1.如果已经建立的udp socket，如果改变了PC的IP，则socket不受影响，前提是用localaddr.sin\_addr.s\_addr = INADDR\_ANY;

但是PC改变IP后，此socket加入的组播需要重新加入：所以重新建立socket，重新加入组播

 2. bind端口之前，设置socket选项REUSEADDR，支持一个端口绑定多个socket。原因是一个端口close之后，并没有立刻释放干净，会进入TIME\_WAIT阶段，通常完全释放需要2分钟

3.对于 tcp 而言，任何一方 IP 更改，连接不定断开

网络字节序

字节序

大端字节序（Big Endian) 最高有效位（MSB：Most Significant Bit）存储于最低内存地址处，最低有效位（LSB：Lowest Significant Bit）存 储于最高内存地址处。

小端字节序（Little Endian） 最高有效位（MSB：Most Significant Bit）存储于最高内存地址处，最低有效位（LSB：Lowest Significant Bit）存 储于最低内存地址处。

主机字节序 不同的主机有不同的字节序，如x86为小端字节序，Motorola 6800为大端字节序，ARM字节序是可配置的。

网络字节序 网络字节序规定为大端字节序

为使网络程序具有可移植性，使同样的C代码在大端和小端计算机上编译后都能正常运行，可以调用以下库函数做网络字节序和主机字节序的转换。

#include <arpa/inet.h>

uint32\_t htonl(uint32\_t hostlong);

uint16\_t htons(uint16\_t hostshort);

uint32\_t ntohl(uint32\_t netlong);

uint16\_t ntohs(uint16\_t netshort);

这些函数名很好记，h表示host，n表示network，l表示32位长整数，s表示16位短整数。例如htonl表示将32位的长 整数从主机字节序转换为网络字节序，例如将IP地址转换后准备发送。如果主机是小端字节序，这些函数将参数做相 应的大小端转换然后返回，如果主机是大端字节序，这些函数不做转换，将参数原封不动地返回。

下面写个小程序测试下主机的大小端：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> File Name: byteorder.c

> Author: Simba

> Mail: dameng34@163.com

> Created Time: Fri 01 Mar 2013 04:16:08 PM CST

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<arpa/inet.h>

int main(void)

{

unsigned int x = 0x12345678;

unsigned char \*p = (unsigned char \*)&x;

printf("%x %x %x %x\n", p[0], p[1], p[2], p[3]);

unsigned int y = htonl(x);

p = (unsigned char \*)&y;

printf("%x %x %x %x\n", p[0], p[1], p[2], p[3]);

return 0;

}

输出为：

simba@ubuntu:~/Documents/code/linux\_programming/UNP/socket$ ./byteorder  78 56 34 12 12 34 56 78

即本主机是小端字节序，而经过htonl 转换后为网络字节序，即大端。

三、地址转换函数

前面提到的 sockaddr\_in 结构体中的成员struct in\_addr sin\_addr表示32位的IP地址。但是我们通常用点分十进制的字符串表示IP地址，以下函数可以在字符串表示和in\_addr表示之间转换。

字符串转in\_addr的函数：

#include <arpa/inet.h>

int inet\_aton(const char \*strptr, struct in\_addr \*addrptr);

in\_addr\_t inet\_addr(const char \*strptr);

int inet\_pton(int family, const char \*strptr, void \*addrptr);

注意，转换而成的32位数是网络字节序的。

in\_addr转字符串的函数：

char \*inet\_ntoa(struct in\_addr inaddr);

const char \*inet\_ntop(int family, const void \*addrptr, char \*strptr, size\_t len);

注意，传入的32位数也是网络字节序的。

其中inet\_pton和inet\_ntop不仅可以转换IPv4的in\_addr，还可以转换IPv6的in6\_addr，因此函数接口是void \*addrptr。

下面写个小程序演示一下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> File Name: addr\_in.c

> Author: Simba

> Mail: dameng34@163.com

> Created Time: Fri 01 Mar 2013 04:16:08 PM CST

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<arpa/inet.h>

int main(void)

{

unsigned int addr = inet\_addr("192.168.0.100"); //转换后是网络字节序（大端）

printf("add=%u\n", ntohl(addr));

struct in\_addr ipaddr;

ipaddr.s\_addr = addr;

printf("%s\n", inet\_ntoa(ipaddr));

return 0;

}

输出为：

simba@ubuntu:~/Documents/code/linux\_programming/UNP/socket$ ./addr\_in  add=3232235620 192.168.0.100

注意，在打印addr的时候先转换成主机字节序。

四、套接字类型

流式套接字(SOCK\_STREAM) 提供面向连接的、可靠的数据传输服务，数据无差错，无重复的发送，且按发送顺序接收。 数据报式套接字(SOCK\_DGRAM) 提供无连接服务。不提供无错保证，数据可能丢失或重复，并且接收顺序混乱。 原始套接字（SOCK\_RAW）