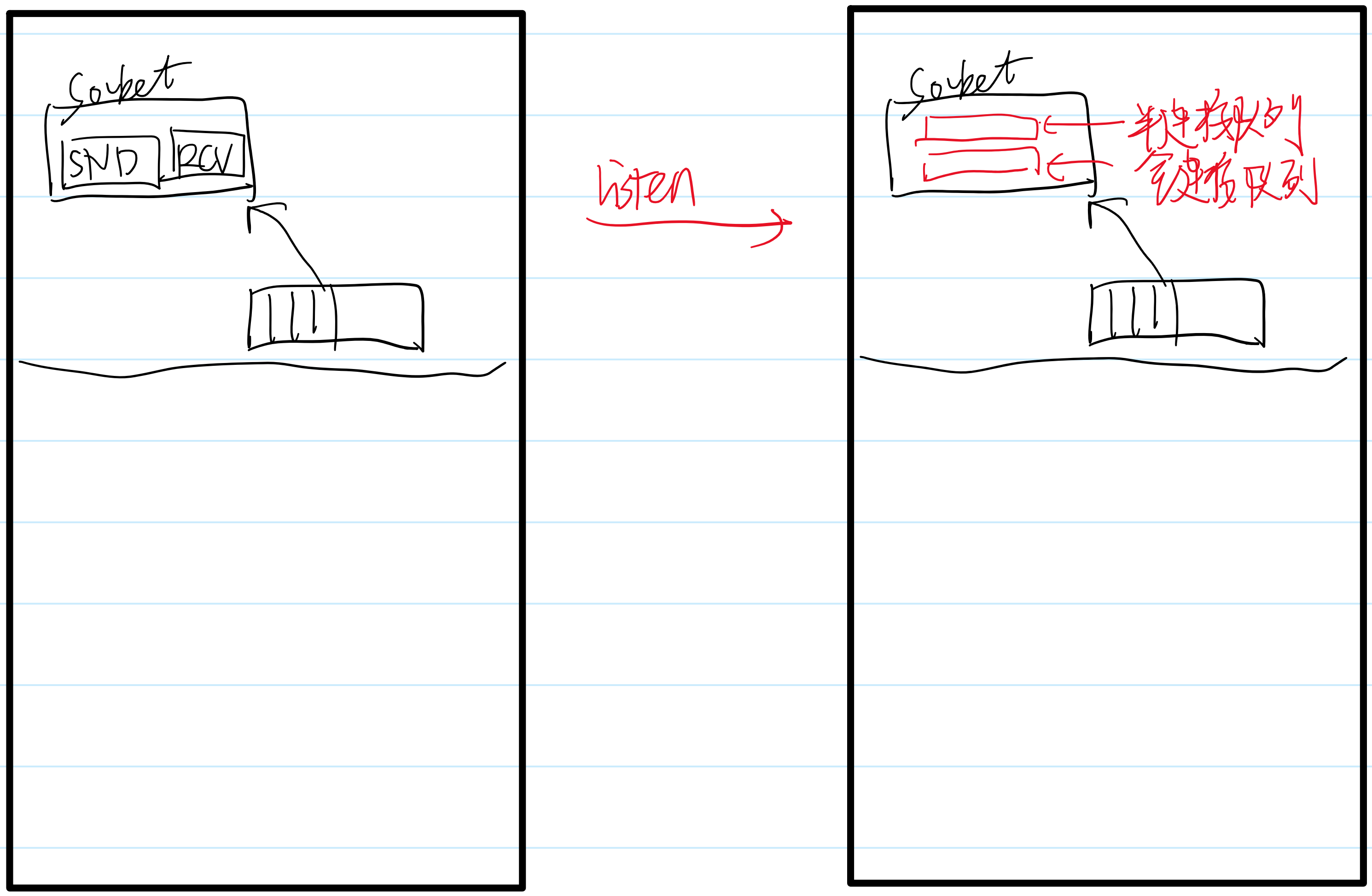
3 用wireshark打开抓包数据分析

# listen

2023年5月8日 10:57

```
int listen(int sockfd, int backlog);
```

← 服务端 进入监听状态

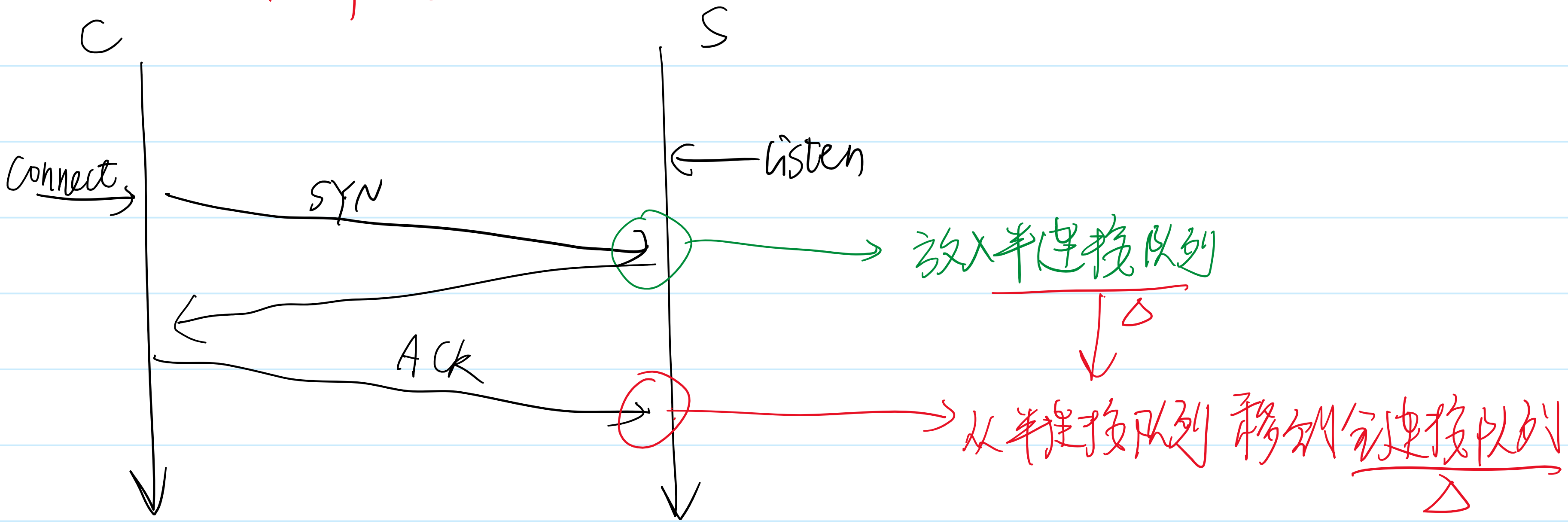


# listen之后

2023年5月8日 11:01

listen之后，socket 不能发送和接收数据

△ 只能新建连接



# DDOS攻击

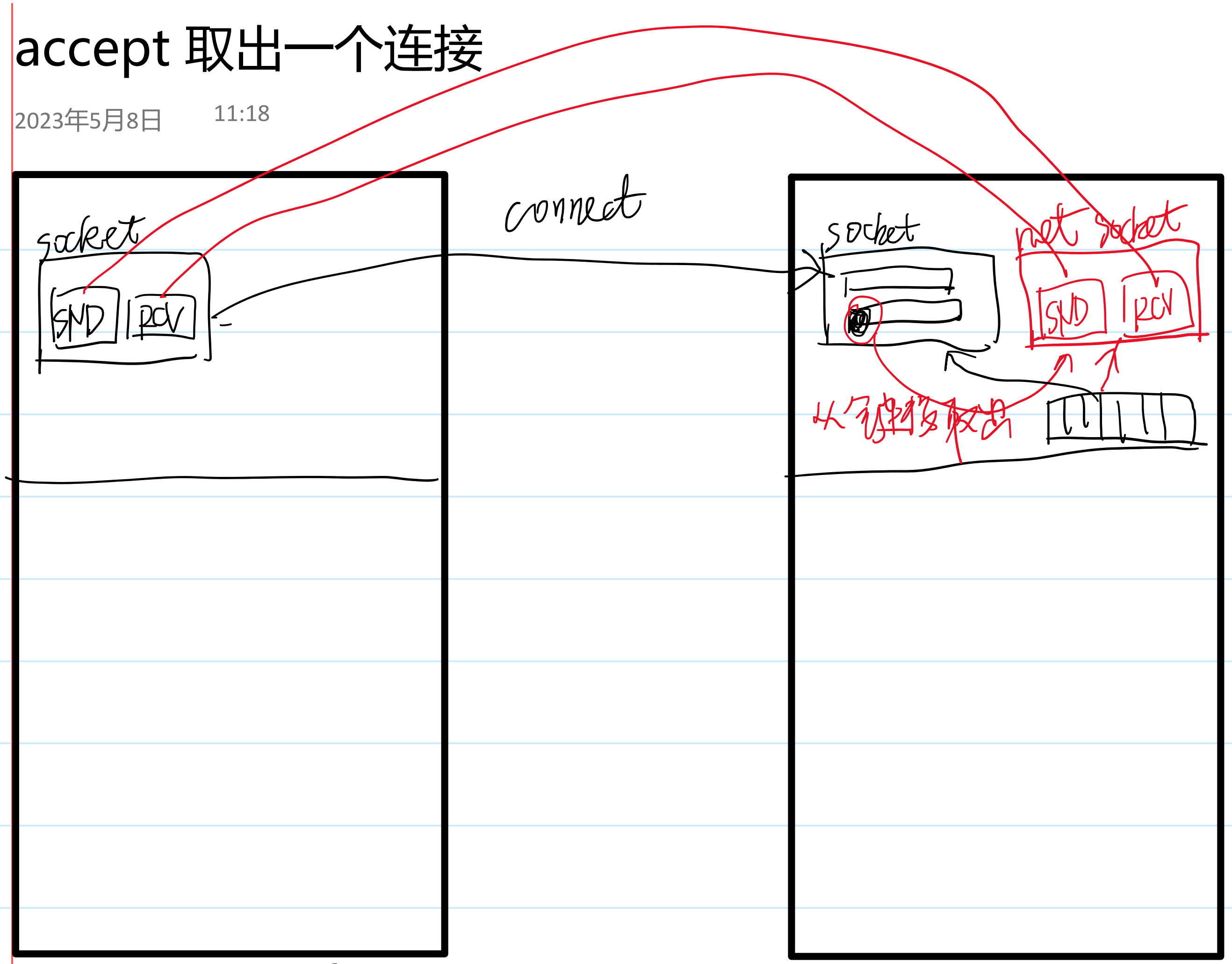
2023年5月8日 11:09

SYN 泛洪 只发第一次握手 $\longrightarrow$ 服务端的半连接队列满了。

肉鸡

# accept 取出一个连接

2023年5月8日 11:18



在客户端 connect完成之后.
服务端调用 accept
从全连接队列中取出连接,
构建新的文件对象
(net socket)

net socket 和客户端直接
通信

监听的

NULL

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

新的 net socket

用来获取 client的地址

用来获取客户端地址长度
addrlen指向内容必须有一个初始值.

# accept的特点

2023年5月8日 11:28

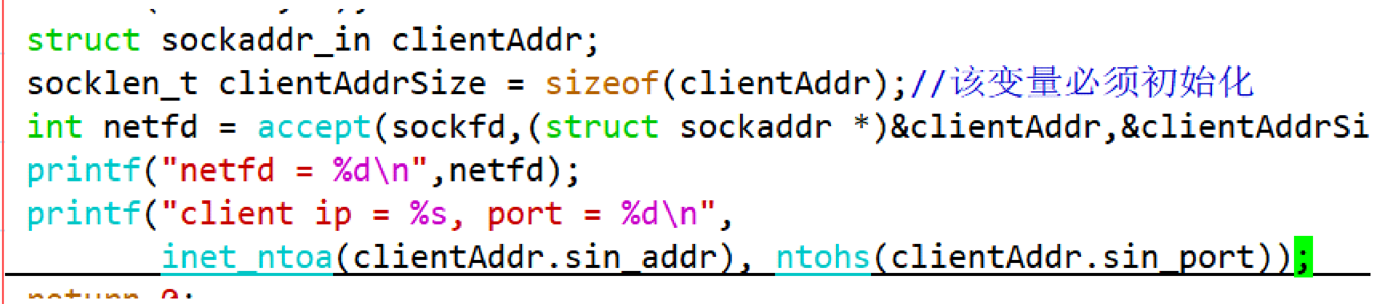
accept时，全连接队列为空，进程会阻塞

accept的性质和read管道很像.

读阻塞. $\longrightarrow$ select

# accept的代码

2023年5月8日 11:37



```c
struct sockaddr_in clientAddr;
socklen_t clientAddrSize = sizeof(clientAddr);//该变量必须初始化
int netfd = accept(sockfd,(struct sockaddr *)&clientAddr,&clientAddrSi
printf("netfd = %d\n",netfd);
printf("client ip = %s, port = %d\n",
       inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
```

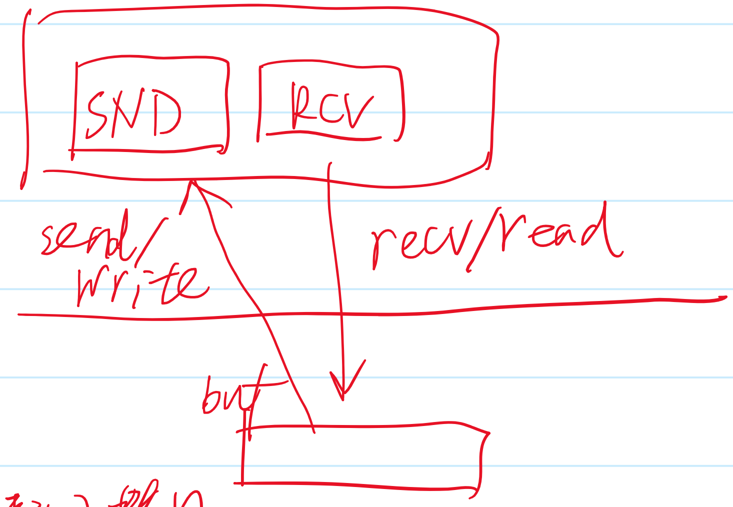
# send和recv

2023年5月8日 11:39

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

ssize_t write(int fd, const void *buf, size_t count);

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

ssize_t read(int fd, void *buf, size_t count);
```

send只是一个特殊的write，只能对socket使用.

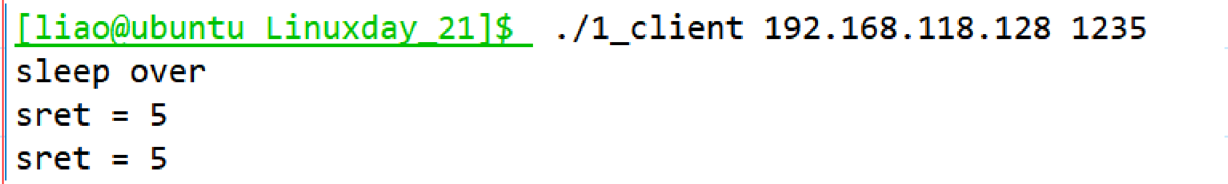
read — — — — — 的read，只能对socket使用.



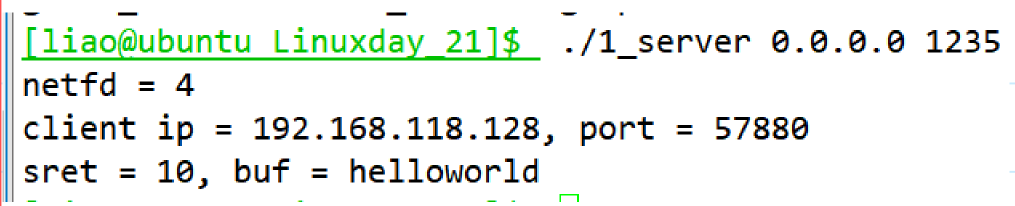
send/recv只是把数据在buf和socket之间来回拷贝.
真正的发送和接收行为是内核协议栈完成.

# tcp是一种流式协议

2023年5月8日 11:54



```
[liao@ubuntu Linuxday_21]$  ./1_client 192.168.118.128 1235
sleep over
sret = 5
sret = 5
```



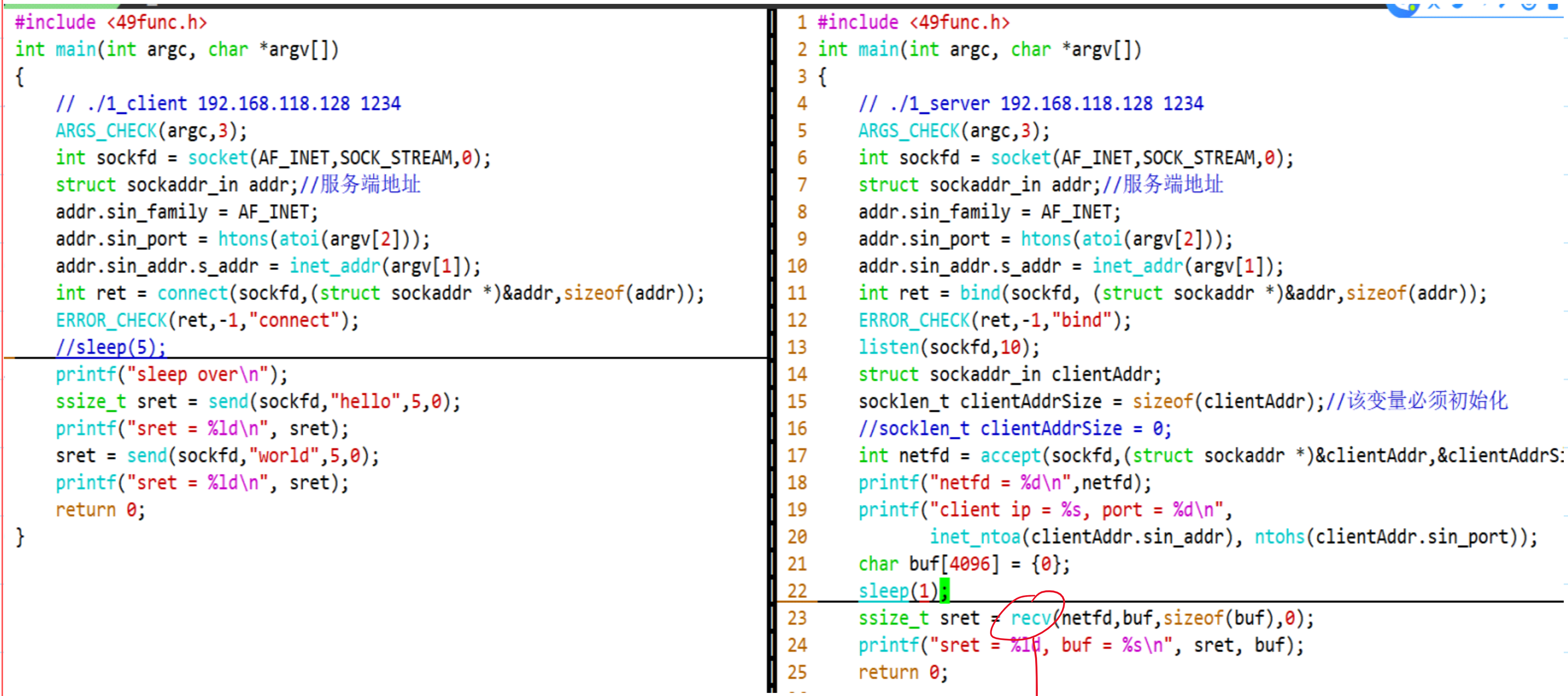
```
[liao@ubuntu Linuxday_21]$  ./1_server 0.0.0.0 1235
netfd = 4
client ip = 192.168.118.128, port = 57880
sret = 10, buf = helloworld
```

send和recv不是一一对应

tcp数据没有边界.

# 代码示例

2023年5月8日 11:55



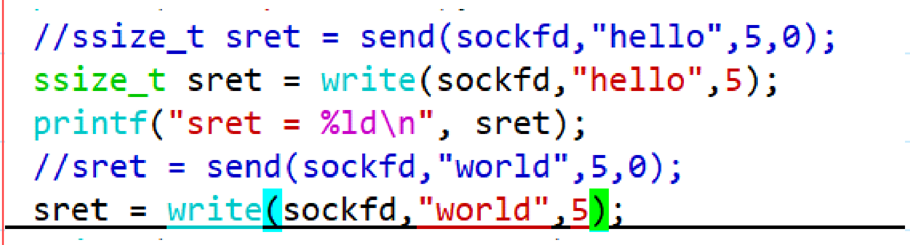
```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./1_client 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_STREAM,0);
    struct sockaddr_in addr;//服务端地址
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = connect(sockfd,(struct sockaddr *)&addr,sizeof(addr));
    ERROR_CHECK(ret,-1,"connect");
    //sleep(5);
    printf("sleep over\n");
    ssize_t sret = send(sockfd,"hello",5,0);
    printf("sret = %ld\n", sret);
    sret = send(sockfd,"world",5,0);
    printf("sret = %ld\n", sret);
    return 0;
}
```

```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./1_server 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_STREAM,0);
    struct sockaddr_in addr;//服务端地址
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = bind(sockfd, (struct sockaddr *)&addr,sizeof(addr));
    ERROR_CHECK(ret,-1,"bind");
    listen(sockfd,10);
    struct sockaddr_in clientAddr;
    socklen_t clientAddrSize = sizeof(clientAddr);//该变量必须初始化
    //socklen_t clientAddrSize = 0;
    int netfd = accept(sockfd,(struct sockaddr *)&clientAddr,&clientAddrS
    printf("netfd = %d\n",netfd);
    printf("client ip = %s, port = %d\n",
           inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
    char buf[4096] = {0};
    sleep(1);
    ssize_t sret = recv(netfd,buf,sizeof(buf),0);
    printf("sret = %ld, buf = %s\n", sret, buf);
    return 0;
```

会阻塞，可用select管理

# read/write 等价于 recv/send

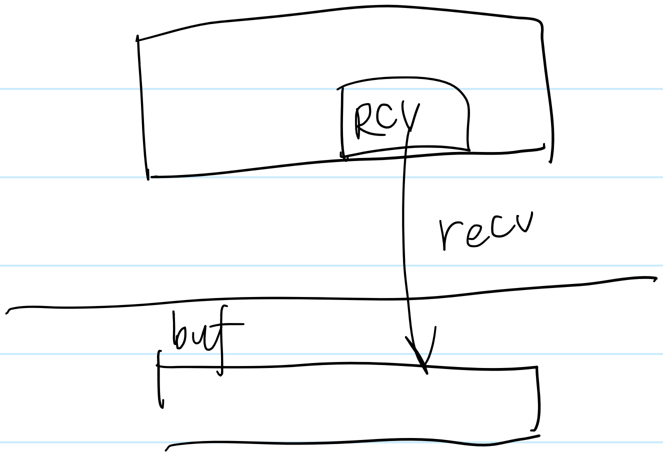
2023年5月8日 14:33



```
//ssize_t sret = send(sockfd,"hello",5,0);
ssize_t sret = write(sockfd,"hello",5);
printf("sret = %ld\n", sret);
//sret = send(sockfd,"world",5,0);
sret = write(sockfd,"world",5);
```

# recv的注意事项

2023年5月8日 14:34



recv(sockfd, buf, count, 0)

RCV为空，recv会阻塞。

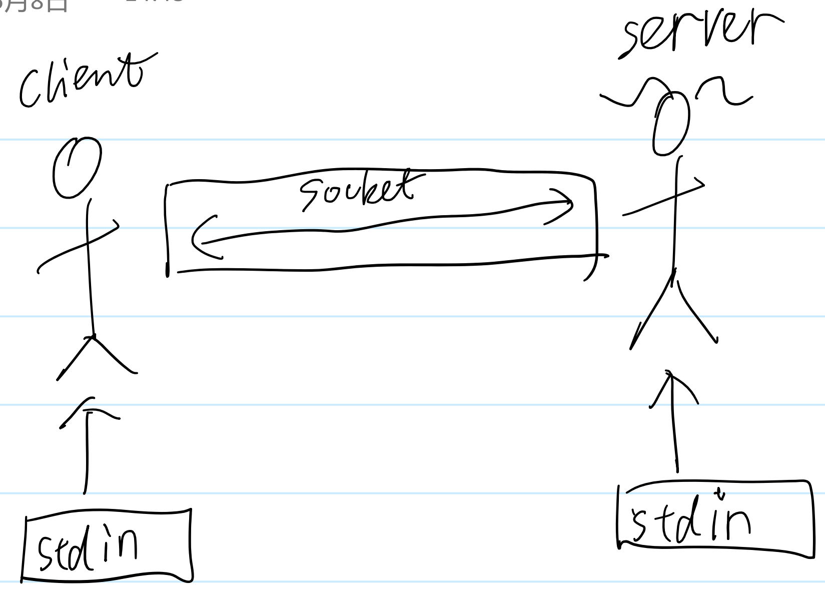
RCV非空：{ recv的ret [0,count)
recv的ret 为0. 对面close
recv的ret 为count

send(sockfd, buf, count, 0)

对面
recv(netfd, buf, count, 0) [0, count]

# 利用网络实现即时聊天

2023年5月8日 14:48



select

① 监听集合 rdset

② 初始化

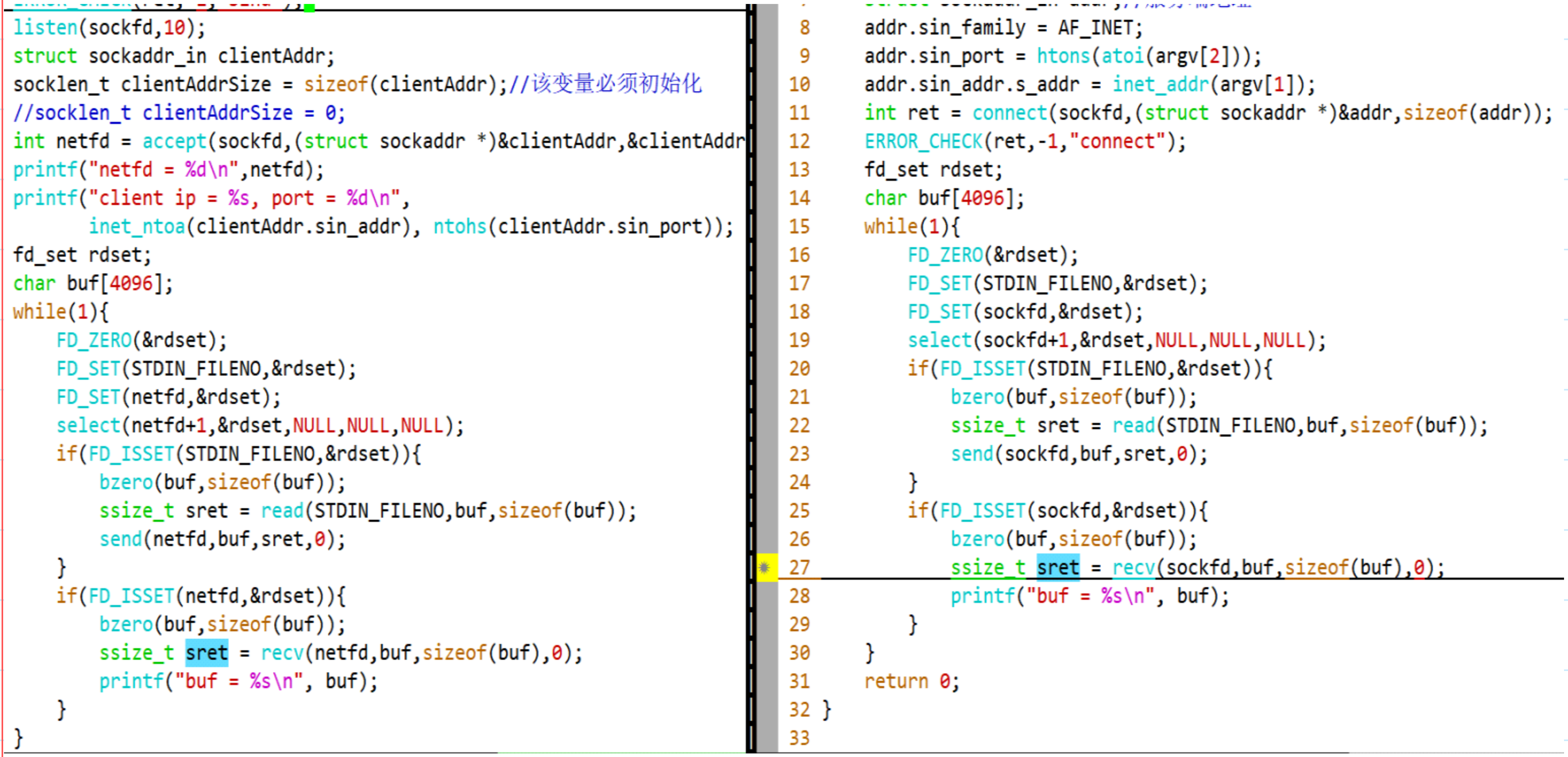
③ 增加监听

④ select

⑤ FD_ISSET { read stdin
recv socket

# 即时聊天的代码

2023年5月8日 15:09



```c
listen(sockfd,10);
struct sockaddr_in clientAddr;
socklen_t clientAddrSize = sizeof(clientAddr);//该变量必须初始化
//socklen_t clientAddrSize = 0;
int netfd = accept(sockfd,(struct sockaddr *)&clientAddr,&clientAddr
printf("netfd = %d\n",netfd);
printf("client ip = %s, port = %d\n",
        inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
fd_set rdset;
char buf[4096];
while(1){
    FD_ZERO(&rdset);
    FD_SET(STDIN_FILENO,&rdset);
    FD_SET(netfd,&rdset);
    select(netfd+1,&rdset,NULL,NULL,NULL);
    if(FD_ISSET(STDIN_FILENO,&rdset)){
        bzero(buf,sizeof(buf));
        ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
        send(netfd,buf,sret,0);
    }
    if(FD_ISSET(netfd,&rdset)){
        bzero(buf,sizeof(buf));
        ssize_t sret = recv(netfd,buf,sizeof(buf),0);
        printf("buf = %s\n", buf);
    }
}
```

```c
    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[2]));
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = connect(sockfd,(struct sockaddr *)&addr,sizeof(addr));
    ERROR_CHECK(ret,-1,"connect");
    fd_set rdset;
    char buf[4096];
    while(1){
        FD_ZERO(&rdset);
        FD_SET(STDIN_FILENO,&rdset);
        FD_SET(sockfd,&rdset);
        select(sockfd+1,&rdset,NULL,NULL,NULL);
        if(FD_ISSET(STDIN_FILENO,&rdset)){
            bzero(buf,sizeof(buf));
            ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
            send(sockfd,buf,sret,0);
        }
        if(FD_ISSET(sockfd,&rdset)){
            bzero(buf,sizeof(buf));
            ssize_t sret = recv(sockfd,buf,sizeof(buf),0);
            printf("buf = %s\n", buf);
        }
    }
    return 0;
}
```

# 写端关闭的时候

2023年5月8日　　15:09

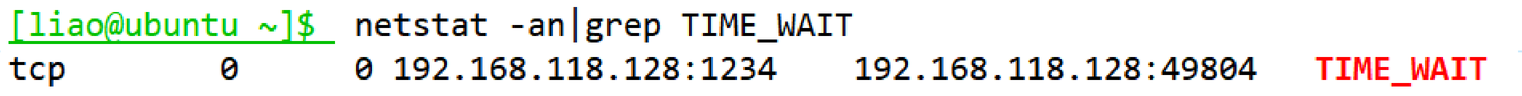
```
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
^C
```

$$\begin{tikzcd}\text{写端关闭} \arrow[r] & \text{对面读端就绪} \arrow[r] & \text{监听读端select就绪}\end{tikzcd}$$

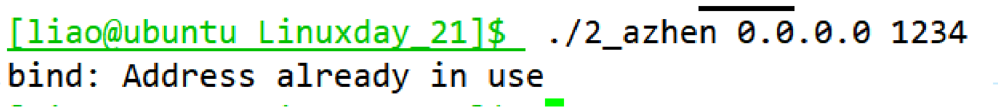
```
while(){
    select
    recv
}
```

# time_wait的影响

2023年5月8日 15:21



```
[liao@ubuntu ~]$  netstat -an|grep TIME_WAIT
tcp        0      0 192.168.118.128:1234    192.168.118.128:49804   TIME_WAIT
```



```
[liao@ubuntu Linuxday_21]$  ./2_azhen 0.0.0.0 1234
bind: Address already in use
```

只要客户端端口号不固定，TIME_WAIT意义不大。

$\uparrow$

可无视之。

# 更改socket的属性

2023年5月8日 15:23

SOL_SOCKET

SO_REUSEADDR

```
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

参数的首地址和长度

int 0假 / 1真

```
int reuse = 1; // SO_REUSEADDR属性的参数
int ret = setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&reuse,sizeof(int));
ERROR_CHECK(ret,-1,"setsockopt");
```

```
[liao@ubuntu ~]$ netstat -an|grep TIME_WAIT
tcp        0      0 192.168.118.128:1234    192.168.118.128:48142   TIME_WAIT
tcp        0      0 192.168.118.128:1234    192.168.118.128:48132   TIME_WAIT
tcp        0      0 192.168.118.128:1234    192.168.118.128:34950   TIME_WAIT
```

# 让服务端支持断线重连

2023年5月8日 15:49

服务端 sockfd → accept
netfd → recv
STDIN_FILENO → read

} 读阻塞 . select

```
int main() {
    socket → setsockopt → bind → listen.
    while(1){
        select();
        if(FD_ISSET(sockfd)){ accept... }
    }
```

→

| 第一次 | 第二次 |
|---|---|
| sockfd | netfd |
| | STDIN_FILENO |

2023年5月8日　　16:40

```c
char buf[4096];
fd_set rdset;//每次select传入的参数
fd_set monitorset;//下次select的监听集合
FD_ZERO(&monitorset);
FD_SET(sockfd,&monitorset);
int netfd;
while(1){
    memcpy(&rdset,&monitorset,sizeof(fd_set));
    select(10,&rdset,NULL,NULL,NULL);//select调用只会修改rdset， 不修改monitorset
    if(FD_ISSET(sockfd,&rdset)){
        struct sockaddr_in clientAddr;
        socklen_t clientAddrSize = sizeof(clientAddr);//该变量必须初始化
        //socklen_t clientAddrSize = 0;
        netfd = accept(sockfd,(struct sockaddr *)&clientAddr,&clientAddrSize);
        printf("netfd = %d\n",netfd);
        printf("client ip = %s, port = %d\n",
               inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
        //希望服务端在连上一个客户端之后
        //可以和这个客户端聊天
        //不与其他客户端建立连接
        FD_CLR(sockfd,&monitorset);
        FD_SET(STDIN_FILENO,&monitorset);
        FD_SET(netfd,&monitorset);
    }
```

2023年5月8日　　16:40

```c
if(FD_ISSET(STDIN_FILENO,&rdset)){
    bzero(buf,sizeof(buf));
    ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
    if(sret == 0){
        send(netfd,"nishigehaoren",13,0);
        FD_SET(sockfd,&monitorset);
        FD_CLR(STDIN_FILENO,&monitorset);
        FD_CLR(netfd,&monitorset);
        close(netfd);
        printf("woyoudanshenle\n");
        continue;
    }
    send(netfd,buf,sret,0);
}
if(FD_ISSET(netfd,&rdset)){
    bzero(buf,sizeof(buf));
    ssize_t sret = recv(netfd,buf,sizeof(buf),0);
    if(sret == 0){//对方断开连接
        FD_SET(sockfd,&monitorset);
        FD_CLR(STDIN_FILENO,&monitorset);
        FD_CLR(netfd,&monitorset);
        close(netfd);
        printf("wohuihaohaode\n");
        continue;
    }
    printf("buf = %s\n", buf);
```

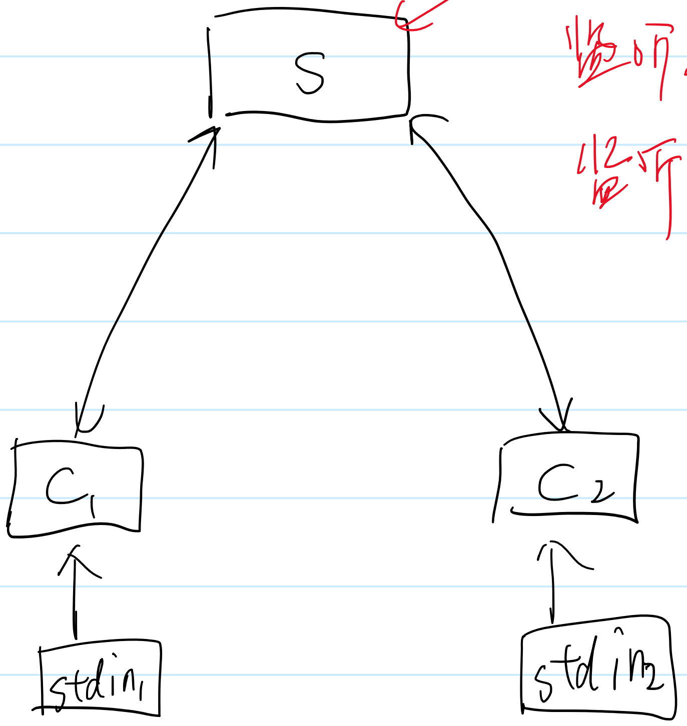
# 如果用微信聊天

2023年5月8日 15:58

不监听 stdin

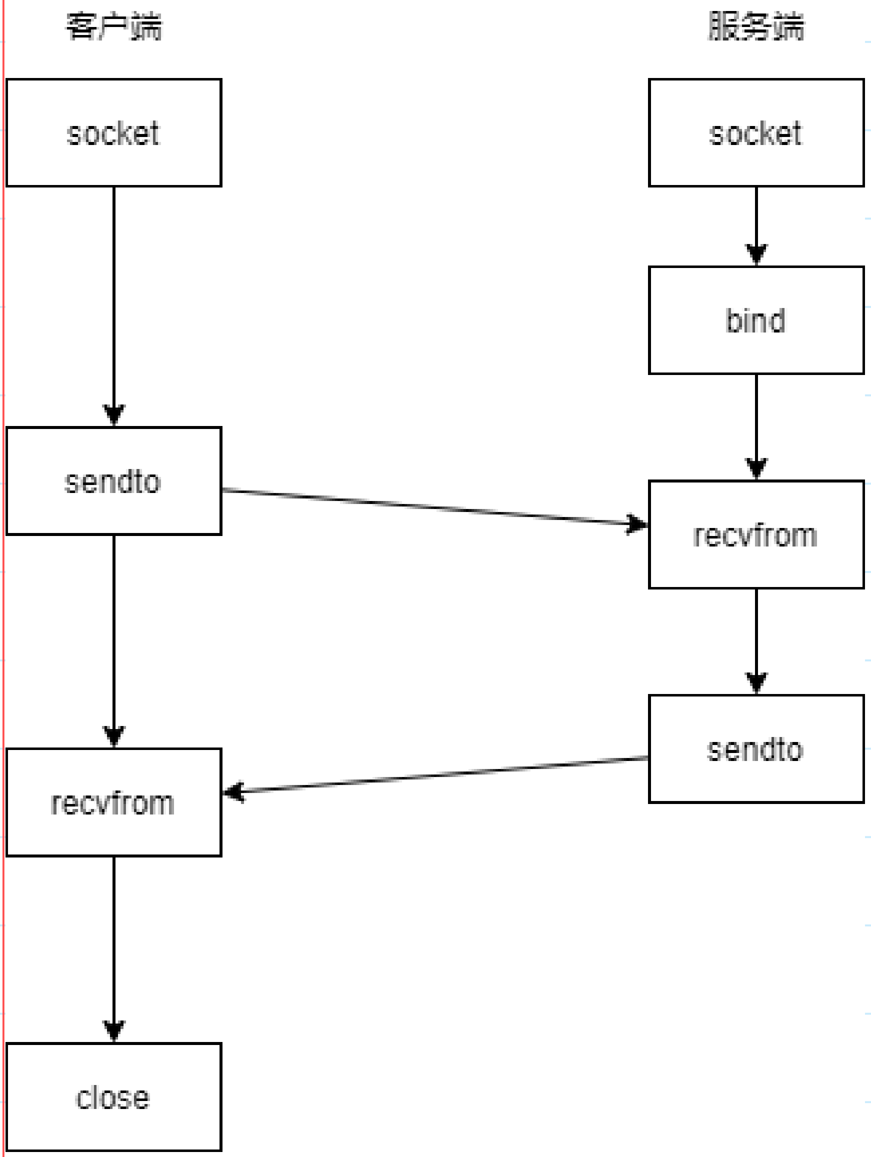
监听sockfd，若就绪 accept 得到一个新netfd

监听多个netfd，若就绪，转发给其他netfd.



# UDP

2023年5月8日 16:58



socket(AF_INET, SOCK_DGRAM, 0)

和tcp不同

tcp 和 udp 能否bind同一个端口

可以bind

# sendto和recvfrom

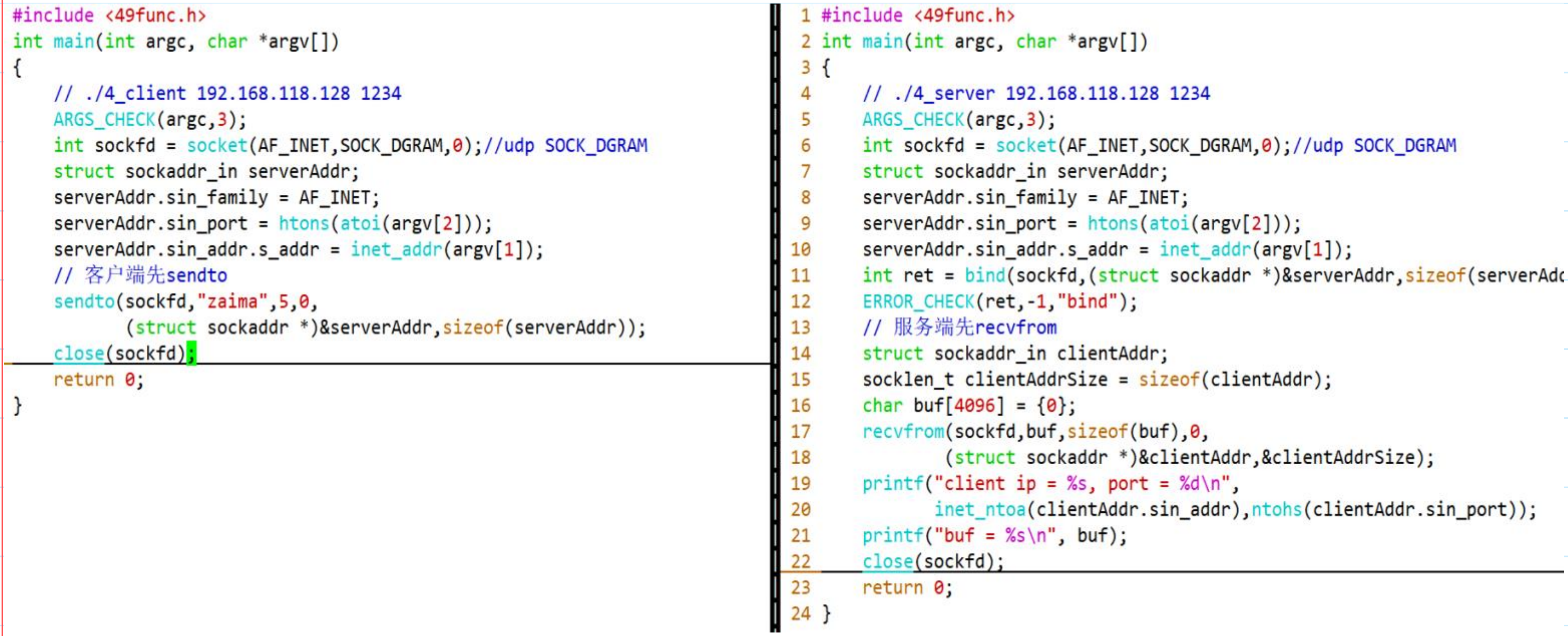
2023年5月8日 17:03

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

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

udp 必须 客户端用sendto, 服务端用 recvfrom

# udp通信的例子

2023年5月8日 17:09

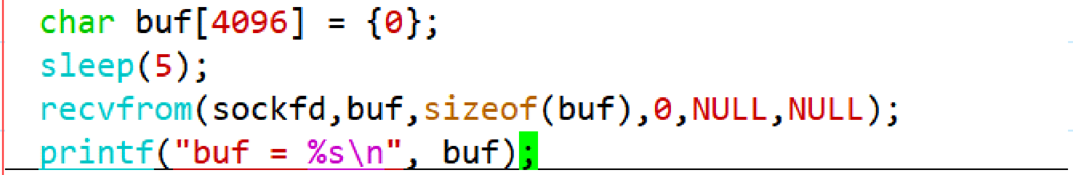


```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./4_client 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_DGRAM,0);//udp SOCK_DGRAM
    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(atoi(argv[2]));
    serverAddr.sin_addr.s_addr = inet_addr(argv[1]);
    // 客户端先sendto
    sendto(sockfd,"zaima",5,0,
           (struct sockaddr *)&serverAddr,sizeof(serverAddr));
    close(sockfd);
    return 0;
}
```

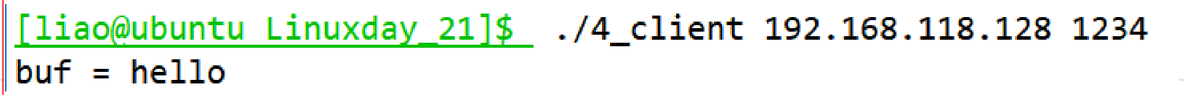
```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./4_server 192.168.118.128 1234
    ARGS_CHECK(argc,3);
    int sockfd = socket(AF_INET,SOCK_DGRAM,0);//udp SOCK_DGRAM
    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(atoi(argv[2]));
    serverAddr.sin_addr.s_addr = inet_addr(argv[1]);
    int ret = bind(sockfd,(struct sockaddr *)&serverAddr,sizeof(serverAdd
    ERROR_CHECK(ret,-1,"bind");
    // 服务端先recvfrom
    struct sockaddr_in clientAddr;
    socklen_t clientAddrSize = sizeof(clientAddr);
    char buf[4096] = {0};
    recvfrom(sockfd,buf,sizeof(buf),0,
             (struct sockaddr *)&clientAddr,&clientAddrSize);
    printf("client ip = %s, port = %d\n",
           inet_ntoa(clientAddr.sin_addr),ntohs(clientAddr.sin_port));
    printf("buf = %s\n", buf);
    close(sockfd);
    return 0;
}
```

# UDP的消息是有边界的

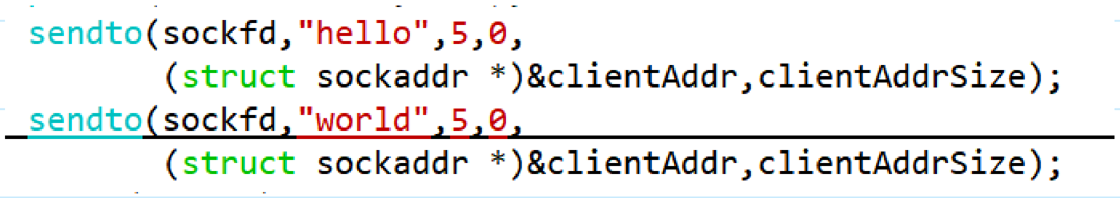
2023年5月8日 17:23



```c
char buf[4096] = {0};
sleep(5);
recvfrom(sockfd,buf,sizeof(buf),0,NULL,NULL);
printf("buf = %s\n", buf);
```



```
[liao@ubuntu Linuxday_21]$  ./4_client 192.168.118.128 1234
buf = hello
```



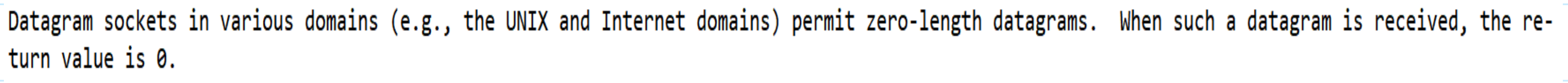
```c
sendto(sockfd,"hello",5,0,
       (struct sockaddr *)&clientAddr,clientAddrSize);
sendto(sockfd,"world",5,0,
       (struct sockaddr *)&clientAddr,clientAddrSize);
```

# 使用udp的即时聊天

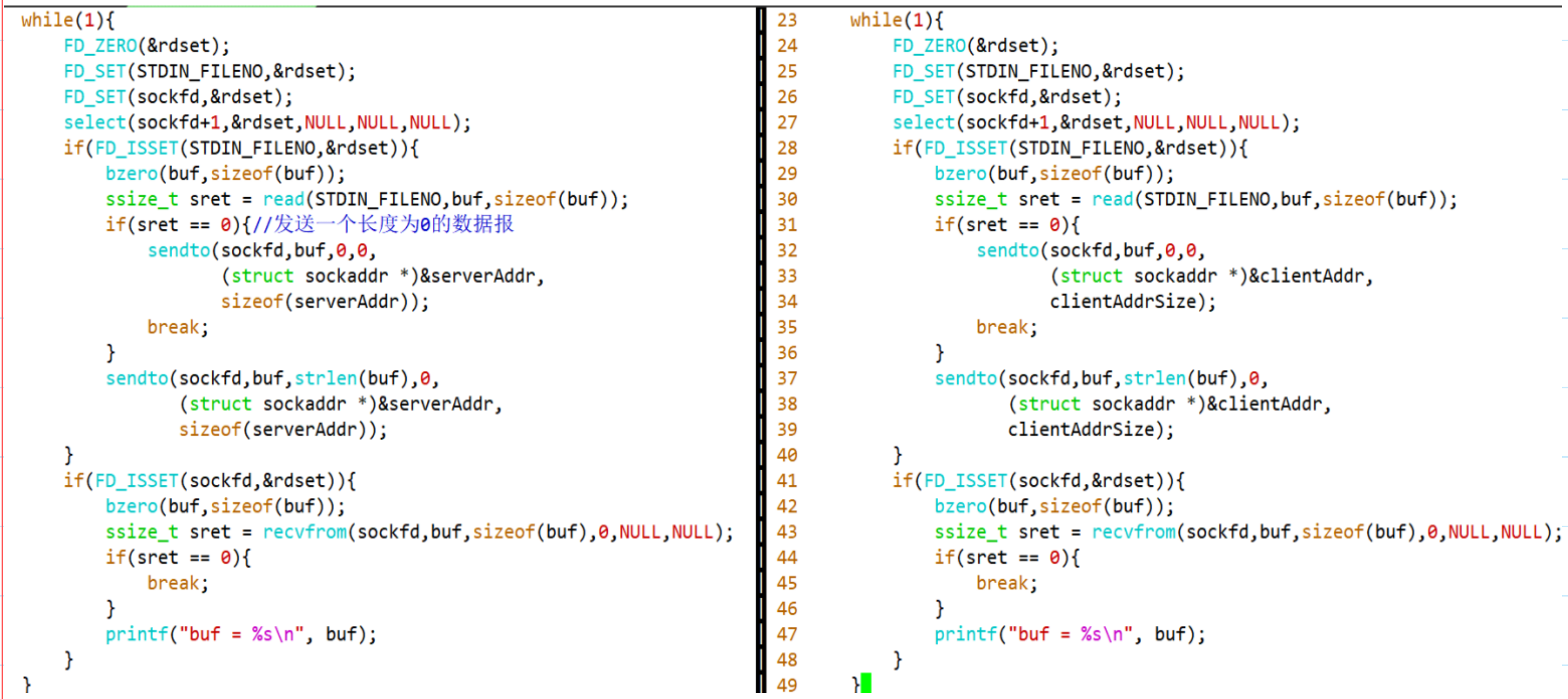
2023年5月8日 17:30

① 服务端先recvfrom，获取客户端的ip和端口

② 聊天终止，需要手动实现.


Datagram sockets in various domains (e.g., the UNIX and Internet domains) permit zero-length datagrams. When such a datagram is received, the re-
turn value is 0.

2023年5月8日 17:49



```c
while(1){
    FD_ZERO(&rdset);
    FD_SET(STDIN_FILENO,&rdset);
    FD_SET(sockfd,&rdset);
    select(sockfd+1,&rdset,NULL,NULL,NULL);
    if(FD_ISSET(STDIN_FILENO,&rdset)){
        bzero(buf,sizeof(buf));
        ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
        if(sret == 0){//发送一个长度为0的数据报
            sendto(sockfd,buf,0,0,
                   (struct sockaddr *)&serverAddr,
                   sizeof(serverAddr));
            break;
        }
        sendto(sockfd,buf,strlen(buf),0,
               (struct sockaddr *)&serverAddr,
               sizeof(serverAddr));
    }
    if(FD_ISSET(sockfd,&rdset)){
        bzero(buf,sizeof(buf));
        ssize_t sret = recvfrom(sockfd,buf,sizeof(buf),0,NULL,NULL);
        if(sret == 0){
            break;
        }
        printf("buf = %s\n", buf);
    }
}
```

```c
while(1){
    FD_ZERO(&rdset);
    FD_SET(STDIN_FILENO,&rdset);
    FD_SET(sockfd,&rdset);
    select(sockfd+1,&rdset,NULL,NULL,NULL);
    if(FD_ISSET(STDIN_FILENO,&rdset)){
        bzero(buf,sizeof(buf));
        ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
        if(sret == 0){
            sendto(sockfd,buf,0,0,
                   (struct sockaddr *)&clientAddr,
                   clientAddrSize);
            break;
        }
        sendto(sockfd,buf,strlen(buf),0,
               (struct sockaddr *)&clientAddr,
               clientAddrSize);
    }
    if(FD_ISSET(sockfd,&rdset)){
        bzero(buf,sizeof(buf));
        ssize_t sret = recvfrom(sockfd,buf,sizeof(buf),0,NULL,NULL);
        if(sret == 0){
            break;
        }
        printf("buf = %s\n", buf);
    }
}
```