

## Linux 操作系统

## 15 网络编程

主讲：杨东平  
中国矿大计算机学院

## 进程通信：套接字(socket)

套接字也是一种进程间通信机制，可用于不同机器间的进程通信

套接字起初是由 Unix 系统的 BSD 分支开发出来的，但现在一般可以移植到其它类 Unix 系统上：Linux 和 System V 的变种都支持套接字

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日4时57分 2

## 有关套接字的头文件和函数

## 头文件

- 数据类型：sys/types.h
- 函数定义：sys/socket.h

## 函数：

- socket()
- bind()
- listen()
- connect()
- accept()
- recv() / recvfrom()
- send() / sendto()
- close()

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日4时58分 3

## socket 函数

Socket描述符：一个int型的变量，标志连接的一方。

原型：int socket(int domain, int type, int protocol);

## 参数：

- domain：程序采用的通讯协议
  - AF\_UNIX：只用于单一的 Unix 系统进程间通信
  - AF\_INET：用于 Internet 通信
- type：采用的通讯协议
  - SOCK\_STREAM：用 TCP 协议
  - SOCK\_DGRAM：用 UDP 协议
- protocol：由于指定了 type，此值一般为 0

## 返回值：

- 成功：Socket 描述符
- 失败：-1，可用 errno 查看出错的详细情况

## 示例：

- TCP：sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
- UDP：sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日4时58分 4

## 通用地址结构：sockaddr

## 定义：

```
struct sockaddr {
    unsigned short sa_family; /*地址族, AF_XXX */
    char sa_data[14]; /* 14字节, 包含套接字中的目标地址和端口信息 */
};
```

## 说明：

- sa\_family：2 字节地址族，包括三种：
  - AF\_INET：IPv4 地址信息(Socket编程只能用它)
  - AF\_INET6：IPv6 地址信息
  - AF\_UNSPEC：使用适用于指定主机名和服务名且适合任何协议族的地址。如果主机既有支持 IPv6 地址，也支持 IPv4 地址，则返回 sockaddr\_in6 结构，否则返回 sockaddr\_in 结构

用于 bind、connect、recvfrom、sendto 等函数的参数，指明地址信息

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日4时58分 5

## IPv4 地址结构：sockaddr\_in

头文件：netinet/in.h  
或 arpa/inet.h

该结构体解决了 sockaddr 的缺陷，把 port 和 addr 分开储存在两个变量中：

```
struct sockaddr_in {
    short int sin_family; /* 协议族, socket 中只能是 AF_INET */
    unsigned short int sin_port; /* 端口号 */
    struct in_addr sin_addr; /* IP 地址 */
    unsigned char sin_zero[8]; /* 使 sockaddr_in 相同大小而保留 */
};
```

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日4时58分 6

**存储 IP 地址的数据结构：in\_addr**

```
struct in_addr
{
    union
    {
        struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b; // 4 字节 IPv4 地址
        struct { u_short s_w1,s_w2; } S_un_w; // 两个短整数 IPv4 地址
        u_long S_addr; // 32 位整数 IPv4 地址
    } S_un;
    #define s_addr S_un.S_addr
};
```

Ø 有三种表达方式：

- ✓ 1) 用四个字节来表示 IP 地址的四个数字
- ✓ 2) 用两个双字节来表示 IP 地址
- ✓ 3) 用一个长整型来表示 IP 地址

网络安全与网络工程系系东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 7

**IPv6 地址结构**

```
struct sockaddr_in6 {
    sa_family_t    sin6_family; /* AF_INET6 */
    in_port_t      sin6_port;    /* port number */
    uint32_t       sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;    /* IPv6 address */
    uint32_t       sin6_scope_id; /* Scope ID (new in 2.4) */
};

struct in6_addr {
    unsigned char  s6_addr[16]; /* IPv6 address */
};
```

网络安全与网络工程系系东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 8

**IPv4 地址函数：inet\_addr、inet\_ntoa**

Ø 这两个函数互为反函数

Ø 原型：unsigned long inet\_addr(const char \*cp);

- ✓ inet\_addr 函数是给 in\_addr 赋值的一种最简单方法，它可以把一个点分十进制 IP 地址字符串转换为 in\_addr 类型，如：addrto.sin\_addr.s\_addr=inet\_addr("192.168.0.2");

Ø 原型：char\* inet\_ntoa(struct in\_addr in);

- ✓ inet\_ntoa 函数将一个 32 位数字表示的 IP 地址转换成点分十进制 IP 地址字符串
- ✓ 由于字节序的问题，不主张使用此函数

网络安全与网络工程系系东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 9

**字节顺序(字节序)**

Ø 字节顺序：指内存多于一个字节类型的数据在内存中的存放顺序，包括：

- ✓ 小端字节序(Little endian)：低字节数据存放在内存低地址处，高字节数据存放在内存高地址处
- ✓ 大端字节序(Big endian)：高字节数据存放在低地址处，低字节数据存放在高地址处

Ø 网络字节顺序(NBO, Network Byte Order)：

- ✓ TCP/IP 中规定的一种数据表示格式，它与具体的 CPU 类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释

✓ 采用 big endian 排序方式

Ø 一般地，基于 x86 平台的 PC 机是小端字节序的，而有的嵌入式平台则是大端字节序的

Ø 网络数据收发：

- ✓ 程序中发数据包时，需要将主机字节序(HBO, Host Byte Order)转换为网络字节序
- ✓ 接收数据包时需要将网络字节序转换为主机字节序

网络安全与网络工程系系东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 10

**字节顺序转换函数**

Ø 头文件：arpa/inet.h

Ø 主机字节顺序转换为网络字节顺序

- ✓ uint32\_t htonl(uint32\_t hostlong);  
F 将无符号长整数从主机字节序转换为网络字节序
- ✓ uint16\_t htons(uint16\_t hostshort);  
F 将无符号短整数从主机字节序转换为网络字节序

Ø 网络字节顺序转换为主机字节顺序

- ✓ uint32\_t ntohl(uint32\_t netlong);  
F 将无符号长整数从网络字节序转换为主机字节序
- ✓ uint16\_t ntohs(uint16\_t netshort);  
F 将无符号短整数从网络字节序转换为主机字节序

网络安全与网络工程系系东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 11

**程序中通常用 sockaddr\_in 结构**

struct sockaddr\_in my\_addr;

```
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(3490);
my_addr.sin_addr.s_addr = inet_addr("132.241.5.10");
bzero(&(my_addr.sin_zero), 8);
// 原型：void bzero(void *, int n);
// 头文件：string.h
// 功能：置字节字符串 s 的前 n 个字节为零且包括 '\0'
```

Ø 注意：sin\_addr.s\_addr

- ✓ 对 client 程序，填服务器 IP
- ✓ 对 server 程序，一般填 INADDR\_ANY，表示接收所有 IP 连接

网络安全与网络工程系系东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 12

**bind 函数**

**作用：**用于 Server 程序，绑定被侦听的端口

**原型：**`int bind(int sockfd, struct sockaddr* servaddr, int addrlen);`

**参数：**

- ✓ `sockfd`: 由 socket 调用返回的文件描述符
- ✓ `servaddr`: 出于兼容性，一般使用 `sockaddr_in` 结构
- ✓ `addrlen`: `servaddr` 结构的长度

**返回值：**

- ✓ 成功: 0
- ✓ 失败: -1, 相应地设定全局变量 `errno`, 最常见的错误是该端口已经被其他程序绑定

**注意：**在 Linux 系统中, 1024 以下的端口只有拥有 root 权限的程序才能绑定

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日 4时58分 13

**connect 函数**

**作用：**用于 Client 程序，连接到某个 Server

**原型：**`int connect(int sockfd, struct sockaddr* servaddr, int addrlen);`

**参数：**

- ✓ `sockfd`: socket 返回的文件描述符
- ✓ `servaddr`: 被连接的服务器端地址和端口信息，出于兼容性，一般使用 `sockaddr_in` 结构
- ✓ `addrlen`: `servaddr` 的长度

**返回值：**

- ✓ 成功: 0
- ✓ 失败: -1, 相应地设定全局变量 `errno`

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日 4时58分 14

**listen 函数**

**作用：**用于 Server 程序，侦听 bind 绑定的套接字

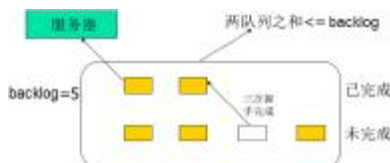
**原型：**`int listen(int sockfd, int backlog)`

**参数：**

- ✓ `sockfd`: 被 bind 的文件描述符 (socket() 建立的)
- ✓ `backlog`: 设置 Server 端请求队列的最大长度

**返回值：**

- ✓ 成功: 0
- ✓ 失败: -1



网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日 4时58分 15

**accept 函数**

**作用：**Server 用它响应连接请求，建立与 Client 连接

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

**参数：**

- ✓ `sockfd`: listen 后的文件描述符 (socket() 建立的)
- ✓ `addr`: 返回 Client 的 IP、端口等信息，确切格式由套接字的地址类别 (如 TCP 或 UDP) 决定；若 `addr` 为 NULL, 则 `addrlen` 应置为 NULL
- ✓ `addrlen`: 返回真实的 `addr` 所指向结构的大小，只要传递指针就可以，但必须先初始化为 `addr` 所指向结构的大小

**注意：**

- F `addr` 通常是指向局部数据结构 `sockaddr_in` 的指针
- F `addrlen` 是局部变量，通常设置为 `sizeof(struct sockaddr_in)`

**返回值：**

- ✓ 成功: Server 用于与 Client 进行数据传输的文件描述符
- ✓ 失败: -1, 相应地设定全局变量 `errno`

**说明：**accept 是阻塞函数，服务器端会一直阻塞到有一个客户程序发出了连接

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日 4时58分 16

**accept() 函数应用示例**

```
struct sockaddr_in their_addr; /* 用于存储连接对方的地址信息 */
int sin_size = sizeof(struct sockaddr_in);

... ..// (依次调用 socket(), bind(), listen() 等函数)
new_fd = accept(sockfd, &their_addr, &sin_size);
printf("对方地址: %s\n", inet_ntoa(their_addr.sin_addr));
... ..
```

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日 4时58分 17

**recv 函数**

**作用：**用于 TCP 协议中接收信息

**原型：**`ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);`

**参数：**

- ✓ `sockfd`: 接收端套接字描述符
- ✓ `buf`: 指向容纳接收信息的缓冲区的指针
- ✓ `nbytes`: `buf` 缓冲区的大小
- ✓ `flags`: 接收标志，一般置为 0 或：

flags	说明
MSG_DONTWAIT	仅本操作非阻塞
MSG_OOB	发送或接收带外数据
MSG_PEEK	窥看外来消息
MSG_WAITALL	等待所有数据

**返回值：**

- ✓ 成功: 实际接收的字节数
- ✓ 失败: -1, 相应地设定全局变量 `errno`
- ✓ 为 0: 表示对端已经关闭

**说明：**recv 缺省是阻塞函数，直到接收到信息或出错

网络安全与网络工程系杨东平 jsxhbc@163.com Linux操作系统 2020年3月2日 4时58分 18

**recvfrom 函数**

Ø作用：用于 UDP 协议中接收信息

Ø原型：ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags, struct sockaddr \*from, socklen\_t \*fromlen);

Ø参数：

- ✓ sockfd: socket 描述符
- ✓ buf: 指向容纳接收 UDP 数据报的缓冲区的指针
- ✓ len: buf 缓冲区的大小
- ✓ flags: 接收标志，一般为 0
- ✓ from: 指明数据的来源
- ✓ fromlen: 传入函数之前初始化为 from 的大小，返回之后存放 from 实际大小

Ø返回值：

- ✓ 成功：实际接收的字节数
- ✓ 失败：-1，错误原因存于 errno 中
- ✓ 为 0：表示对端已经关闭

Ø说明：recvfrom 是阻塞函数，直到接收到信息或出错

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux操作系统 2020年3月2日 4时58分 19

**send 函数**

Ø作用：用于 TCP 协议中发送信息

Ø原型：ssize\_t send(int sockfd, const void \*buf, size\_t nbytes, int flags);

Ø参数：

- ✓ sockfd: 指定发送端套接字描述符
- ✓ buf: 存放要发送数据的缓冲区
- ✓ nbytes: 实际要发送的数据的字节数
- ✓ flags: 一般设置为 0 或：

flags	说明
MSG_DONTROUTE	绕过路由表查找
MSG_DONTWAIT	仅本操作非阻塞
MSG_OOB	发送或接收带外数据

Ø返回值：

- ✓ 成功：已发送的字节数
- ✓ 失败：-1，相应地设定全局变量 errno

Ø说明：send 缺省是阻塞函数，直到发送完毕或出错

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux操作系统 2020年3月2日 4时58分 20

**sendto 函数**

Ø作用：用于 UDP 协议中发送信息

Ø原型：int sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

Ø参数：

- ✓ sock: 将要向其发送数据的套接字
- ✓ buf: 指向将要发送数据的缓冲区
- ✓ len: 数据缓冲区的长度
- ✓ flags: 一般是 0
- ✓ to: 指明数据的目的地址
- ✓ tolen: to 内存区的长度

Ø返回值：

- ✓ 成功：实际传送出去的字符数
- ✓ 失败：-1，错误原因存于 errno 中

Ø说明：sendto 缺省是阻塞函数，直到发送完毕或出错

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux操作系统 2020年3月2日 4时58分 21

**close 函数**

Ø作用：用于 TCP，关闭特定的 socket 连接

Ø原型：int close(int sockfd);

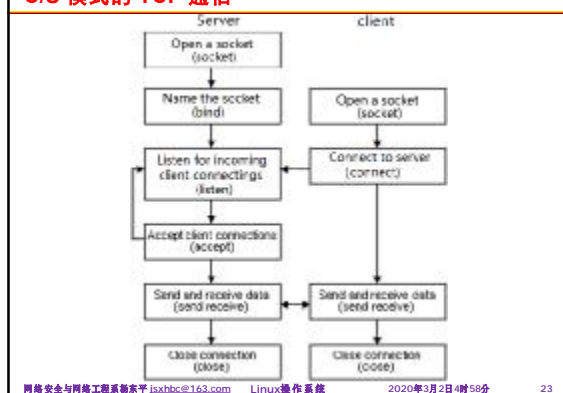
Ø返回值：

- ✓ 成功：0
- ✓ 失败：-1，并置错误码 errno：
  - F EBADF: sockfd 描述符无效
  - F EINTR: close 函数被信号中断
  - F EIO: IO 错误

Ø说明：

- ✓ close 一个 TCP socket 的缺省行为是把该 socket 标记为已关闭，然后立即返回到调用进程。该描述字不能再由调用进程使用
- ✓ close 操作只是使相应 socket 描述字的引用计数 -1，只有当引用计数为 0 的时候，才会触发 TCP 客户端向服务器发送终止连接请求

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux操作系统 2020年3月2日 4时58分 22

**C/S 模式的 TCP 通信****TCP 应用举例：一个简单的 TCP Server**

Ø基本步骤：

- ✓ 程序初始化
- ✓ 持续监听一个固定的端口
- ✓ 收到 Client 的连接后建立一个 socket 连接
- ✓ 与 Client 进行通信和信息处理
- ✓ 接收 Client 通过 socket 连接发送来的数据，进行相应处理并返回处理结果，如 BBS Server
- ✓ 通信结束后中断与 Client 的连接

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux操作系统 2020年3月2日 4时58分 24

### 一个简单的 TCP Server(TCPServer.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define MYPORT 3490 // 侦听的端口号
#define BACKLOG 10 // 侦听队列长度

void main()
{
    int serverfd, clientfd; // 侦听的套接字和客户套接字描述符
    struct sockaddr_in serveraddr; // 侦听的套接字地址信息
    struct sockaddr_in clientaddr; // 客户端的地址信息
    int clientsize; // 客户端地址信息结构大小
```

网络安全与网络工程系系办 jxxhbc@163.com Linux操作系统 2020年3月2日 4时58分 25

```
// 1 取得侦听套接字 socket 描述符
if((serverfd=socket(AF_INET, SOCK_STREAM,0)) == -1)
{
    perror("Server socket failed");
    exit(1);
}

// 2 填写侦听套接字地址信息的 sockaddr_in 结构
serveraddr.sin_family = AF_INET; // 使用 Internet 地址
serveraddr.sin_port = htons(MYPORT); // 网络字节序的侦听端口号
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY); // 网络字节序侦听 IP
bzero(&(serveraddr.sin_zero),8); // 其余部分置 0

// 3 绑定端口
if(bind(serverfd, (struct sockaddr*)&serveraddr,
        sizeof(struct sockaddr))==-1)
{
    perror("Server bind failed");
    exit(1);
}
```

```
// 4 监听端口
if(listen(serverfd, BACKLOG) == -1)
{
    perror("Server listen failed");
    exit(1);
}
// 主循环
while(1)
{
    // 5 接受客户端的连接请求
    clientsize = sizeof(struct sockaddr_in);
    clientfd = accept(serverfd, (struct sockaddr *)&clientaddr,
                     &clientsize);
    if(clientfd == -1)
    {
        perror("Server accept failed");
        continue;
    }
    int cliptip = clientaddr.sin_addr.s_addr; // 客户端的 IP 地址
    printf("Got connection from %d.%d.%d.%d\n",
           cliptip&255,(cliptip>>8)&255,(cliptip>>16)&255,(cliptip>>24)&255);
```

```
// 6 产生新进程(线程)处理读写 socket
if(!fork())
{
    // 子进程
    char msg[14]="Hello,world!\n";
    if(send(clientfd,msg,14,0)==-1) // 使用客户端 socket 描述符发送信息
        perror("send failed");

    // 7 读写完成, 关闭插口
    close(clientfd);
    exit(0);
}

close(clientfd); // 无法生成子进程时有用

while(waitpid(-1,NULL,WNOHANG)>0); // 清除所有子进程
}
```

### TCP应用举例：一个简单的TCP Client

#### Client 步骤

- ✓ 程序初始化
- ✓ 连接到某个 Server 上, 建立 socket 连接
- ✓ 与 Server 进行通信和信息处理
- ✓ 接收 Server 通过 socket 连接发送来的数据, 进行相应处理
- ✓ 通过 socket 连接向 Server 发送请求信息
- ✓ 通信结束后中断与 Server 的连接

网络安全与网络工程系系办 jxxhbc@163.com Linux操作系统 2020年3月2日 4时58分 29

### 一个简单的 TCP Client(TCPClient.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3490 // 服务器端的侦听端口
#define MAXDATASIZE 100 // 一次可以读的最大字节数

int main(int argc, char *argv[])
{
    int sockfd, numbytes; // 客户端 socket 描述符和地址信息结构大小
    char buf[MAXDATASIZE]; // 读取的数据缓冲区
    struct sockaddr_in their_addr; // Server 址信息
```

网络安全与网络工程系系办 jxxhbc@163.com Linux操作系统 2020年3月2日 4时58分 30

```
// (1) 取得客户端 socket 描述符
if ((sockfd=socket(AF_INET,SOCK_STREAM,0)) == -1)
{
    perror("socket");
    exit(1);
}

// (2) 填写被连接的服务端侦听的地址信息的 sockaddr_in 结构
their_addr.sin_family = AF_INET; // 使用 IP 地址
their_addr.sin_port = htons(PORT); // short, NBO, 服务端侦听端口号
their_addr.sin_addr.s_addr = inet_addr("192.168.116.169"); // 服务器 IP 地址
bzero(&(their_addr.sin_zero),8); // 其余部分设成 0

// (3) 向服务端发起连接
if (connect(sockfd, (struct sockaddr *)&their_addr,
            sizeof(struct sockaddr)) == -1)
{
    perror("connect");
    exit(1);
}
```

```
// (4) 读 socket
if ((numbytes=recv(sockfd,buf,MAXDATASIZE,0))!=-1)
{
    perror("recv");
    exit(1);
}
buf[numbytes] = '\0';
printf("Received: %s",buf);

// (5) 关闭 socket
close(sockfd);

return 0;
}
```

```
root@localhost ~# ./TCPServer
Got connection from 192.168.116.169
Got connection from 192.168.116.169
Got connection from 192.168.116.169
Got connection from 192.168.116.169
Got connection from 192.168.116.169
Got connection from 192.168.116.169
root@localhost ~# ./TCPClient
Received: Hello,world!
root@localhost ~# ./TCPClient
Received: Hello,world!
root@localhost ~# ./TCPClient
Received: Hello,world!
root@localhost ~# ./TCPClient
Received: Hello,world!
root@localhost ~# ./TCPClient
Received: Hello,world!
root@localhost ~# ./TCPClient
Received: Hello,world!
root@localhost ~#
```

### UDP 与 TCP 的区别

- Ø 基于连接与无连接
- Ø 流模式与数据报模式
  - ✓ TCP 保证数据正确性, UDP 可能丢包
  - ✓ TCP 保证数据顺序, UDP 不保证
- Ø 对系统资源的要求(TCP 较多, UDP 少)

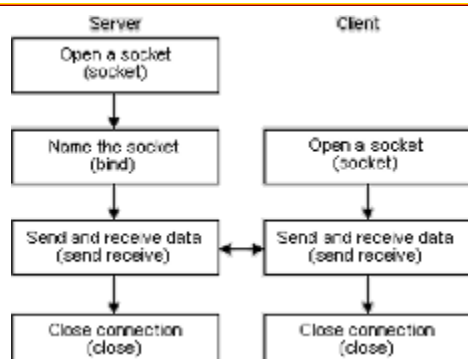
网络安全与网络工程系蔡东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 33

### UDP 与 TCP 的编程区别

- Ø socket() 的参数不同
- Ø UDP Server 不需要调用 listen 和 accept
- Ø UDP 收发数据用 sendto / recvfrom 函数
- Ø TCP: 地址信息在 connect/accept 时确定
  - ✓ UDP: 在 sendto / recvfrom 函数中每次均需指定地址信息

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 34

### UDP C/S 通信



网络安全与网络工程系蔡东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 35

### UDP 应用举例: Server 部分(UDPServer.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define MYPORT 3490 // 监听端口

void main()
{
    int serverfd; // 侦听的 socket 描述符
    struct sockaddr_in serveraddr; // 客户端地址信息
    struct sockaddr_in clientaddr; // 服务器端地址信息
    int addrlen; // 地址结构的大小
    int retval; // 接收的数据字节数
    char buf[128]; // 数据缓冲区
}
```

网络安全与网络工程系蔡东平 jsxbhc@163.com Linux 操作系统 2020年3月2日 4时58分 36

```
// 1 获取服务器的侦听 socket 描述符
if((serverfd=socket(AF_INET,SOCK_DGRAM,0))==-1)
{
    perror("socket");
    exit(1);
}

// 2 填写服务器端侦听的 IP 地址和端口号
serveraddr.sin_family = AF_INET; // 使用 IP 地址
serveraddr.sin_port = htons(MYPORT); // 网络字节序的本机端口号
serveraddr.sin_addr.s_addr=INADDR_ANY; // 自动填写本机 IP
bzero(&(serveraddr.sin_zero),8); // 其余部分置 0

// 3 绑定 socket 描述符和侦听 IP 地址及端口
if(bind(serverfd,(struct sockaddr *)&serveraddr,
        sizeof(serveraddr)) == -1)
{
    perror("bind");
    exit(1);
}
```

```
// 主循环
while(1)
{
    // 4 接收从客户端发送过来的数据
    retval = recvfrom(serverfd, buf, 128, 0,
        (struct sockaddr *)&clientaddr, &addrsz);
    int clientip = clientaddr.sin_addr.s_addr;
    printf("Received datagram from %d.%d.%d.%d\n",
        clientip&255,(clientip>>8)&255,(clientip>>16)&255,(clientip>>24)&255);

    if(retval==0) {
        perror("recvfrom");
        close(serverfd);
        break;
    }

    // 将接收的数据发回给客户端
    retval = sendto(serverfd, buf, 128, 0,
        (struct sockaddr *)&clientaddr, addrsz);
}
```

### UDP 应用举例：Client 部分

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3490 // UDP 服务器端的侦听端口号
#define MAXDATASIZE 100 // 一次可读取的最大字节数

int main(int argc, char* argv[])
{
    int clientfd; // UDP 客户端的 socket 描述符
    int numbytes; // 数据字节数
    int addrsz; // 地址结构的大小
    char buf[MAXDATASIZE] = "Hello,World!";
    struct sockaddr_in serveraddr; // UDP 服务器的地址信息
```

网络安全与网络工程系徐东平 jsxbbc@163.com Linux 操作系统 2020年3月2日 4时58分 39

```
// 1 获取 UDP 客户端的 socket 描述符
if((clientfd=socket(AF_INET,SOCK_DGRAM,0))==-1)
{
    perror("socket");
    exit(1);
}

// 2 填写 UDP 服务器端的地址信息
serveraddr.sin_family = AF_INET; // 使用 IP 地址
serveraddr.sin_port = htons(PORT); // 网络字节序的服务器侦听端口号
serveraddr.sin_addr.s_addr = inet_addr("192.168.116.169"); // 服务器端侦听 IP 地址
bzero(&(serveraddr.sin_zero),8); // 其余部分设置成 0
```

```
// 3 向 UDP 服务器发送数据
if((numbytes = sendto(clientfd,buf,MAXDATASIZE,0,
    (struct sockaddr *)&serveraddr,sizeof(serveraddr)))==1) {
    perror("sendto");
    exit(1);
}
printf("Send:%s\n",buf);

// 4 从 UDP 服务器接收数据
if((numbytes = recvfrom(clientfd,buf,MAXDATASIZE,0,
    (struct sockaddr *)&serveraddr,&addrsz))==-1)
{
    perror("recvfrom");
    exit(1);
}

buf[numbytes]='\0';
printf("Received:%s\n",buf);
close(clientfd);
return 0;
}
```

```
[root@localhost ~]# ./UDPServer
Received datagram from 255.127.0.0
Received datagram from 192.168.116.169
[root@localhost ~]#
```

```
[root@localhost ~]# ./UDPClient
Send:Hello,World!
Received:Hello,World!
[root@localhost ~]#
```

### UDP 编程的适用范围

Ø 部分满足以下几点要求时，应该用 UDP

- ✓ 面向数据报
- ✓ 网络数据大多为短消息
- ✓ 拥有大量 Client
- ✓ 对数据安全性无特殊要求
- ✓ 网络负担非常重，但对响应速度要求高

Ø 例子：ICQ、ping

网络安全与网络工程系徐东平 jsxbbc@163.com Linux 操作系统 2020年3月2日 4时58分 42