套接字：

抽象出来的通信端点接口

套接字被UNIX系统认为是一个文件描述符，传统的文件描述符引用的本主机磁盘上的文件，socket套接字文件描述符引用的任一网络中可以寻到的进程。

可以指定网络上任意一个进程读取和发送数据

从逻辑上来说，我们对自身的套接字并不关心，我们只需要关系想要读取或者发送的对端套接字

建立：

1.int socket(int domain, int type, int protocal)

domain——确定通信的地址模式：AF_INET, AF_INET6, AF_UNIX(AF_LOCAL), AF_UNSPEC

type——确定套接字类型，即想要的服务特点：SOCK_DGRAM, SOCK_RAM, SOCK_SEQPACKET,SOCK_STREAM

protocol——在domain和type的约束下，选择的具体的通信协议，默认为0，即选择对应的默认协议

succ， 返回文件描述符，err，返回-1

关闭套接字：

1. int close(int fd)

释放文件描述符，但是只有在指向文件表项的文件描述符指针数目为1的时候，在close后才会彻底的关闭该套接字，否则，其他指向该文件表的文件描述符依旧可以来进行读写操作。

2. int shutdown(int fd, int how)

不管引用计数，直接关闭这个文件的读写特性，所有指向该文件表的文件描述符都会被设置成相同的状态

how: SHUT_RD, SHUT_WR, SHUT_RDWR

suc, re 0， else re -1

字节序：

数据的低位存储在低地址就是小端字节序，存储在高地址就是大端字节序

字节序转换

```
uint32_t htonl(uint32_t hostint32)

uint32_t ntohl(uint32_t netint32)

uint16_t htons(uint16_t hostint16)

uint16_t ntohs(uint16_t netint16)
```

地址格式： 套接字描述网络进程地址的数据结构

```
struct sockaddr
{
    sa_family_t sa_family;  //必须有的字段，代表地址簇
    char sa_data[];//基于特定系统的自己的实现
}
```

标准

```
struct in_addr
{
    in_addr_t s_addr;
}

struct sockaddr_in
{
    sa_family_t sa_family;  //必须有的字段，代表地址簇
    in_port_t sin_port;//端口
    struct in_addr sin_addr;//IP地址
    unsigned char sin_zero[8]; //全部被0填充
}
```

linux下实现

IP地址序列化(人类习惯的点分十进制字符串表达)：

const char *inet_ntop(int domain, const void *addr, char *str, socklen_t size)

succ，re 序列化后的地址字符串， else re NULL

int inet_pton(int domain, const char *str, void *addr);

succ, re 1; 格式错误 re 0， 失败 re -1

套接字与地址绑定

int bind(int fd, const struct sockaddr *addr, socklen_t len); 使得套接字在全网具有唯一的身份

成功，返回 0， 出错， 返回-1

如果你这个进程希望被其他的进程主动的连接，那就要bind绑定一个地址，并且这个地址对于其他进程是已知的公共地址。

如果本进程总是主动的去连接其他的进程，那么本身就没必要绑定，这样在系统发送数据的时候，会给自己的套接字绑定一个可用的地址

int getsockname(int fd, struct sockaddr *addr, socklen_t *alenp)

得到已经绑定了地址的套接字的绑定地址
成功返回 0， 失败，返回-1

int getpeername(int fd, struct sockaddr *addr, socklen_t *alenp)

如果套接字已经和对等方连接，那么返回对等方的地址
成功，返回0，失败，返回-1

套接字与建立连接_connect

int connect(int fd, const struct sockaddr *addr, socklen_t len);

成功，返回0，失败，返回-1

1. 处理connect产生的错误

2. SOCK_DGRAM时采用connect的原因

套接字与建立连接_listen

int listen(int sockfd, int backlog);

成功，返回0，失败，返回-1

1. backlog值得是允许同时进行三次握手的最大连接数目，一旦队列满，系统会拒绝多余的连接请求

套接字与建立连接_accept

int accept(int sockfd, struct sockaddr *addr, socklen_t len);

成功，返回文件描述符，失败，返回-1

1. 如果没有连接请求到来，accept会阻塞直到请求到来，如果sockfd为非阻塞的模式，那么accept在没有连接请求的时候直接返回-1，并设置错误号为EAGAIN或者EWOULDBLOCK

2.返回的是文件描述符与调用connect的客户端相连，这个文件描述符与传给accept的描述符具有相同的套接字类型和地址族。原始的套接字没有关联到这个连接，而是继续保持可用状态接受客户端的连接。

数据传输_发送

ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);

成功，返回发送的字节数，失败，返回-1

ssize_t sendto(int sockfd, const void *buf, size_t nbytes, int flags, const struct sockaddr *destaddr, socklen_t destlen);

成功，返回发送的字节数，失败，返回-1

ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);

成功，返回发送的字节数，失败，返回-1

1. flags用来设置其他的高级特性，例如允许带外数据，一般默认设置为0

2.send会发送给sockfd中绑定的对端地址。 如果send中没有绑定，那将不能使用send。一般调用了connect的文件描述符，和accept返回的文件描述符都会绑定对端地址。

3.sendto不需要绑定对端地址。需要在调用的时候显式的指定， 如果是一个面向连接的套接字，那么sendto的目的地址会被忽略

数据传输_接收

ssize_t recv(int sockfd, const void *buf, size_t nbytes, int flags);

成功，返回接收的字节数，失败，返回-1，如果已经没有数据或对等方已经按序结束，返回0

ssize_t recvfrom(int sockfd, const void *buf, size_t nbytes, int flags, struct sockaddr *destaddr, socklen_t *destlen);

成功，返回接收的字节数，失败，返回-1，如果已经没有数据或对等方已经按序结束，返回0, 并且获得发送方的地址信息

ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

成功，返回接收的字节数，失败，返回-1，如果已经没有数据或对等方已经按序结束，返回0

套接字选项

可以获取和设置下面三种：

通用选项，工作在所有的套接字类型，level设置为SOL_SOCKET

在套接字层次管理的选项，但是依赖下层协议的支持

特定于某协议的选项，每个协议独有的

int setsockopt(int sockfd, int level, int option, const void *val, socklen_t len);

成功，返回0，失败，返回-1

int getsockopt(int sockfd, int level, int option, void *val,socklen_t *lenp);

成功，返回0， 失败，返回-1

带外数据

带外数据：

与普通数据相比，它允许更高优先级的数据传输，带外数据先行传输，即使发送队列已有数据。TCP支持一个字节的紧急数据，如果发送的紧急数据超过一个字节，那只把最后一个字节视为紧急数据，UDP不支持。

如何产生？

在send的时候，将flag指定MSG_OOB。

如何处理紧急数据？

收到紧急数据时，会产生SIGURG信号， 如果设置了套接字的拥有进程，该信号会被拥有进程接收

fcntl(sockfd, F_SETOWN, pid);

fcntl(sockfd, F_GETOWN, 0);

紧急标记:将紧急数据放在普通的数据流中，提供一个紧急标记来标示

int sockatmark(int sockfd); //在标记处，返回1， 没在标记处， 返回0， 出错，返回-1

当下一个要读取的字节在紧急标志处时，返回1；