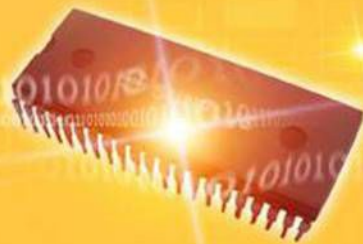


嵌入式系统工程师



网络编程—UDP



- 编程准备-字节序、地址转换
- UDP介绍、编程流程
- UDP编程-创建套接字
- UDP编程-发送、绑定、接收数据
- UDP编程-client、server



- 编程准备-字节序、地址转换
- UDP介绍、编程流程
- UDP编程-创建套接字
- UDP编程-发送、绑定、接收数据
- UDP编程-client、server



➤ 概念

是指多字节数据的存储顺序

➤ 分类

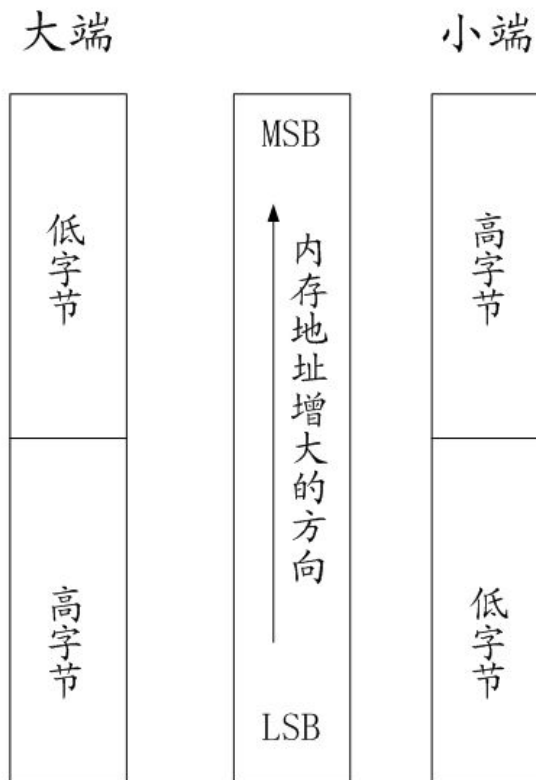
- 小端格式: 将低位字节数据存储在低地址
- 大端格式: 将高位字节数据存储在低地址

➤ 注意

- LSB: 低地址
- MSB: 高地址

➤ 想一想

- 怎样确定主机的字节序?



思考: 0x0102, 大端如何存储, 小端又如何存储

► 确定主机字节序程序

```
1  #include <stdio.h>
2  int main(int argc, char *argv[])
3  {
4      union{
5          short s;
6          char c[sizeof(short)];
7      }un;
8
9      un.s = 0x0102;
10     if( (un.c[0] == 1) && (un.c[1] == 2))
11     {
12         printf("big-endian\n");
13     }
14     else if( (un.c[0] == 2) && (un.c[1] == 1))
15     {
16         printf("little-endian\n");
17     }
18
19     return 0;
20 }
```

➤ 特点

- 网络协议指定了通讯字节序一大端
- 只有在多字节数据处理时才需要考虑字节序
- 运行在同一台计算机上的进程相互通信时,一般不用考虑字节序
- 异构计算机之间通讯,需要转换自己的字节序为网络字节序

➤ 在需要字节序转换的时候一般调用特定字节序转换函数

➤ `uint32_t htonl(uint32_t hostint32);`

- 功能:

将32位主机字节序数据转换成网络字节序数据

- 参数:

`hostint32`: 待转换的32位主机字节序数据

- 返回值:

成功: 返回网络字节序的值

- 头文件: `#include <arpa/inet.h>`

- `uint16_t htons (uint16_t hostint16);`
- 功能:
 - 将16位主机字节序数据转换成网络字节序数据
- 参数:
 - `uint16_t`: unsigned short int
 - `hostint16`: 待转换的16位主机字节序数据
- 返回值:
 - 成功: 返回网络字节序的值
- 头文件: `#include <arpa/inet.h>`

- `uint32_t ntohs(uint32_t netint32);`
- 功能:
 - 将32位网络字节序数据转换成主机字节序数据
- 参数:
 - `uint32_t`: unsigned int
 - `netint32`: 待转换的32位网络字节序数据
- 返回值:
 - 成功: 返回主机字节序的值
- 头文件: `#include <arpa/inet.h>`

➤ `uint16_t ntohs (uint16_t netint16);`

- 功能:

将16位网络字节序数据转换成主机字节序数据

- 参数:

`uint16_t`: unsigned short int

`netint16`: 待转换的16位网络字节序数据

- 返回值:

成功: 返回主机字节序的值

- 头文件: `#include <arpa/inet.h>`

➤ 示例

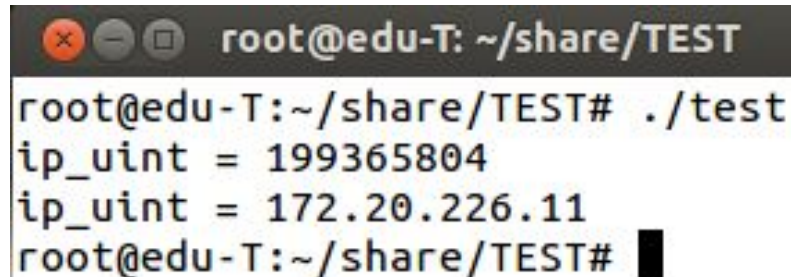
```
1  #include <stdio.h>
2  #include <arpa/inet.h>
3
4  int main(int argc, char *argv[])
5  {
6      int a = 0x01020304;
7      short int b = 0x0102;
8      printf("htonl(0x%08x) = 0x%08x\n", a, htonl(a));
9      printf("htons(0x%04x) = 0x%04x\n", b, htons(b));
10     return 0;
11 }
```

➤ 结果

```
root@edu-T:~/share/lh/work/net/hton# ./main
htonl(0x01020304) = 0x04030201
htons(0x0102) = 0x0201
```

- `int inet_pton(int family,`
 `const char *strptr, void *addrptr);`
- 功能：将点分十进制数串转换成32位无符号整数
 - 参数：
 - family 协议族
 - strptr 点分十进制数串
 - addrptr 32位无符号整数的地址
 - 返回值：成功返回1 、 失败返回其它
 - 头文件：#include <arpa/inet.h>

```
1  #include <stdio.h>
2  #include <arpa/inet.h>
3  int main(int argc, char *argv[])
4  {
5      char ip_str[] = "172.20.226.11";
6      unsigned int ip_uint = 0;
7      unsigned char *ip_p = NULL;
8      // 用char可以吗?
9      inet_pton(AF_INET, ip_str, &ip_uint);
10     printf("ip_uint = %d\n", ip_uint);
11
12     ip_p = (unsigned char *)&ip_uint;
13     printf("ip_uint = %d.%d.%d.%d\n", *ip_p, *(ip_p+1), *(ip_p+2), *(ip_p+3));
14     return 0;
15 }
```



```
root@edu-T: ~/share/TEST
root@edu-T:~/share/TEST# ./test
ip_uint = 199365804
ip_uint = 172.20.226.11
root@edu-T:~/share/TEST#
```

➤ `const char *inet_ntop(int family,
const void *addrptr,
char *strptr, size_t len);`

- 功能:

将32位无符号整数转换成点分十进制数串

- 参数:

family

协议族

addrptr

32位无符号整数

strptr

点分十进制数串

len

strptr缓存区长度

- len的宏定义

```
#define INET_ADDRSTRLEN    16    //for ipv4
```

```
#define INET6_ADDRSTRLEN  46    //for ipv6
```

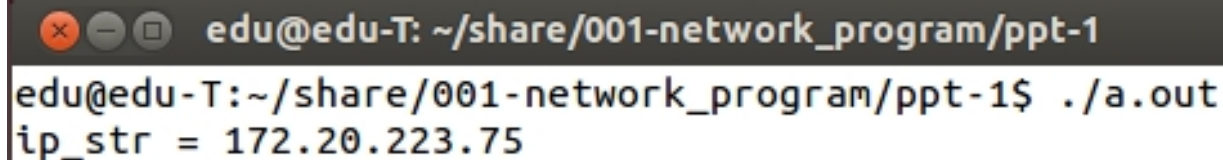
- 返回值:

成功: 则返回字符串的首地址

失败: 返回NULL

- 头文件: `#include <arpa/inet.h>`


```
1  #include <stdio.h>
2  #include <arpa/inet.h>
3  int main()
4  {
5      unsigned char ip[] = {172,20,223,75};
6      char ip_str[16] = "NULL";
7
8      inet_ntop(AF_INET, (unsigned int *)ip, ip_str, 16);
9      printf("ip_str = %s\n", ip_str);
10
11     return 0;
12 }
```



```
edu@edu-T: ~/share/001-network_program/ppt-1
edu@edu-T:~/share/001-network_program/ppt-1$ ./a.out
ip_str = 172.20.223.75
```

- 编程准备-字节序、地址转换
- UDP介绍、编程流程
- UDP编程-创建套接字
- UDP编程-发送、绑定、接收数据
- UDP编程-client、server



➤ UDP协议

面向无连接的用户数据报协议，在传输数据前不需要先建立连接；目的地主机的运输层收到UDP报文后，不需要给出任何确认

➤ UDP特点

- 相比TCP速度稍快些
- 简单的请求/应答应用程序可以使用UDP
- 对于海量数据传输不应该使用UDP
- 广播和多播应用必须使用UDP

➤ UDP应用

DNS (域名解析)、NFS (网络文件系统)、RTP (流媒体) 等

- 网络通信要解决的是不同主机进程间的通信
 - 首要问题是网络间进程标识问题
 - 以及多重协议的识别问题
- 20世纪80年代初，加州大学Berkeley分校在BSD(一个UNIX OS版本)系统内实现了TCP/IP协议；其网络程序编程开发接口为socket
- 随着UNIX以及类UNIX操作系统的广泛应用，socket成为最流行的网络程序开发接口

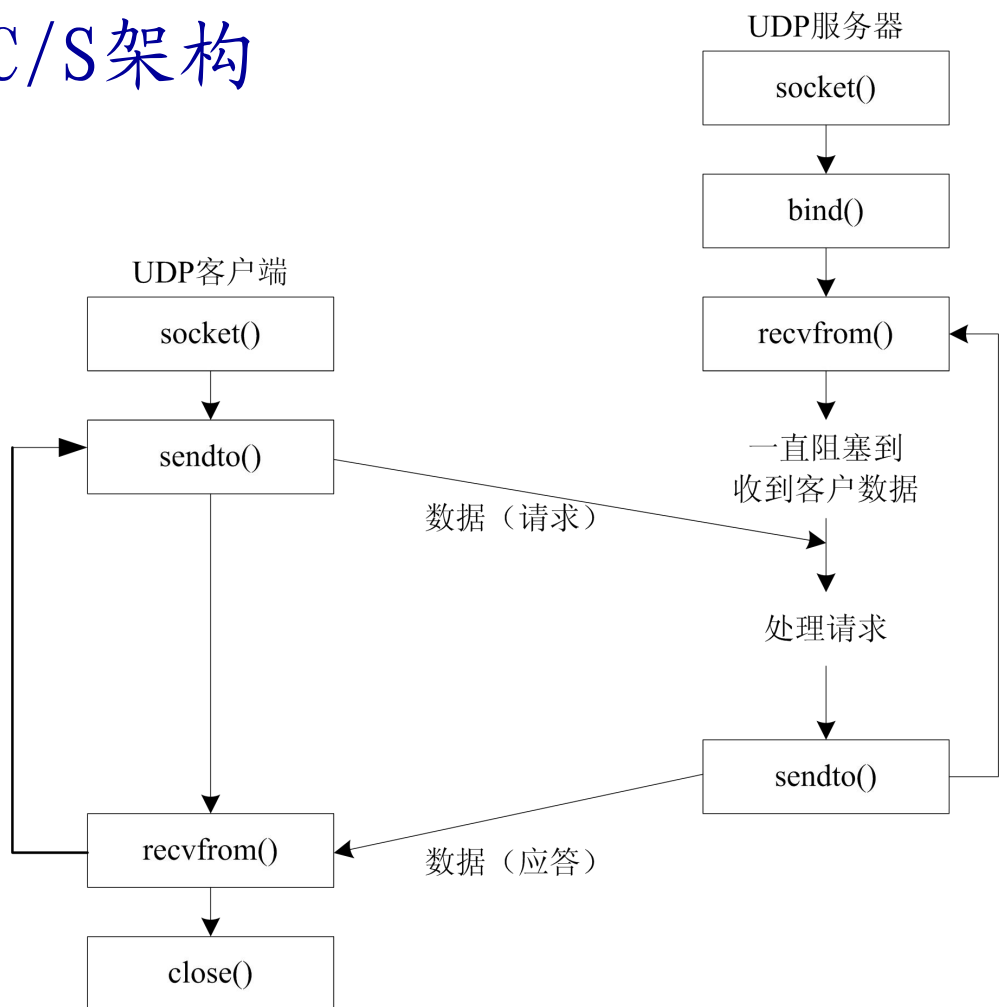
➤ socket作用

- 提供不同主机上的进程之间的通信

➤ socket特点

- socket也称“套接字”
- 是一种文件描述符,代表了一个通信管道的一个端点
- 类似对文件的操作一样,可以使用read、write、close等函数对socket套接字进行网络数据的收取和发送等操作
- 得到socket套接字（描述符）的方法调用socket()

➤ UDP编程C/S架构



- 编程准备-字节序、地址转换
- UDP介绍、编程流程
- UDP编程-创建套接字
- UDP编程-发送、绑定、接收数据
- UDP编程-client、server



➤ `int socket(int family, int type, int protocol);`

- 功能

创建一个用于网络通信的socket套接字（描述符）

- 参数

family: 协议族 (AF_INET、AF_INET6、PF_PACKET等)

type: 套接字类 (SOCK_STREAM、SOCK_DGRAM、SOCK_RAW等)

protocol: 协议类别 (0、IPPROTO_TCP、IPPROTO_UDP等)

- 返回值：套接字
- 特点
 - 创建套接字时，系统不会分配端口
 - 创建的套接字默认属性是主动的，即主动发起服务的请求；当作为服务器时，往往需要修改为被动的
- 头文件：`#include <sys/socket.h>`

➤ 创建UDP套接字demo

```
1  int sockfd = 0;  
2  sockfd = socket(AF_INET, SOCK_DGRAM, 0);  
3  if(sockfd < 0)  
4  {  
5      perror("socket");  
6      exit(-1);  
7  }
```

➤ 注意

- AF_INET: IPv4协议
- SOCK_DGRAM: 数据报套接字
- 0: 选择所给定的family和type组合的系统默认值

- 编程准备-字节序、地址转换
- UDP介绍、编程流程
- UDP编程-创建套接字
- UDP编程-发送、绑定、接收数据
- UDP编程-client、server



➤ IPv4套接字地址结构

```
1 struct in_addr
2 {
3     in_addr_t    s_addr; //4字节
4 };
5 struct sockaddr_in
6 {
7     sa_family_t   sin_family; //2字节
8     in_port_t     sin_port;    //2字节
9     struct in_addr sin_addr;    //4字节
10    char           sin_zero[8]; //8字节
11 };
```

➤ 头文件: #include <netinet/in.h>

➤ 通用套接字地址结构

- 为了使不同格式地址能被传入套接字函数, 地址须要强制转换成通用套接字地址结构
- 头文件: #include <netinet/in.h>

```
1 struct sockaddr
2 {
3     sa_family_t sa_family;    //2字节
4     char        sa_data[14]; //14字节
5 };
```

➤ 注意

- 以上3个结构在linux系统中已经定义

➤ 两种地址结构使用场合

- 在定义源地址和目的地址结构的时候，选用struct sockaddr_in;

例: `struct sockaddr_in my_addr;`

- 当调用编程接口函数，且该函数需要传入地址结构时需要用struct sockaddr进行强制转换

例: `bind(sockfd, (struct sockaddr*)&my_addr, sizeof(my_addr));`

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

➤ 功能

- 向to结构体指针中指定的ip, 发送UDP数据

➤ 参数:

- `sockfd`: 套接字
- `buf`: 发送数据缓冲区
- `nbytes`: 发送数据缓冲区的大小

➤ 参数

- flags: 一般为0
- to: 指向目的主机地址结构体的指针
- addrlen: to所指向内容的长度

➤ 注意

- 通过to和addrlen确定目的地址
- 可以发送0长度的UDP数据包

➤ 返回值

- 成功: 发送数据的字符数
- 失败: -1

➤ 向“网络调试助手”发送消息

```
edu@edu-T: ~/share/001-network_program/ppt-1  
edu@edu-T:~/share/001-network_program/ppt-1$ ./udp-send  
send data to UDP server 172.20.226.1:8080!  
hello sunplusedu
```

发送的数据

ubuntu



接收到的数据

windows

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8  int main(int argc, char *argv[])
9  {
10     unsigned short port = 8080;
11     char *server_ip = "172.20.226.1";
```

```
13     if( argc > 1 )                //服务器ip地址
14     {
15         server_ip = argv[1];
16     }
```

指定server信息

```
18     if( argc > 2 )                //服务器端口
19     {
20         port = atoi(argv[2]);
21     }
```

```
23     int sockfd;
24     sockfd = socket(AF_INET, SOCK_DGRAM, 0);    //创建UDP套接字
25     if(sockfd < 0)
26     {
27         perror("socket");
28         exit(-1);
29     }
```

1.创建UDP套接字

```
30
31 struct sockaddr_in dest_addr;
32 bzero(&dest_addr, sizeof(dest_addr));
33 dest_addr.sin_family = AF_INET;
34 dest_addr.sin_port = htons(port);
35 inet_pton(AF_INET, server_ip, &dest_addr.sin_addr);
```

2.填充目的server的信息

```
36
37 printf("send data to UDP server %s:%d!\n", server_ip, port);
38
```

```
39 while(1)
40 {
41     char send_buf[512] = "";
42     fgets(send_buf, sizeof(send_buf), stdin); //获取输入
43     send_buf[strlen(send_buf)-1] = '\0';
44     sendto(sockfd, send_buf, strlen(send_buf), 0, (struct sockaddr*)&dest_addr, sizeof(dest_addr));
45 }
46
```

3.发送数据到server

```
47 close(sockfd);
48 return 0;
49 }
```

- UDP网络程序想要收取数据需要什么条件?
 - 确定的ip地址
 - 确定的port
- 怎样完成上面的条件呢?
 - 接收端 使用bind函数，来完成地址结构与socket套接字的绑定，这样ip、port就固定了
 - 发送端 在sendto函数中指定接收端的ip、port，就可以发送数据了

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

➤ 功能：将本地协议地址与sockfd绑定

➤ 参数

- sockfd: socket套接字
- myaddr: 指向特定协议的地址结构指针
- addrlen: 该地址结构的长度

➤ 返回值

- 成功：返回0
- 失败：其他

➤ bind示例

```
int err_log = 0;
unsigned short port = 8000;
struct sockaddr_in my_addr;

bzero(&my_addr, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(port);
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);

err_log = bind(sockfd, (struct sockaddr*)&my_addr, sizeof(my_addr));
if(err_log != 0)
{
    perror("bind");
    close(sockfd);
    exit(-1);
}
```

➤ 注意: INADDR_ANY 通配地址, 值为0

➤ 头文件: #include <sys/socket.h>

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

➤ 功能

- 接收UDP数据，并将源地址信息保存在from指向的结构中

➤ 参数

- sockfd: 套接字
- buf: 接收数据缓冲区
- nbytes: 接收数据缓冲区的大小

- flags: 套接字标志 (常为0)
- from: 源地址结构体指针, 用来保存数据的来源
- addrlen: from所指内容的长度

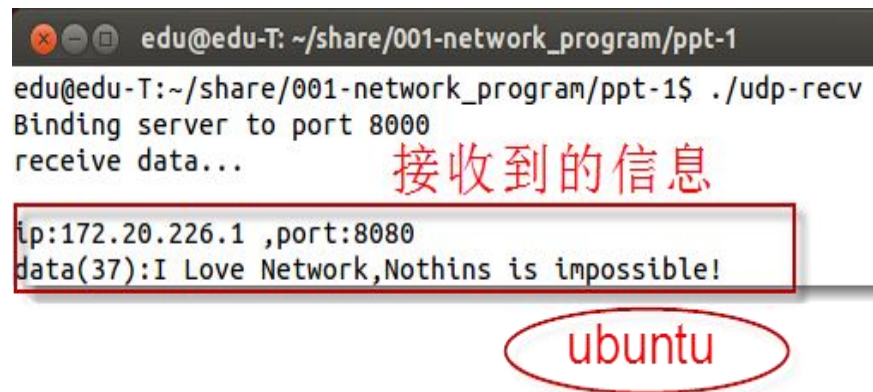
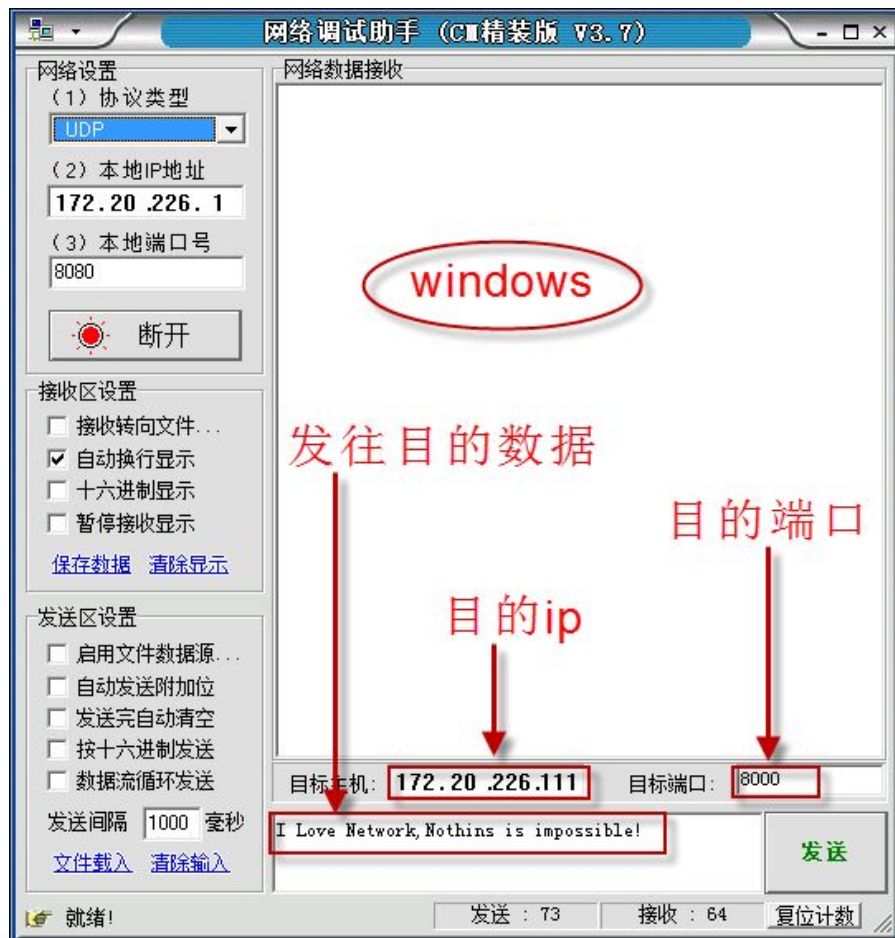
➤ 注意:

- 通过from和addrlen参数存放数据来源信息
- from和addrlen可以为NULL, 表示不保存数据来源

➤ 返回值:

- 成功: 接收到的字符数
- 失败: -1

► 接收“网络调试助手”的数据



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <sys/socket.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8  int main(int argc, char *argv[])
```

```
9  {
10     unsigned short port = 8000;
```

```
11     if(argc > 1)
12     {
13         port = atoi(argv[1]);
14     }
```

修改本程序的端口

```
15
16     int sockfd;
17     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
18     if(sockfd < 0)
19     {
20         perror("socket");
21         exit(-1);
22     }
```

1.创建套接字

```
23
24     struct sockaddr_in my_addr;
25     bzero(&my_addr, sizeof(my_addr));
26     my_addr.sin_family = AF_INET;
27     my_addr.sin_port = htons(port);
28     my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

2.填充本程序信息

```
30 printf("Binding server to port %d\n", port);
31 int err_log;
32 err_log = bind(sockfd, (struct sockaddr*)&my_addr, sizeof(my_addr));
33 if(err_log != 0)
34 {
35     perror("bind");
36     close(sockfd);
37     exit(-1);
38 }
```

3.绑定本程序要使用的信息

```
39 printf("receive data...\n");
40 while(1)
41 {
42     int recv_len;
43     char recv_buf[512] = "";
44     struct sockaddr_in client_addr;
45     char cli_ip[INET_ADDRSTRLEN] = ""; //INET_ADDRSTRLEN=16
46     socklen_t cliaddr_len = sizeof(client_addr);
47
48     recv_len = recvfrom(sockfd, recv_buf, sizeof(recv_buf), 0, (struct sockaddr*)&client_addr, &cliaddr_len);
49     inet_ntop(AF_INET, &client_addr.sin_addr, cli_ip, INET_ADDRSTRLEN);
50     printf("\nip:%s ,port:%d\n", cli_ip, ntohs(client_addr.sin_port));
51     printf("data(%d):%s\n", recv_len, recv_buf);
52 }
53 close(sockfd);
54 return 0;
55 }
```

4.收取数据

- 编程准备-字节序、地址转换
- UDP编程整体流程
- UDP编程-创建套接字
- UDP编程-发送、接收数据
- UDP编程-client、server

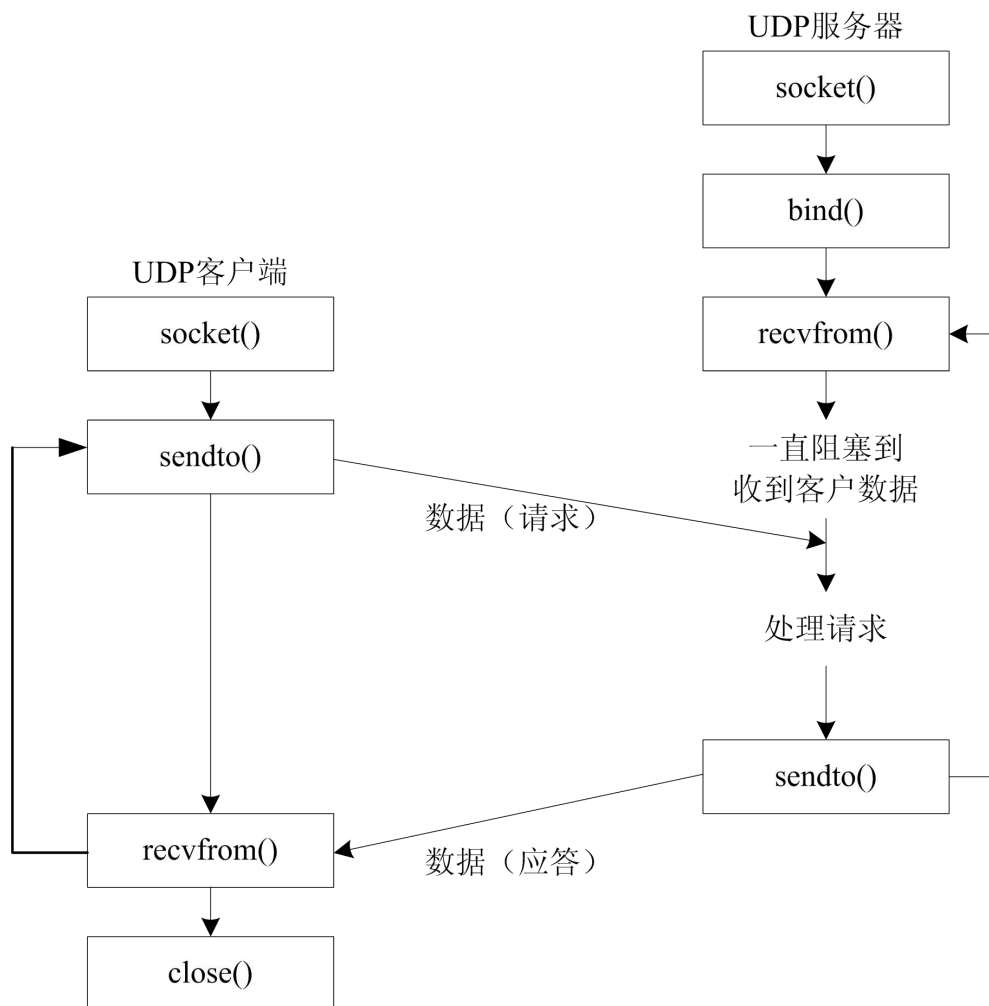


➤ 想一想

- 上节中的2个demo中发送数据端其实就是client; 接收数据端就是server, 那么client能否接收数据? server能否发送数据呢?

➤ 答

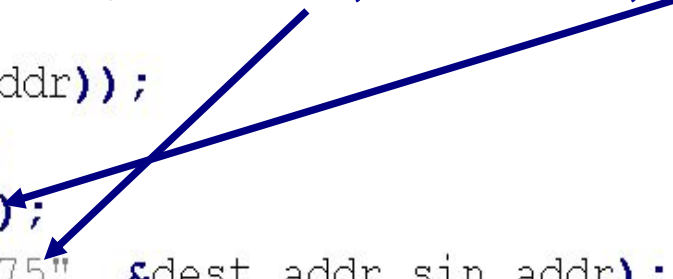
- 其实在网络编程开发中client和server双方既可以有发送数据还可以接收数据; 一般认为提供服务的一方为server; 而接受服务的另一方为client



➤ UDP客户端注意点

- 本地IP、本地端口（我是谁）
- 目的IP、目的端口（发给谁）
- 在客户端的代码中，我们只设置了目的IP、目的端口

```
bzero(&dest_addr, sizeof(dest_addr));  
dest_addr.sin_family = AF_INET;  
dest_addr.sin_port = htons(8080);  
inet_pton(AF_INET, "172.20.223.75", &dest_addr.sin_addr);
```



- 客户端的本地ip、本地port是我们调用sendto的时候linux系统底层自动给客户端分配的；分配端口的方式为随机分配，即每次运行系统给的port不一样

➤ UDP服务器注意点

- 服务器之所以要bind是因为它的本地port需要是固定，而不是随机的
- 服务器也可以主动地给客户端发送数据

➤ 客户端也可以用bind，这样客户端的本地端口就是固定的了，但一般不这样做



值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

