一、字节顺序和转换函数

1、大小端

不同机器内部对变量的字节存储顺序不同，有的采用大端模式，有的采用小端模式 **大端模式**：高字节数据存放在低字节处，低字节数据存放在高字节处 **小端模式**：低字节数据存放在低字节处，高字节数据存放在高字节处

2、网络字节顺序

在网络上传输数据时，由于数据传输的两端可能对应不同的硬件平台，采用的存储字节顺序也可能不一致，因此TCP/IP协议规定了在网络上必须采用网络字节顺序（也就是大端模式）。 数据在发送到网络之前将其转换成大端模式，在接受到数据之后再将其转换成符合接收端主机的存储模式。

3、大小端转换函数

```
#include<netinet.h>
uint32_t htonl(uint32_t, hostlong);
unit16_t htons(uint16_t, hostshort);
uint32_t ntohl(uint32_t, netlong);
uint16_t ntohs(uint16_t, netshort);12345
```

htonl：host to network long,用于将主机unsigned int 型数据转换成网络字节顺序 htons：host to network short,用于将主机unsigned short 型数据转换成网络字节顺序 ntohl、ntohs的功能分别与htonl、htons相反。

二、INET 系列函数

通常我们习惯使用字符串形式的网络地址，但在网络上进行数据通信时需使用二进制形式且为网络字节顺序的IP地址。Linux为网络地址的格式转换提供了一系列函数。

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
int inet_aton（const char *cp, struct in_addr *inp）;
in_addr_t inet_addr(const char *cp);
in_addr_t inet_network(const char *cp);
char* inet_ntoa(struct in_addr in);
struct in_addr inet_makeaddr(int net, int host);
in_addr_t inet_lnaof(struct in_addr in);
in_addr_t inet_netof(struct in_addr in);12345678910
```

- 函数**inet_aton** inet_aton（）将参数cp所指向的字符串形式的IP地址转换为二进制的网络字节顺序的IP地址，转换后的结果存于参数inp所指向的空间中 。

  > 执行成功返回非0值，参数无效则返回0。

- 函数**inet_addr** inet_addr()将参数cp所指向的字符串形式的网络地址转换为网络字节顺序形式的二进制的IP 地址。

  > 执行成功时将转换后的结果返回，参数无效返回INADDR_NONE（一般该值为-1） 该函数已过时。因为对有效地址“255.255.255.255”它也返回-1（因为-1的补码形式为0xFFFFFFFF），使得用户可能将“255.255.255.255”当成无效的非法地址，而inet_aton()不存在这种问题。

- 函数**inet_network** inet_network()将参数cp所指向的字符串形式的网络地址转换为主机字节顺序形式的二进制IP地址。

  > 执行成功返回转换后的结果，参数无效返回-1。

- 函数**inet_ntoa** inet_ntoa()将值为in的网络字节顺序形式的二进制IP地址转换成以“.”分隔的字符串形式。

  > 执行成功返回结果字符串的指针，参数无效返回NULL。

- 函数**inet_makeaddr** inet_makeaddr()将把网络号为参数net，主机号为参数host的两个地址组合成一个网络地址。
- 函数**inet_lnaof**
- inet_lnaof()从参数in中提取出主机地址。

  > 执行成功返回主机字节顺序形式的主机地址。

- 函数**inet_netof** inet_netof()从参数in中提取处网络地址。

  > 执行成功返回主机字节顺序形式的网络地址。

演示

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>

int main()
{
    char            buffer[32];
    int             ret = 0;
    int             host = 0;
    int             network = 0;
    unsigned int    address = 0;
    struct in_addr in;

    in.s_addr = 0;

    //输入一个以"."分隔的字符串形式的IP地址
    printf("please input your IP adsress:");
    fgets(buffer, 31, stdin);
    buffer[31] = '\0';

    //示例使用inet_aton()函数
    if (ret = inet_aton(buffer, &in) == 0) {
        printf("inet_aton:\t invalid address\n");
    } else {
        printf("inet_aton:\t0x%x\n", in.s_addr);
    }

    //示例使用inet_addr()函数
    if ((address = inet_addr(buffer)) == INADDR_NONE) {
        printf("inet_addr: \t invalid address\n");
    } else {
        printf("inet_addr:\t0x%x\n", address);
    }

    //示例使用inet_network()函数
    if ((address = inet_network(buffer)) == -1) {
        printf("inet_network:\t invalid address\n");
    } else {
```

三、套接字属性

1.getsockopt()&&setsockopt()

套接字创建以后，就可以利用它来传输数据，但有时可能对套接字的工作方式有特殊的要求，此时就需要修改套接字的属性。系统提供了套接字选项来控制套接字的属性，使用函数getsockopt可以获取套接字的属性，使用setsockopt可以设置套接字的属性。

```
#include<sys/types.h>
#include<sys/socket.h>
int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen);
int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);
```

> s：一个套接字。 level：进行套接字选项操作的层次，可以取SOL_SOCKET（通用套接字）、IPPROTO_IP（IP层套接字）、IPPROTO_TCP（TCP层套接字）等值。 optname：套接字选项的名称。 optval：对于getsockopt（），optval用来存放获得的套接字选项；对于setsockopt（），optval是待设置的套接字选项的值。 optlen：对于getsockopt（），在调用函数前optlen的值为optval指向的空间大小，调用完成后其值为optval所保存的结果的实际大小；对于setsockopt（），optlen是optval选项的长度。

两个函数执行成功时都返回0，出错都返回-1。

2.SOL_SOCKET的选项

- **SO_KEEPALIVE**
  如果没有设置SO_KEEPALIVE选项，那么即使TCP连接已经很长时间没有数据传输时，系统也不会检测这个连接是否有效。对于服务器进程，如果某一客户端非正常断开连接，则服务器进程将一直被阻塞等待。因此服务器端程序需要使用这个选项，如果某个客户端一段时间内**没有反应则关闭该连接**
- **SO_RCVLOWAT和SO_SNDLOWAT**
  SO_RCVLOWAT表示接受缓冲区的下限，只有当接受缓冲区中的数据超过了SO_RCVLOWAT才会将数据传送到上层应用程序。SO_SNDLOWAT表示发送缓冲区的下限，只有当发送缓冲区中数据超过了SO_SNDLOWAT才会将数据发送出去。Linux下这两个值都为1且不能更该，也就是说只要有数据就接受或发送。
- **SO_RCVTIMEO和SO_SNDTIMEO**
  这两个选项可以设置对套接字读或写的超时时间，具体时间由下面这个结构指定：

```
struct timeval {
   long tv_sec;        //秒数
   long tv_usec;       //微秒数
};
```

tv_sec指定秒数，tv_usec指定微秒数。超过时间为这两个时间的和。在某个套接字连接上，若读或写超时，则认为接收或发送数据失败。

- **SO_BINDTODEVICE** 将套接字绑定到特定的网络接口，此后只有该网络接口上的数据才会被套接字处理。如果将选项值设置为空字符串或选项长度设置为0将取消绑定。
- **SO_DEBUG** 该选项只能对TCP套接字使用，设置了该选项之后系统将保存TCP发送和接收的所有数据的相关信息，以方便调试程序。

- **SO_REUSEADDR**

  Linux系统中，如果socket绑定了一个端口，该socket正常关闭或程序异常退出后的一段时间内，该端口依然维持原来的绑定状态，其他程序无法绑定该端口，如果设置了该选项则可以避免这个问题。

- **SO_TYPE** 用于获取套接字的类型。如SOCK_DGRAM,SOCK_STREAM,SOCK_SEQPACKET,SOCK_RDM等。
- **SO_ACCEPTCONN** 该选项用来检测套接字是否处于监听状态，如果为0表示处于非监听状态，如果为1表示正在监听。
- **SO_DONTROUTE** 设置该选项表示在发送IP数据包时不使用路由表来寻找路由。
- **SO_BROADCAST** 该选项用来决定套接字是否能在网络上广播数据。实际应用中要在网络上广播数据必须硬件支持广播并且使用的是SOCK_SGRAM套接字。
- **SO_SNDBUF和SO_RCVBUF** 这两个选项用于设置套接字的发送和接收缓冲区大小。对于TCP类型的套接字，缓冲区太小会影响TCP的流量控制；对于UDP类型的套接字，如果套接字的数据缓冲区满则后续数据将被丢弃，实际应用中应根据需要设置一个合适的大小。
- **SO_ERROR** 获取套接字内部的错误变量so_error，当套接字上发生了异步错误时，系统将设置套接字的so_error。异步错误是指错误的发生和错误被发现的时间不一致，通常在目的主机非正常关闭时发生这种错误。

四、多路复用

1.多路复用的两种方法

在客户端/服务器模型中，服务器端需要同时处理多个客户端的连接请求，此时就需要使用多路复用。

- 实现多路复用最简单的方法是采用非阻塞方式套接字，服务器端不断的查询各个套接字的状态，如果有数据到达则读出数据，如果没有数据则查看下一个套接字。这种方法简单，但是在轮询过程中浪费了大量的CPU时间，效率非常低。
- 另一种方法是服务器进程并不主动的询问套接字状态，而是向系统登记希望监视的套接字，然后阻塞。当套接字上有事件发生时（如有数据到达），系统通知服务器进程告知哪个套接字上发生了什么事件，服务器进程查询对应套接字并进行处理。在这种工作方式下，套接字上没有事件发生时，服务器进程不会去查询套接字的状态，从而不会浪费CPU时间，提高了效率。

2.select函数

select函数可以实现第二种多路复用

```
#include<sys/select.h>
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval
*timeout);
```

> n：需要监视的文件描述符集，腰间使得文件描述符值为0～n-1。 reafds：指定需要监视的可读文件描述符集合，当这个集合中的一个描述符上有数据到达时，系统将通知调用select函数的程序。 writefds：指定需要监视的可写文件描述符集合，当这个集合中的某个描述符可以发数据时，程序将收到通知。 exceptfds：指定需要监视的异常文件描述符集合，当该集合中的一个描述符发生异常之后，程序将收到通知。 timeout：指定了阻塞时间，如果在这段时间内监视的文件描述符上都没有事件发生，则函数select（）将返回0。

selcet()设定的要监视的文件描述符集合中有描述符发生了事件，则select将返回发生时间的文件描述符的个数。

```
struct timeval {
    long tv_sec;     //秒数
    long tv_usec;    //微秒数
};
```

> 如果将timeout设为NULL，则函数select将一直被阻塞，直到某个文件描述符上发生了事件。如果将timeout设为0，则此时相当于非阻塞方式，函数select查询完文件描述符集合的状态后立即返回。如果将timeout设成某一时间值，在这个时间内如果没有事件发生，函数select将返回；如果在这段时间内有事件发生，程序将收到通知。

3.文件描述符集合的宏

```
FD_CLR(int fd, fd_set *set);          //将文件描述符fd从文件描述符集set中删除
FD_ISSET(int fd, fd_set *set);        //测试fd是否在set中
FD_SET(int fd, fd_set *set);          //在文件描述符集合set中增加文件描述符fd
FD_ZERO（fd_set *set）；               //将文件描述符集合set清空1234
```