# 基础知识

## 字节序

|  |  |
| --- | --- |
| 网络字节序 | is Most Significant Byte first (big endian) |
| i80x86 host 字节序 | host byte order is Least Significant Byte first (little endian) |

## 字节序转换函数

|  |  |  |  |
| --- | --- | --- | --- |
| Head files | Functions | Description | Comments |
| #include <arpa/inet.h> | uint32\_t htonl(uint32\_t hostlong); | convert values between host and network byte order  \*数值的转换 | host->net  long |
| uint16\_t htons(uint16\_t hostshort); | host->net  short |
| uint32\_t ntohl(uint32\_t netlong); | net->host  long |
| uint16\_t ntohs(uint16\_t netshort); | net->host  short |
| #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h> | int inet\_aton(const char \*cp, struct in\_addr \*inp); | Internet address manipulation routines  \*字符型地址和值之间的转换 | host->net  **number-and-dots** -> binary  &  store it in struct in\_addr  &  returns non-zero if the address is valid, zero if not |
| in\_addr\_t inet\_addr(const char \*cp); | host->net  obsoleted because -1 is a valid address (255.255.255.255)  **number-and-dots** -> binary  &  return the binary |
| in\_addr\_t inet\_network(const char \*cp); | net->host  number-and-dots ->binary |
| char \*inet\_ntoa(struct in\_addr in); | net->host  binary -> numbers-and-dots |
| struct in\_addr inet\_makeaddr(int net, int host); | host->net  combine the network number net with the local address host in network net |
| in\_addr\_t inet\_lnaof(struct in\_addr in); | returns the **local host address** part of the Internet address in. The local host address is returned in local host byte order |
| in\_addr\_t inet\_netof(struct in\_addr in); | returns the **network number** part of the Internet Address in. The network number is returned in local host byte order |

# Socket

Socket是进程通讯的一种方式，即调用这个网络库的一些API函数实现分布在不同主机的相关进程之间的数据交换。

几个定义：

（1）IP地址：即依照TCP/IP协议分配给本地主机的网络地址，两个进程要通讯，任一进程首先要知道通讯对方的位置，即对方的IP。

（2）端口号：用来辨别本地通讯进程，一个本地的进程在通讯时均会占用一个端口号，不同的进程端口号不同，因此在通讯前必须要分配一个没有被访问的端口号。

（3）连接：指两个进程间的通讯链路。

（4）半相关：网络中用一个三元组可以在全局唯一标志一个进程：

（协议，本地地址，本地端口号）

这样一个三元组，叫做一个半相关,它指定连接的每半部分。

（4）全相关：一个完整的网间进程通信需要由两个进程组成，并且只能使用同一种高层协议。也就是说，不可能通信的一端用TCP协议，而另一端用UDP协议。因此一个完整的网间通信需要一个五元组来标识：

（协议，本地地址，本地端口号，远地地址，远地端口号）

这样一个五元组，叫做一个相关（association），即两个协议相同的半相关才能组合成一个合适的相关，或完全指定组成一连接。

<http://acm.tzc.edu.cn/acmhome/projectList.do?method=projectNewsDetail&nid=2>

## Server and client

Server

Client

socket

bind

listen

socket

connect

send

accept

recv

recv

send

close

close

## Socket

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> | |
| 原型 | int socket(int domain, int type, int protocol); | |
| 参数 | domain | The domain parameter specifies a communication domain |
| type | specifies the communication semantics |
| protocol | The protocol specifies a particular protocol to be used with the socket |
| 返回值 | On success | a file descriptor for the new socket is returned. |
| On error | -1 is returned, and errno is set appropriately. |

### Domain

主要协议族

Name Purpose Man page

PF\_UNIX, PF\_LOCAL Local communication unix(7)

PF\_INET IPv4 Internet protocols ip(7)

PF\_INET6 IPv6 Internet protocols

PF\_IPX IPX - Novell protocols

PF\_NETLINK Kernel user interface device netlink(7)

PF\_X25 ITU-T X.25 / ISO-8208 protocol x25(7)

PF\_AX25 Amateur radio AX.25 protocol

PF\_ATMPVC Access to raw ATM PVCs

PF\_APPLETALK Appletalk ddp(7)

PF\_PACKET Low level packet interface packet(7)

AF\_UNIX=AF\_LOCAL,

PF\_UNIX=PF\_LOCAL,

AF\_LOCAL=PF\_LOCAL,

AF\_INET=PF\_INET.

通过查看/usr/include/bits/socket.h发现AF\_XXX和PF\_XXX是可以互相替代使用的。

Windows中AF\_INET和PF\_INET是完全一样的，而在Unix/Linux系统中，在不同的版本中这两者有微小差别.对于BSD是AF,对于POSIX是PF.

### Type

Linux2.6内核

/\* Types of sockets. \*/

enum \_\_socket\_type

{

SOCK\_STREAM = 1, /\* Sequenced, reliable, connection-based \*/ tcp网络层

SOCK\_DGRAM = 2, /\* Connectionless, unreliable datagrams \*/ udp网络层

SOCK\_RAW = 3, /\* Raw protocol interface. \*/ ip网络层

SOCK\_RDM = 4, /\* Reliably-delivered messages. \*/

SOCK\_SEQPACKET = 5, /\* Sequenced, reliable, connection-based, datagrams of fixed maximum length. \*/

SOCK\_DCCP = 6, /\* Datagram Congestion Control Protocol. \*/

SOCK\_PACKET = 10, /\* Linux specific way of getting packets at the dev level. For writing rarp and other similar things on the user level. \*/ datalink数据链路层

/\* Flags to be ORed into the type parameter of socket and socketpair and

used for the flags parameter of paccept. \*/

SOCK\_CLOEXEC = 02000000, /\* Atomically set close-on-exec flag for thenew descriptor(s). \*/

SOCK\_NONBLOCK = 04000 /\* Atomically mark descriptor(s) as non-blocking. \*/

};

SOCK\_STREAM

Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK\_DGRAM

Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK\_SEQPACKET

Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each read system call.

SOCK\_RAW

Provides raw network protocol access.

SOCK\_RDM

Provides a reliable datagram layer that does not guarantee ordering.

SOCK\_PACKET

Obsolete and should not be used in new programs; see packet(7).

### Protocol

**Protocol定义**

查看所有的protocols number在文件/etc/protocols

Normally only a single protocol exists to support a particular socket type within a given protocol family, in which case protocol can be specified as 0.

However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner.

The protocol number to use is specific to the "communication domain" in which communication is to take place;

**Protocol macros**

查看所有的Ethernet Protocol ID在头文件/usr/include/linux/if\_ether.h，其中定义了IPPROTO\_TCP， IPPROTO\_UDP等

**默认protocol**

内核使用IPPROTO\_IP仅仅是为了在某些情况下默认匹配协议类型而已。

例如

SOCK\_DGRAM协议族默认的协议是IPPROTO\_UDP，

SOCK\_STREAM默认的是IPPROTO\_TCP。

即   
socket(PF\_INET, SOCK\_DGRAM, IPPROTO\_IP) 和 socket(PF\_INET, SOCK\_DGRAM, IPPROTO\_UDP)是一样的，   
socket(PF\_INET, SOCK\_STREAM, IPPROTO\_IP) 和 socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)是一样的。

### RAW Socket

|  |  |  |  |
| --- | --- | --- | --- |
| socket | 能 | 不能 | comments |
| socket(  AF\_INET,  SOCK\_RAW,  IPPROTO\_TCP|IPPROTO\_UDP|IPPROTO\_ICMP) | 该套接字可以接收和发送协议类型为(tcp udp icmp等)发往本机的ip数据包,不包括mac地址. | 不能收到非发往本地ip的数据包(ip软过滤会丢弃这些不是发往本机ip的数据包). | 发送接收ip数据包 |
| 不能收到从本机发送出去的数据包. |
| 不能修改mac地址。数据包是网络层不等于数据帧，不包括mac地址，数据帧是链路层的，包括mac地址。要发送的话需要自己组织tcp udp icmp等头部.可以setsockopt来自己包装ip头部，这种套接字用来写个ping程序比较适合 |
| socket(  PF\_PACKET,  SOCK\_RAW,  htons(ETH\_P\_IP|ETH\_P\_ARP|ETH\_P\_ALL)) | 接收发往本地mac的数据帧 |  | 发送接收以太网数据帧 |
| 接收从本机发送出去的数据帧(第3个参数需要设置为ETH\_P\_ALL) |  |
| 接收非发往本地mac的数据帧(网卡需要设置为promisc混杂模式) |  |
| socket(  AF\_INET,  SOCK\_PACKET, htons(ETH\_P\_IP|ETH\_P\_ARP|ETH\_P\_ALL)) | 发送接收以太网数据帧（过时了,不要用啊） | | |
| （SOCK\_PACKET 方式使用结构 sockaddr\_pkt来保存数据链路层信息，但该结构缺乏包类型信息；其次，如果参数 MSG\_TRUNC 传递给读包函数recvmsg()、recv()、recvfrom()等，则函数返回的数据包长度是实际读到的包数据长度，而不是数据包真正的长度。Libpcap 的开发者在源代码中明确建议不使用 2.0，http://www.cnblogs.com/huyc/archive/2011/10/19/2217134.html） | | |

### 总结

1. 处理TCP、UDP、ICMP的数据包

socket(AF\_INET, SOCK\_RAW, IPPROTO\_TCP|IPPROTO\_UDP|IPPROTO\_ICMP)

UDP

socket(PF\_INET, SOCK\_DGRAM, IPPROTO\_IP) 和   
socket(PF\_INET, SOCK\_DGRAM, IPPROTO\_UDP)是一样的，   
socket(AF\_INET, SOCK\_DGRAM, 0);也是一样的

socket(AF\_INET, SOCK\_RAW, IPPROTO\_UDP);

TCP

socket(PF\_INET, SOCK\_STREAM,IPPROTO\_IP) 和   
socket(PF\_INET, SOCK\_DGRAM, IPPROTO\_TCP)一样。   
socket(PF\_PACKET, SOCK\_RAW, IPPROTP\_TCP);

1. 处理各种IP数据包

socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_IP|ETH\_P\_ARP|ETH\_P\_ALL))

1. 处理链路层上所有的数据包

socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL))

### FP\_PACKET

在LINUX内核版本中（2.0 releases),一个名为PF\_PACKET的协议簇被加了进来！这个簇允许应用程序直接利用网络驱动程序发送和接收报文，避免了原来的协议栈处理过程，在这种情况下，所有SOCKET发出的报文直接送到以太网卡接口，而接口收到的任何报文将直接送到应用程序  
  
The PF\_PACKET协议簇支持两个稍微有点不同的SOCKET类型，SOCK\_DGRAM和SOCK\_RAW。前者让内核处理添加或者去除以太网报文头部工作，而后者则让应用程序对以太网报文头部有完全的控制！在SOCKET调用中的协议类型必须符合/usr/include/linux/if\_ether.h中定义的以太网IDs中的一个，除非遇到特别声明的协议，一般你可以用ETH\_P\_IP来处理IP的一组协议（TCP,UDP,ICMP,raw IP等等）因为它们容易受到一些很严重的安全问题的牵连（比如你可以伪造一个MAC地址），所以只有具有root权限才可以使用PF\_PACKET-family socket.这也就是为什么只有具有root权限后才能运行嗅探器的原因！  
<http://www.cnblogs.com/jinrize/archive/2009/11/24/1609902.html>

参考链接

<http://blog.csdn.net/ttyttytty12/article/details/8141910>

## Bind

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> | |
| 原型 | int bind(int sockfd, const struct sockaddr \*my\_addr, socklen\_t addrlen); | |
| 参数 | sockfd | socket fd |
| my\_addr | see below |
| addrlen | address length |
| 返回值 | On success, zero is returned.  On error, -1 is returned, and errno is set appropriately. | |

**struct sockaddr**

The actual structure passed for the my\_addr argument will depend on the address family. The sockaddr structure is defined as something like:

struct sockaddr {

sa\_family\_t sa\_family;

char sa\_data[14];

}

It is defined in file /usr/include/bits/socket.h

|  |  |
| --- | --- |
| struct sockaddr | 通用的接口 |
| struct sockaddr\_in | 用于AF\_INET域，之后强转成struct sockaddr\* |
| struct sockaddr\_un | 用于AF\_UNIX域，之后强转成struct sockaddr\* |

**sockaddr\_in**

sockaddr\_in（在netinet/in.h中定义）：  
struct  sockaddr\_in {  
short  int  sin\_family;                      /\* Address family \*/  
unsigned  short  int  sin\_port;       /\* Port number \*/  
struct  in\_addr  sin\_addr;              /\* Internet address \*/  
unsigned  char  sin\_zero[8];         /\* Same size as struct sockaddr \*/  
};  
struct  in\_addr {  
unsigned  long  s\_addr;  
};

typedef struct in\_addr {  
union {  
            struct{  
                        unsigned char s\_b1,  
                        s\_b2,  
                        s\_b3,  
                        s\_b4;  
                        } S\_un\_b;  
           struct {  
                        unsigned short s\_w1,  
                        s\_w2;  
                        } S\_un\_w;  
            unsigned long S\_addr;  
          } S\_un;  
} IN\_ADDR;

**sockaddr\_un**

sockaddr\_un (sys/un.h)

#define UNIX\_PATH\_MAX 108

struct sockaddr\_un {

sa\_family\_t sun\_family; /\*PF\_UNIX或AF\_UNIX \*/

char sun\_path[UNIX\_PATH\_MAX]; /\* 路径名 \*/

};

## Listen

To accept connections, a socket is first created with socket(2), a willingness to accept incoming connections and a queue limit for incoming connections are specified with listen(), and then the connections are accepted with accept(2).

The listen() call applies only to sockets of type SOCK\_STREAM or SOCK\_SEQPACKET.

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/socket.h> | |
| 原型 | int listen(int sockfd, int backlog); | |
| 参数 | sockfd | socket fd |
| backlog | defines the maximum length the queue of pending connections may grow to |
| 返回值 | On success, zero is returned.  On error, -1 is returned, and errno is set appropriately. | |

## Connect

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> | |
| 原型 | int connect(int sockfd, const struct sockaddr \*serv\_addr, socklen\_t addrlen); | |
| 参数 | sockfd | socket fd |
| serv\_addr | peer’s addr, see below |
| addrlen | address length |
| 返回值 | If the connection or binding succeeds, zero is returned.  On error, -1 is returned, and errno is set appropriately. | |

注：

If the socket has not already been bound to a local address, connect() shall bind it to an address which, unless the socket's address family is AF\_UNIX, is an unused local address.

Connect的动作会根据connection-mode的不同而有所差异：

|  |  |  |
| --- | --- | --- |
| not connection-mode | SOCK\_DGRAM sockets  serv\_addr指示数据包发送的目标地址，并限制了后续recv()接受数据包的来源 | |
| connection-mode | O\_NONBLOCK is set | connect()直接返回错误，并置errno=EINPROGRESS，但是连接请求不被中断，连接会被异步地建立。  如果connect()被信号中断，则置errno=EINTR，连接请求不被中断，连接会被异步地建立。  后续的select() and poll()能够得知连接是否成功建立。 |
| O\_NONBLOCK is not set | connect()会被block直到超时，如果超时了也未能建立连接，connect()会返回错误，连接请求会终止。 |

## Accept

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> | |
| 原型 | int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen); | |
| 参数 | sockfd | sockt fd |
| addr | This structure is filled in with the address of the peer socket. |
| addrlen | this is a value-result argument: it should initially contain the size of the structure pointed to by addr;  on return it will contain the actual length (in bytes) of the address returned. When addr is NULL nothing is filled in. |
| 返回值 | On success, accept() returns a non-negative integer that is a descriptor for the accepted socket.  On error, -1 is returned, and errno is set appropriately. | |

## Send

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> | |
| 原型 | ssize\_t send(int s, const void \*buf, size\_t len, int flags);  ssize\_t sendto(int s, const void \*buf, size\_t len, int flags, const struct sockaddr \*to, socklen\_t tolen);  ssize\_t sendmsg(int s, const struct msghdr \*msg, int flags); | |
| 参数 | s | socket fd |
| buf | buffer pointer |
| len | buffer size |
| flags | see below |
| to | pointer to sockaddr structure containing the destination address |
| tolen | length of “to” |
| msg | pointer to struct msghdr, see below |
| 返回值 | the number of characters sent | On success |
| -1 | On error and errno is set appropriately |

## Receive

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <sys/types.h>  #include <sys/socket.h> | |
| 原型 | ssize\_t recv(int s, void \*buf, size\_t len, int flags);  ssize\_t recvfrom(int s, void \*buf, size\_t len, int flags,  struct sockaddr \*from, socklen\_t \*fromlen);  ssize\_t recvmsg(int s, struct msghdr \*msg, int flags); | |
| 参数 | s | socket fd |
| buf | buffer for the data to be received |
| len | Specifies the length in bytes of the buffer pointed to by the buf argument |
| flags | see below |
| from | If from is not NULL, and the underlying protocol provides the **source address**, this source address is filled in. |
| fromlen | The argument fromlen is a value-result parameter, initialized to the size of the buffer associated with from, and **modified on return** to indicate the actual size of the address stored there. |
| msg | see below |
| 返回值 | These calls return the number of bytes received, or -1 if an error occurred.  The return value will be 0 when the peer has performed an orderly shutdown. | |

The recvmsg() call uses a msghdr structure to minimize the number of directly supplied parameters. This structure has the following form, as defined in <sys/socket.h>:

struct msghdr {

void \*msg\_name; /\* optional address \*/

socklen\_t msg\_namelen; /\* size of address \*/

struct iovec \*msg\_iov; /\* scatter/gather array \*/

size\_t msg\_iovlen; /\* # elements in msg\_iov \*/

void \*msg\_control; /\* ancillary data, see below \*/

socklen\_t msg\_controllen; /\* ancillary data buffer len \*/

int msg\_flags; /\* flags on received message \*/

};

### Flags

The flags parameter is the bitwise OR of zero or more of the following flags.

MSG\_CONFIRM (Linux 2.3+ only)

Tell the link layer that forward progress happened: you got a successful reply from the other side. If the link layer doesn't get this it will regularly reprobe the neighbour (e.g. via a unicast ARP). Only valid on SOCK\_DGRAM

and SOCK\_RAW sockets and currently only implemented for IPv4 and IPv6. See arp(7) for details.

MSG\_DONTROUTE

Don't use a gateway to send out the packet, only send to hosts on directly connected networks. This is usually used only by diagnostic or routing programs. This is only defined for protocol families that route; packet sockets

don't.

MSG\_DONTWAIT

Enables non-blocking operation; if the operation would block, EAGAIN is returned (this can also be enabled using the O\_NONBLOCK with the F\_SETFL fcntl(2)).

MSG\_EOR

Terminates a record (when this notion is supported, as for sockets of type SOCK\_SEQPACKET).

MSG\_MORE (Since Linux 2.4.4)

The caller has more data to send. This flag is used with TCP sockets to obtain the same effect as the TCP\_CORK socket option (see tcp(7)), with the difference that this flag can be set on a per-call basis.

Since Linux 2.6, this flag is also supported for UDP sockets, and informs the kernel to package all of the data sent in calls with this flag set into a single datagram which is only transmitted when a call is performed that does

not specify this flag. (See also the UDP\_CORK socket option described in udp(7).)

MSG\_NOSIGNAL

Requests not to send SIGPIPE on errors on stream oriented sockets when the other end breaks the connection. The EPIPE error is still returned.

MSG\_OOB

Sends out-of-band data on sockets that support this notion (e.g. of type SOCK\_STREAM); the underlying protocol must also support out-of-band data.

The definition of the msghdr structure follows. See recv(2) and below for an exact description of its fields.

struct msghdr {

void \*msg\_name; /\* optional address \*/

socklen\_t msg\_namelen; /\* size of address \*/

struct iovec \*msg\_iov; /\* scatter/gather array \*/

size\_t msg\_iovlen; /\* # elements in msg\_iov \*/

void \*msg\_control; /\* ancillary data, see below \*/

struct iovec \*msg\_iov; /\* scatter/gather array \*/

size\_t msg\_iovlen; /\* # elements in msg\_iov \*/

void \*msg\_control; /\* ancillary data, see below \*/

socklen\_t msg\_controllen; /\* ancillary data buffer len \*/

int msg\_flags; /\* flags on received message \*/

};

You may send control information using the msg\_control and msg\_controllen members. The maximum control buffer length the kernel can process is limited per socket by the net.core.optmem\_max sysctl; see socket(7).

### ERRORS

These are some standard errors generated by the socket layer. Additional errors may be generated and returned from the underlying protocol modules; see their respective manual pages.

EACCES (For Unix domain sockets, which are identified by pathname) Write permission is denied on the destination socket file, or search permission is denied for one of the directories the path prefix. (See path\_resolution(2).)

EAGAIN or EWOULDBLOCK

The socket is marked non-blocking and the requested operation would block.

EBADF An invalid descriptor was specified.

ECONNRESET

Connection reset by peer.

EDESTADDRREQ

The socket is not connection-mode, and no peer address is set.

EFAULT An invalid user space address was specified for a parameter.

EINTR A signal occurred before any data was transmitted.

EINVAL Invalid argument passed.

EISCONN

The connection-mode socket was connected already but a recipient was specified. (Now either this error is returned, or the recipient specification is ignored.)

EMSGSIZE

The socket type requires that message be sent atomically, and the size of the message to be sent made this impossible.

ENOBUFS

The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion. (Normally, this does not occur in Linux. Packets are just silently

dropped when a device queue overflows.)

ENOMEM No memory available.

ENOTCONN

The socket is not connected, and no target has been given.

ENOTSOCK

The argument s is not a socket.

EOPNOTSUPP

Some bit in the flags argument is inappropriate for the socket type.

EPIPE The local end has been shut down on a connection oriented socket. In this case the process will also receive a SIGPIPE unless MSG\_NOSIGNAL is set.

## Close

|  |  |  |
| --- | --- | --- |
| 头文件 | #include <unistd.h> | |
| 原型 | int close(int fd); | |
| 参数 | fd | socket fd |
| 返回值 | zero | on success |
| -1 | on error, and errno is set appropriately |

## 代码示例

<http://blog.csdn.net/wind19/article/details/6156339>

## 一览表

|  |  |  |
| --- | --- | --- |
| API | Arguments | Comments |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |