在前面的示例演示代码、socket常用函数中，服务端需要将创建的socket进行bind到本地的struct sockadd\_in结构对象上，在服务端accept和客户端connet时，以及sendto/recvfrom中需要传递struct sockaddr对象指针作为参数。

我们在使用过程中需要将地址的文本表达式转换为存放在套接字地址结构中的 二进制数据，需要调用地址转换函数，而这些函数都是与协议相关的。

因此，本节主要说明常用的套接字结构地址，以及地址结构的成员变量，以及相关的函数，另外还介绍用于获取或返回与某个套接字关联的本地/外地的协议地址。

**1、套接字地址结构**

对于不同的协议族，都定义了不同的套接字地址结构，都是以sockaddr\_开头，对应每个协议族的唯一后缀尾。

* 1. **IPv4套接字地址结构**

typedef uint32\_t in\_addr\_t;

struct in\_addr {

in\_addr\_t s\_addr; /\* IPv4 address，**network byteordered** \*/

}

struct sockaddr\_in {

sa\_family\_t sin\_family; /\* **uint16\_t** AF\_INET \*/

in\_port\_t sin\_port; /\* **uint16\_t** Port number. **network byteordered** \*/

struct in\_addr sin\_addr; /\* **uint32\_t** Internet address. \*/

/\* Pad to size of `struct sockaddr'. \*/

unsigned char sin\_zero[sizeof (struct sockaddr) -

sizeof (sa\_family\_t) -

sizeof (in\_port\_t) -

sizeof (struct in\_addr)]; // 实际是**8**个字节

}; // 一共 16字节

* 1. **IPv6套接字地址结构**

struct sockaddr\_in6

{

sa\_family\_t sin6\_family; /\* **uint16\_t** **AF\_INET6** \*/

in\_port\_t sin6\_port; /\* **uint16\_t** Transport layer port # \*/

uint32\_t sin6\_flowinfo; /\* **uint32\_t** IPv6 flow information \*/

struct in6\_addr sin6\_addr; /\* **128bits** IPv6 address \*/

uint32\_t sin6\_scope\_id; /\* **uint32\_t** IPv6 scope-id \*/

}; // 一共 28 字节

* 1. **通用套接字地址结构**

struct sockaddr {

sa\_family\_t sin\_family; /\* **uint16\_t** AF\_INET \*/

char sa\_data[14]; /\* **14 Bytes** Address data. \*/

} // 16字节

Socket的函数地址参数基本都是sockaddr结构，根据实际传入的第二个参数地址结构长度来决定解析哪一种套接字地址结构。

* 1. **新的通用套接字地址结构**

#define \_\_ss\_aligntype unsigned long int

#define \_SS\_PADSIZE (\_SS\_SIZE - \_\_SOCKADDR\_COMMON\_SIZE - sizeof (\_\_ss\_aligntype))

struct sockaddr\_storage

{

sa\_family\_t ss\_family; /\* **2 Bytes** Address family, etc. \*/

char \_\_ss\_padding[\_SS\_PADSIZE]; /\* **118 Bytes** \*/

\_\_ss\_aligntype \_\_ss\_align; /\* **8 Bytes** Force desired alignment. \*/

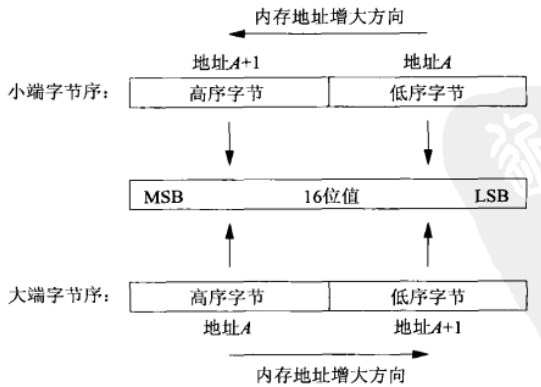
}; // 128 字节

sockaddr\_storage类型满足对齐、空间足够大，内容透明，需要强制转换成适用于ss\_family字段对应的地址类型套接字地址结构。

**2、套接字地址相关函数**

**2.1 网络字节序**

网络协议补习指定网络字节序，即大端字节序。大端字节序下，数据的高位数据保存在内存区的高地址上。



主机字节序和网络字节序的转换函数有

#include <netinet/in.h>

uint32\_t ntohl (uint32\_t \_\_netlong);

uint16\_t ntohs (uint16\_t \_\_netshort); // 返回主机字节序

uint32\_t htonl (uint32\_t \_\_hostlong);

uint16\_t htons (uint16\_t \_\_hostshort); // 返回网络字节序

函数后缀虽然是l和s（long和short），但是long这里实际还是32位的值。在小端操作系统中会进行大小端转换，大端操作系统实际是一个空宏，不做转换。

**2.2 字节操作**

多字节的操作函数有两组，不对数据作解释，也不假设数据是以空字符结束的C字符串。以空字符结尾的C字符串是头文件<string.h>中以str（表示字符串）开头的函数处理的。

名字以b（表示字节）开头的函数起源于4.2 BSD，以mem（表示内存）开头的函数在支持ANSI C函数库的系统都支持。

#include <strings.h>

void bzero(void \*dest, size\_t nbytes);

void bcopy(void \*src, void \*dest, size\_t nbytes); // 重叠正常

int bcmp(void \*ptr1, void \*ptr2, size\_t nbytes);

#include <string.h>

void \*memset(void \*src, void \*dest, int c, size\_t len);

void \*memcopy(void \*dest, const void \*src, size\_t nbytes); // 重叠时使用memmove

int memcmp(const void \*ptr1, const void \*ptr2, size\_t nbytes);

**2.3 地址转换**

介绍在ASCII字符串格式与网络字节序的二进制数据之间转换网络地址。

1. **inet\_aton、inet\_addr和inet\_ntoa**

点分十进制数串(如“192.168.3.123”)与长度为32位的网络字节序IPv4地址转换。

#include <arpa/inet.h>

int net\_aton(const char \*strptr, struct in\_addr \*addrptr); // 转换成功返回1，否则返回0

in\_addr\_t inet\_addr(const char \*strptr); // 字符串有效返回32位IPv4地址，否则返回INADDR\_NONE

char \*inet\_ntoa(struct in\_addr addrptr); //返回点分十进制数串的字符指针

函数inet\_addr()失败返回0xffffffff，实际是”255.255.255.255”的广播地址，有些手册出错时返回-1，已经废弃，使用新版的inet\_aton函数；inet\_ntoa函数是不可重入（有其他多个函数体重公用一个全局变量，多线程会有问题），建议使用新版的inet\_ntop函数。

1. **inet\_pton、inet\_ntop**

这两个函数是随IPv6出现的，也支持IPv4地址转换，并且都是可重入的。函数名中p和n分别表示表达presentation和数值numeric。地址的表达格式是ASCii字符串，数值格式是存放在套接地址结构中的二进制值。

#include <arpa/inet.h>

int net\_pton(int family, const char \*strptr, void \*addrptr); // 转换成功返回1，无效字符串返回0，出错返回-1

const char \*inet\_ntop(int family, const void \*addrptr, char \*strptr, size\_t len); //返回结果指针，出错返回NULL

两个函数的参数family可以是AF\_INET、AF\_INET6。第二个函数的len参数是目标存储单元的大小，为避免超过缓冲区长度，定义了两个宏

#include <arpa/inet.h>

#define INET\_ADDRSTRLEN 16

#define INET6\_ADDRSTRLEN 46

在IPv4地址转换中，我们可以替换老代码

foo.sin\_addr.s\_addr = inet\_addr(cp); <== itet\_pton(AF\_INET, cp, &foo.sin\_addr);

ptr = inet\_ntoa(foo.sin\_addr); <== char str[INET\_ADDRLEN];

ptr=inet\_ntop(AF\_INET, &foo.sin\_addr, str, INET\_ADDRLEN);

**2.4 地址操作**

套接字的地址处理主要在两个地方，第一个是构建struct sockaddr\_in和struct sockaddr\_in6地址结构用于bind()或connect()或sendto()，第二个是用在recvfrom()以获取struct sockaddr\_in或struct sockaddr\_in6。

1. 构建地址结构

struct sockaddr\_in addr\_in4;

// 初始化内存(若ip和ort都赋值非零值，可去掉)

//memset(&addr\_in4,0,sizeof(addr\_in4));

// 协议地址族

addr\_in4.sin\_family = AF\_INET; // IPv4

// 地址赋值

const char ip4\_addr[] = "192.168.3.100";

addr\_in4.sin\_addr.s\_addr = INADDR\_ANY; //INADDR\_ANY=0, 内核选择

addr\_in4.sin\_addr.s\_addr = inet\_addr(ip4\_addr); //废弃，仅IPv4

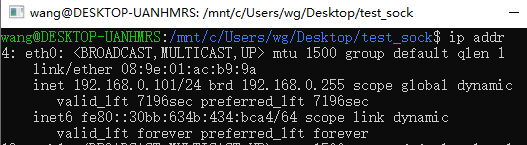
inet\_aton(ip4\_addr, &addr\_in4.sin\_addr); // 仅IPv4

inet\_pton(AF\_INET, ip4\_addr, &addr\_in4.sin\_addr); // 通用IPv4/6,建议

// 端口赋值

addr\_in4.sin\_port = 0; // 内核选择

ddr\_in4.sin\_port = htons(8080); //网络序



struct sockaddr\_in6 addr\_in6;

memset(&addr\_in6, 0, sizeof(addr\_in6));

// 协议地址族

addr\_in6.sin6\_family = AF\_INET6; // IPv6

// 地址赋值

const char ip6\_addr[] = "fe80::30bb:634b:434:bca4";

addr\_in6.sin6\_addr = IN6ADDR\_ANY\_INIT; // 内核选择

addr\_in6.sin6\_addr.s6\_addr16[0] = htons(std::stoi("fe80",0,16));

addr\_in6.sin6\_addr.s6\_addr16[1] = 0;

addr\_in6.sin6\_addr.s6\_addr16[2] = 0;

addr\_in6.sin6\_addr.s6\_addr16[3] = 0;//手动赋值

addr\_in6.sin6\_addr.s6\_addr16[4] = htons(std::stoi("30bb",0,16));

addr\_in6.sin6\_addr.s6\_addr16[5] = htons(std::stoi("634b",0,16));

addr\_in6.sin6\_addr.s6\_addr16[6] = htons(std::stoi("434",0,16));

addr\_in6.sin6\_addr.s6\_addr16[7] = htons(std::stoi("bca4",0,16));

inet\_pton(AF\_INET6, ip6\_addr, &addr\_in6.sin6\_addr); // 建议

// 端口赋值

addr\_in6.sin6\_port = 0; // 内核选择

addr\_in6.sin6\_port = htons(8080); //网络序

1. 获取地址结构

struct sockaddr\_storage storage;

socklen\_t socklen =sizeof(storage); // 区别是ipv4,还是ipv6

int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&storage, &socklen);

if (len < 0){

printf("%s: recv failed. err %s \n", \_\_func\_\_, strerror(errno));

}

else{

// 获取客户端的ip、和port

char ip[INET6\_ADDRSTRLEN];

int port;

if(socklen == sizeof(sockaddr\_in)) { //Ipv4

sockaddr\_in clientaddr = \*(sockaddr\_in\*)&storage;

inet\_ntop(AF\_INET, &clientaddr.sin\_addr, ip, socklen);

port = ntohs(clientaddr.sin\_port);

printf("%s: client [%s:%d] recv %2d: %s\n", \_\_func\_\_, ip, port, len, buf);

}

else if(socklen == sizeof(sockaddr\_in6)) { //Ipv6

sockaddr\_in6 clientaddr = \*(sockaddr\_in6\*)&storage;

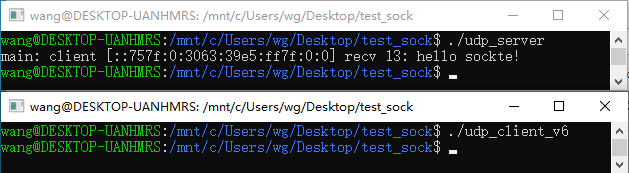
inet\_ntop(AF\_INET6, &clientaddr.sin6\_addr, ip, socklen);

port = ntohs(clientaddr.sin6\_port);

printf("%s: client [%s:%d] recv %2d: %s\n", \_\_func\_\_, ip, port, len, buf);

}

}



**3、套接字关联的本地/外地地址**

在网络通信中，TCP服务端可能需要知道当前建立连接的客户端地址，或者UDP/TCP客户端未调用bind函数绑定本地地址、仅调用bind地址仅赋值ip或port的一者，需要获取内核分配的地址信息，就需要通用到如下两个函数

#include <sys/socket.h>

int getpeername (int sockfd, struct sockaddr \*localaddr, socklen\_t \* addrlen);

int getsockname (int sockfd, struct sockaddr \*peeraddr, socklen\_t \* addrlen); // 成功返回0，出错返回-1

这两个函数的使用方式，参看后面的TCP套接字编程。

网络编程（8）TCP套接字编程（2）并发服务器。。。。链接