1.　创建套接字：

接口声明：int socket(int domain, int type, int protocol);

参数：
domain：域。
　　AF_INET/PF_INET：　网际协议
　　AF_UNIX/PF_UNIX：本地协议，可写成 AF_LOCAL/PF_LOCAL
type：类型。
　　SOCK_STREAM：流式套接字
　　SOCK_DGRAM：数据报套接字
protocol：协议。
　　一般为 0

返回值：
成功：待连接套接字
失败：-1

备注：在网际协议中，选择流式套接字就代表 TCP 协议，选择数据包套接字就代表 UDP 协议，第三个参数 protocol 一般都不用。

==========

2、绑定套接字与网络地址

接口声明：int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

参数：
sockfd：待连接套接字
addr：包含本地地址（IP+PORT）的通用地址结构体的指针
addrlen：地址结构体大小

返回值：
成功：0
失败：-1

备注：
通用地址结构体的定义：

```
struct sockaddr
{
　　sa_family_t  sa_family;
　　char　　sa_data[14];
```

```
};
```

特殊地址结构体 —— IPv4 地址结构体：

```
struct sockaddr_in
{
	u_short sin_family;   // 地址族
	u_short sin_port;     // 端口
	struct in_addr sin_addr;   // IPV4 地址
	char sin_zero[8];
};

struct in_addr
{
	in_addr_t s_addr;     // 无符号 32 位网络地址
};
```

特殊地址结构体 —— IPv6 地址结构体：

```
struct sockaddr_in6
{
	u_short sin6_family;    // 地址族
	__be16 sin6_port;       // 端口
	__be32 sin6_flowinfo; // 流信息
	struct in6_addr sin6_addr; // IPv6 地址
	__u32 sin6_scope_id;
};
```

特殊地址结构体 ——UNIX 域地址结构体：

```
struct sockaddr_un
{
	u_short sun_family;   // 地址族
	char sun_path[108];   // 套接字文件路径
};
```

==========

1. 将待连接套接字设置为监听套接字，并设置最大同时接收连接请求个数

接口声明：int listen(int sockfd, int backlog);

参数：
sockfd：待连接套接字
backlog：最大同时接收连接请求个数

返回值：

成功：0，并将 sockfd 设置为监听套接字
失败：-1

备注：
由于历史原因，各种系统对 backlog 的理解并不一致，以 LINUX 为例，监听端能同时接收的最大连接请求个数为 backlog+4

==========

4、等待对端连接请求

接口声明：int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

参数：
sockfd：监听套接字
addr：通用地址结构体，用以存储对端地址（IP+PORT）
addrlen：参数 addr 的存储区域大小

返回值：
成功：已连接套接字（非负整数）
失败：-1

==========

5、连接对端监听套接字

接口声明：int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

参数：
sockfd：待连接套接字
addr：包含对端地址（IP+PORT）的通用地址结构体的指针
addrlen：地址结构体大小

返回值：
成功：0
失败：-1

==========

1. 断开本端连接套接字

接口声明：int close(int fd);

参数：
fd：已连接套接字

返回值：
成功：0
失败：-1

备注：
同时断开读端和写端

==========

1. 断开本端连接套接字

接口声明：int shutdown(int sockfd, int how);

参数：
sockfd：已连接套接字
how：断开方式。
SHUT_RD：关闭读端
SHUT_WR：关闭写端
SHUT_RDWR：同时关闭读写端

返回值：
成功：0
失败：-1

备注：
在只关闭一端的时候，另一端可以继续使用。

==========

7.1、将文本地址转化为二进制地址

接口声明：int inet_pton(int af, const char *src, void *dst);

参数：
af：地址族。
AF_INET：IPv4 地址
AF_INET6：IPv6 地址
src：指向“点分式”IPv4 或 IPv6 地址的指针，例如“192.168.1.100”
dst：类型为 struct in_addr *或者 struct in6_addr *的指针

返回值：
成功：1
失败：0 代表地址与地址族不匹配，-1 代表地址不合法

7.2、将二进制地址转化为文本地址

接口声明：const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);

参数：
af：地址族。
AF_INET：IPv4 地址
AF_INET6：IPv6 地址
src：类型为 struct in_addr *或者 struct in6_addr *的指针
dst：地址缓冲区指针，缓冲区至少
size：地址缓冲区大小，至少要 INET_ADDRSTRLEN 或者 INET6_ADDRSTRLEN 个字节

返回值：
成功：dst
失败：NULL

==========

8.1、向 TCP 套接字发送数据

接口声明：ssize_t send(int sockfd, const void *buf, size_t len, int flags);

参数：
sockfd：已连接套接字
buf：即将被发送的数据
len：数据长度
flags：发送标志。
MSG_NOSIGNAL：当对端已关闭时，不产生 SIGPIPE 信号
MSG_OOB：发送紧急（带外）数据，只针对 TCP 连接

返回值：
成功：已发送字节数
失败：-1

备注：
当 flags 为 0 时，send 与 write 作用相同。

8.2、向 UDP 套接字发送数据

接口声明：ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);

参数：
sockfd：UDP 套接字
buf：即将发送的数据
len：数据的长度
flags：发送标志，与函数 send 的 flags 完全一致
dest_addr：对端网络地址
addr_len：地址长度

返回值：
成功：已发送字节数
失败：-1

备注：
当 dest_addr 为 NULL，addrlen 为 0 时，sendto 与 send 作用一致

==========

9.1、从 TCP 套接字接收数据

接口声明： ssize_t recv(int sockfd, void *buf, size_t len, int flags);

参数：
sockfd：已连接套接字
buf：存储数据缓冲区
len：缓冲区大小
flags：接收标志
MSG_OOB：接收紧急（带外）数据

返回值：
成功：已接收字节数
失败：-1

备注：
当 flags 为 0 时，recv 与 read 作用相同。

9.2、从 UDP 套接字接收数据

接口声明：ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,

struct sockaddr *src_addr, socklen_t *addrlen);

参数：
sockfd：UDP 套接字
buf：储存数据缓冲区
len：缓冲区大小
flags：接收标志，与函数 send 的 flags 完全一致
src_addr：对端网络地址
addrlen：地址长度

返回值：
成功：已接收字节数
失败：-1

==========

10.1、多路复用

接口声明：int select(int nfds, fd_set *readfds, fd_set *writefds,

fd_set *exceptfds, struct timeval *timeout);

参数：
nfds：所有正在监测的套接字的最大值加 1
readfds：读就绪文件描述符集合
writefds：写就绪文件描述符集合
exceptfds：异常就绪文件描述符集合
timeout：超时控制

返回值：
成功：就绪文件描述符总数（当超时返回时为 0）
失败：-1

备注：
文件描述符集合操作函数：

```
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

10.2、多路复用

接口声明：int poll(struct pollfd *fds, nfds_t nfds, int timeout);

参数：
fds：监测文件描述符结构体数组
nfds：数组元素个数
timeout：超时控制

返回值：
成功：
失败：

备注：

```
struct pollfd
{
        int fd;      // 监测的文件描述符
        short events;     // 监测的状态
        short revents;    // 实际发生的状态
};
```

其中，监测的状态可以用以下宏来标记：
POLLIN：读就绪
POLLPRI：紧急数据读就绪
POLLOUT：写就绪
POLLRDHUP：对端已关闭或已关闭写端（仅对流式套接字有效）
POLLERR：发生错误

==========

1. 字节序转换
接口声明：

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

参数：
hostlong：　　主机字节序的长整型数据
hostshort：　　主机字节序的短整型数据
netlong：　网络字节序的长整型数据
netshort：　　网络字节序的短整型数据

返回值：
对应的字节序数据

作者：林世霖