## 辅助函数

### 大小端转换

#include <arpa/inet.h>

uint16\_t htons(uint16\_t hostshort);

功能：小端数据转大端、短整型转换、小端也叫主机字节序，大端也叫网络字节序

参数：

hostshort：传入端口号

返回值：

成功返回：转换之后的数据

失败返回：man没写失败情况

注意：

1、大端数据转小端 ntohs() 短整型，用于端口号

2、atoi()：把字符串转换为整型，返回值没写失败情况

3、uint32\_t htonl(uint32\_t hostlong); 长整型小端转大端，用于IPV4

4、uint32\_t ntohl(uint32\_t netlong); 长整型大端转小端，用于IPV4

6、这是整型转换，字符串配合其他函数使用 或者 使用其他函数

7、需要什么字节序直接转，如果字节序正确那结果不会改变

### 点分制十进制字符串转IP地址

#include <arpa/inet.h>

int inet\_pton(int af, const char \*src, void \*dst);

功能：把点分制十进制字符串，转为网络字节序的整型IP地址

参数：

af：地址族协议

AF\_INET：IPV4

AF\_INET6：IPV6

src：点分制十进制字符串

dst：传出参数，保存转换后的IP地址

返回值：

成功返回：1

失败返回：-1

传入的点分十进制字符串无效返回：0

#include <arpa/inet.h>

const char \*inet\_ntop(int af, const void \*src, char \*dst, socklen\_t size);

功能：把网络字节序转为，点分十进制字符串

参数：

af：地址族协议

AF\_INET：IPV4

AF\_INET6：IPV6

src：网络字节序的整型IP地址

dst：装换成功后的 点分十进制字符串 存储位置

size：第三个参数dst 的内存大小

返回值：

失败返回：NULL

成功返回：非空指针，指向第三个参数的指针

### 利用域名获取IP地址

#include <netdb.h>

extern int h\_errno;

struct hostent \*gethostbyname(const char \*name);

参数：

name：域名字符串

返回值：

成功返回：保存ip信息的结构体

失败返回：NULL

hostent结构体：

struct hostent

{

char \*h\_name; 官方域名(Official domain name)。不懂

char \*\*h\_aliases;多个域名可以访问统一主页，同一IP也可以绑定多个域名，保存除了官方域名外的其他域名。不懂

int h\_addrtype; Ipv4或者Ipv6保存在这里，不懂

int h\_length;保存IP地址的长度，不懂

char \*\*h\_addr\_list; 保存域名对应的IP地址

}

示例代码：

struct hostent \* host = gethostbyname("www.naver.com");

printf("官方域名：%s\n",host->h\_name);

for (int i = 0; host->h\_aliases[i]; i++)

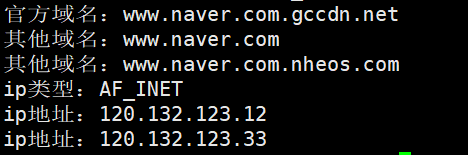
printf("其他域名：%s\n",host->h\_aliases[i]);

printf("ip类型：%s\n",(host->h\_addrtype==AF\_INET)?"AF\_INET":"AF\_INET6");

for (int i = 0; host->h\_addr\_list[i]; i++)

printf("ip地址：%s\n", inet\_ntoa(\*(struct in\_addr\*)host->h\_addr\_list[i]));

运行结果：



### 利用IP获取域名

#include <netdb.h>

#include <sys/socket.h> /\* for AF\_INET \*/

extern int h\_errno;

struct hostent \*gethostbyaddr(const void \*addr,

socklen\_t len,

int type);

功能：传入IP地址，返回域名

参数：

addr：含有IP地址的in\_addr结构体指针。

len：IPV4传4，IPV6传16

type：IPV4传AF\_INET,IPV6传AF\_INET6

返回值：

成功返回：保存ip信息的结构体

失败返回：NULL

注意：

1. 这个函数好像需要某种服务才可以使用，一般情况下函数调用不成功，返回NULL,可以尝试IP地址为127.0.0.1

示例代码：

struct sockaddr\_in addr;

memset(&addr,0,sizeof(addr));

addr.sin\_addr.s\_addr = inet\_addr("220.181.57.216");

struct hostent \* host =

gethostbyaddr((char\*)&addr.sin\_addr.s\_addr,4,AF\_INET);

printf("官方域名：%s\n", host->h\_name);

for (int i = 0; host->h\_aliases[i]; i++)

printf("其他域名：%s\n", host->h\_aliases[i]);

printf("ip类型：%s\n", (host->h\_addrtype == AF\_INET) ? "AF\_INET" : "AF\_INET6");

for (int i = 0; host->h\_addr\_list[i]; i++)

printf("ip地址：%s\n", inet\_ntoa(\*(struct in\_addr\*)host->h\_addr\_list[i]));

### 半关闭函数

#include <sys/socket.h>

int shutdown(int sockfd, int how);

参数：

sockfd：需要关闭的套接字

how：关闭的方式

SHTU\_RD：断开输入流

SHUT\_WR：断开输出流

SHUT\_RDWR：同时断开I/O流

返回值：

成功返回：0

失败返回：-1

### 端口复用

#include <sys/types.h>

#include <sys/socket.h>

// 设置套接字属性

int setsockopt(int sockfd, int level, int optname, const void \*optval, socklen\_t optlen);

功能：这里只介绍端口复用，端口复用需要在bind前调用。

参数：

sockfd: 要操作的套接字

optname: 端口复用的级别(二选一)

SO\_REUSEADDR

SO\_REUSEPORT

optval: 端口复用-> 对应的是整形数

1: 可以复用

0: 不能复用

optlen: optval参数的内存大小

返回值：

成功返回：0

失败返回：-1，设置errno

注意：

1、TCP通信时主动关闭连接方，在四次挥手时，会等待2MSL时间，才会关闭程序，这个时间会占用端口

2、设置端口复用后，只有最后一个bind这个端口的程序能使用这个端口

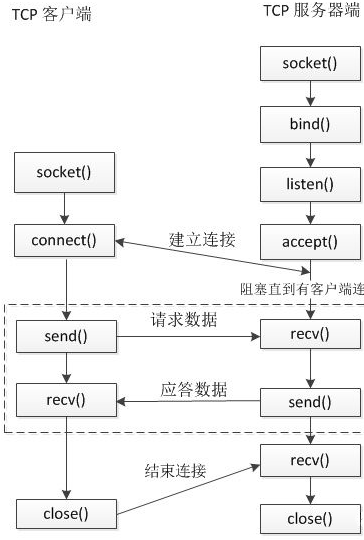
3、



## TCP通信

### tcp通信说明

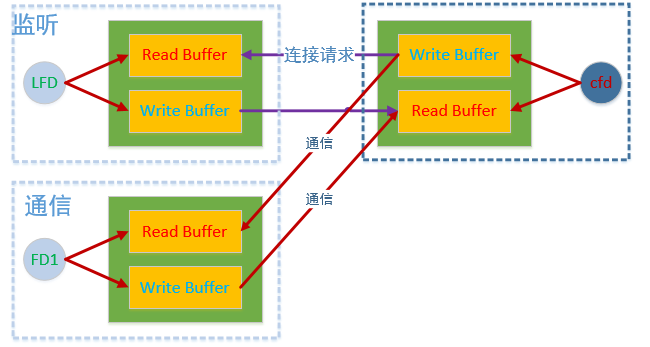
1、TCP通信流程图



2、 TCP套接字缓冲区

write写满时会阻塞

read 读时没有数据也会阻塞



3、TCP通信是面向连接的，安全的，流式传输协议。TCP建立连接时会3次握手，断开连接时会4次挥手

### 创建套接字

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int socket(int domain, int type, int protocol);

参数：

domain：地址族协议

PF\_INET：ipv4互联网协议族

PF\_INET6：ipv6互联网协议族

PF\_LOCAL：本地通信的UNIX协议族

PF\_PACKET：底层套接字协议族

PF\_IPX：IPX Novell协议族

type：通信过程中使用的协议

SOCK\_STREAM：流式协议（TCP）

SOCK\_DGRAM：报式协议（UDP）

protocol：具体的通信过程中使用的协议

0：一般写0，上个参数中的协议，默认就是TCP和UDP

IPPROTO\_TCP：满足ipv4和面向连接的协议

IPPROTO\_UDP：满足ipv4和面向消息的协议

返回值：

成功返回：文件描述符

失败返回：-1

注意：

1、domain：套接字中的协议族信息，也可以把PF开头换成AF开头

### 给套接字分配（绑定）ip地址和端口

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr \*addr,socklen\_t addrlen);

参数：

sockfd：用于监听的服务器端套接字

addr：存有地址和端口的结构体

一般使用sockaddr\_in结构体，使用时强转

addrlen：第二个结构体的长度

返回值：

成功返回：0

失败返回：-1

所需结构体：

头文件：

#include<arpa/inet.h>

struct sockaddr\_in

{

sa\_family\_t sin\_family; //地址族

AF\_INET：ipv4网络协议中使用的地址族

AF\_INET6：ipv6网络协议中使用的地址族

AF\_LOCAL：本地通信中采用的UNIX协议地址族

uint16\_t sin\_port; //TCP/UDP端口号

htons()：转大端

atoi()：把字符串转换为整型

struct in\_addr sin\_addr; //32位（四个字节）IP地址

char sin\_zero[8]; //不使用，但是必须置0，使用memset

}

struct in\_addr

{

In\_addr\_t s\_addr; //32位ip地址

htonl()：转大端

INADDR\_ANY：自动获取服务器端的IP的地址（整型）

点分制转大端：int inet\_pton(int af, const char \*src, void \*dst);

AF\_INET：

AF\_INET6

成功返回：1

失败返回：-1

传入的点分十进制字符串无效返回：0

}

注意：

1、测试时使用IP地址不是自动获取的情况下，连接时也不能使用127.1

### 进入等待连接请求状态

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int listen(int sockfd, int backlog);

参数：

sockfd：用于监听的服务器端套接字

backlog：等待处理的连接队列的长度

指定的数值不能大于 /proc/sys/net/core/somaxconn 中存储的数据, 默认为128

返回值：

成功返回：0

失败返回：-1

### 受理客服端连接请求

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

参数：

sockfd：服务器端套接字

addr：传入一个sockaddr\_in结构体用于存储客服端的地址信息

addrlen：第二个结构体的内存大小，需要socklen\_t类型的指针，所以请在外面定义，直接sizeof不行

返回值：

成功返回：用于和客服端通信的套接字

失败返回：-1

注意：

1、函数带阻塞

### 客服端请求连接

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr \*addr,socklen\_t addrlen);

参数：

sockfd：创建好的客服端套接字

addr：传入保存有服务器IP和端口sockaddr\_in结构体，

addrlen：第二个结构体的长度

返回值：

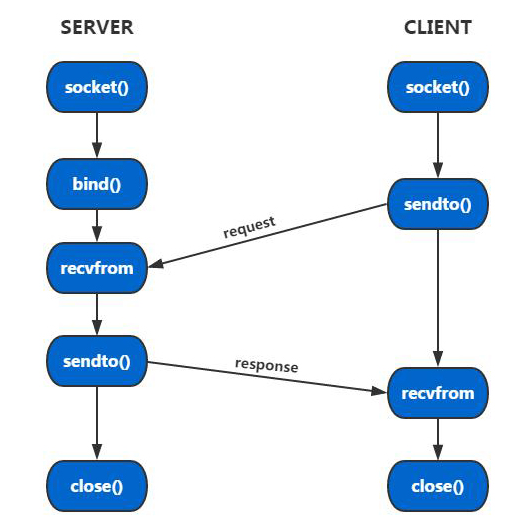
成功返回：0

失败返回：-1,

## UDP通信

### UDP通信说明

1、通信流程图



2、UDP通信时面向无连接的，不安全的，报式传输协议，效率高。

### UDP发送数据

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t sendto(int sockfd,

const void \*buf,

size\_t len,

int flags,

const struct sockaddr \*dest\_addr,

socklen\_t addrlen);

参数：

sockfd：用于传输数据的UDP套接字

buf：需要传输的数据

len：传输多少字节的数据

flags：可选项参数，传0即可

dest\_addr：存有目标地址信息的结构体

addrlen：dest\_addr的大小

返回值：

成功返回：传输的字节数

失败返回：-1

### UDP接收数据

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t recvfrom(int sockfd,

void \*buf,

size\_t len,

int flags,

struct sockaddr \*src\_addr,

socklen\_t \*addrlen);

功能：阻塞等待接收数据

参数：

sockfd：用于接收数据的UDP套接字

buf：接收到的数据

len：接收多少字节的数据

flags：可选项参数，若没有则传递0

src\_addr：保存发送端地址信息的结构体，不需要可以传NULL

addrlen：结构体src\_addr的大小

返回值：

成功返回：接收到的字节数

失败返回：-1

UDP通信示例：

1、服务器

[src\linux网络编程\UDP\UDP通信案例\servers.c](src/linux网络编程/UDP/UDP通信案例/servers.c)

2、客服端

[src\linux网络编程\UDP\UDP通信案例\client.c](src/linux网络编程/UDP/UDP通信案例/client.c)

## 案例：

### 多进程服务器

说明：使用多进程实现并发访问

[src\linux网络编程\多进程服务器\tcp.c](src/linux网络编程/多进程服务器/tcp.c)

### 多线程服务器

说明：使用多线程实现并发访问

[src\linux网络编程\多线程服务器\tcp.c](src/linux网络编程/多线程服务器/tcp.c)

### select IO多路转接

说明：

1、使用select函数监视套接字的变化，实现并发访问。

2、select可以跨平台在windows使用。

3、fd\_set集合限制，最多监视1024个套接字

#include <sys/select.h>

/\* According to earlier standards \*/

#include <sys/time.h>

#include <sys/types.h>

#include <unistd.h>

int select(int nfds,

fd\_set \*readfds,

fd\_set \*writefds,

fd\_set \*exceptfds,

struct timeval \*timeout);

参数：

nfds：需要监听的文件描述符数量，当前创建出来的最大套接字ID+1，或者直接1024

readfds：监听读缓冲区是否有数据，调用时传入需要监听的套接字集合，结束时返回读缓冲区有数据的集合，不关心可以传NULL

writefds：监听写缓冲区是否可以写入数据(缓冲区没满就可以写)，调用时传入需要监听套接字集合，结束时返回可以写入数据的套接字集合，不关心可以传NULL

exceptfds：监听套接字是否有异常，使用参照上面两个参数，不关心可以传NULL

timeval：设置单次监听最长阻塞时间，传NULL一直阻塞监听

返回值：

失败返回：-1

成功返回：发生变化的文件描述符数

返回0：表示本次监听结束没变化的文件描述符

注意：

1、下列函数可以用于用于操作fd\_set类型

void FD\_CLR(int fd, fd\_set \*set); 把某个文件描述符移除

int FD\_ISSET(int fd, fd\_set \*set);判断某个套接字是否存在

存在返回：1

不存在返回：0

void FD\_SET(int fd, fd\_set \*set);把某个文件描述符加入

void FD\_ZERO(fd\_set \*set);初始化，置0

示例代码：

[src\linux网络编程\select\_IO多路转接\select.c](src/linux网络编程/select_IO多路转接/select.c)

[src\linux网络编程\select\_IO多路转接\temp\_select.c](src/linux网络编程/select_IO多路转接/temp_select.c)

### poll

说明：

1、poll是linux下select 的简单改进版

2、使用数组管理需要监听的套接字，可以监听超过1024个套接字

#include <poll.h>

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout);

功能：和select一样，监听套接字

参数：

fds：需要监听的数组

nfds：需要监听多少个套接字

timeout：阻塞时长

0：不阻塞

-1：永久阻塞

>0：阻塞时长，单位毫秒

返回值：

失败返回：-1

成功返回：检测的数组中有多少个文件描述符变化

结构体：

struct pollfd

{

int fd; /\* 委托内核检测的文件描述符 \*/

short events; /\* 委托内核检测文件描述符的什么事件 \*/

POLLIN 读缓冲区是否有数据

POLLOUT 写缓冲区是否能写入

short revents; /\* 文件描述符实际发生的事件 \*/

};

代码示例：

[src\linux网络编程\poll\server.c](src/linux网络编程/poll/server.c)

### epoll

说明：

1、使用epoll 实现高性能并发服务器

2、epoll有两种工作模式，默认使用LT模式：读缓冲区有数据就会通知

3、ET模式高性能模式：读缓冲区发生改变才会通知

- ET模式下一般需要使用while保证数据被读完，所以建议设置套接字为非阻塞模式

- ET模式只需要在上树时加个宏即可使用

#include <unistd.h>

#include <fcntl.h>

// 设置套接字为非阻塞

// 获取套接字flag属性

int flag = fcntl(cfd, F\_GETFL);

// 添加非阻塞属性

flag = flag | O\_NONBLOCK;

// 将新的flag属性设置给cfd

fcntl(cfd, F\_SETFL, flag);

#include <sys/epoll.h>

int epoll\_create(int size);

功能：创建一颗红黑树，维护需要监视的套接字

参数:

size: 没意义, 随便写个数就行，别写0

返回值;

成功返回: 文件描述符, 操作epoll树的根节点

失败返回：-1

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

功能：对epoll 树进行管理：添加、删除节点，修改节点属性

参数:

- epfd: 树的根节点，上一个函数的返回值

- op: 要进行什么样的操作

EPOLL\_CTL\_ADD: 注册新节点, 添加到红黑树上

EPOLL\_CTL\_MOD: 修改套接字的属性

EPOLL\_CTL\_DEL: 从红黑树上删除节点

- fd: 要操作的套接字

- event: 要监视什么操作，删除节点传NULL

返回值：

成功返回：0

失败返回：-1，设置ernno

所需结构体：

struct epoll\_event {

uint32\_t events; 要监视的属性,可以位运算同时使用多个宏

- EPOLLIN 监视读

- EPOLLOUT 监视写

- EPOLLERR 监视错误

- EPOLLET ET触发模式

epoll\_data\_t data; 联合体，在下面

};

typedef union epoll\_data {

void \*ptr; // 复杂，可以使用结构体，给每一个节点一些自定义属性和方法（回调方法），但是第一个属性必须是 fd 保证能正确赋值

int fd; // 简单，套接字

uint32\_t u32;

uint64\_t u64;

} epoll\_data\_t;

int epoll\_wait(int epfd, struct epoll\_event \*events, int maxevents, int timeout);

功能：阻塞等待系统通知，处理有变化的套接字

参数：

epfd：树的根节点，第一个函数的返回值

epoll\_event：传出参数，保存发生变化的套接字，使用数组保存多个

maxevents：第二个参数的数组个数

timeout：阻塞时间

* 0，不阻塞
* -1，一直阻塞
* >0，阻塞时长，毫秒

返回值：

成功返回：变化的套接字个数

失败返回：-1

代码示例：

1、默认LT模式epoll 示例

[src\linux网络编程\epoll\epoll.c](src/linux网络编程/epoll/epoll.c)

2、ET模式代码示例

[src\linux网络编程\epoll\epoll\_et.c](src/linux网络编程/epoll/epoll_et.c)

3、ET模式、prt指针的使用

[src\linux网络编程\epoll\epoll\_et\_prt.c](src/linux网络编程/epoll/epoll_et_prt.c)

### UDP广播

不怎么懂

说明：

1、广播强制发送数据，组播可以选择是否接收数据

2、广播就是给局域网内同一网段多个设备发送数据

2、只能在局域网中使用

3、客服端只要绑定了服务器广播使用的端口，就可以收到广播数据

服务器流程：

1、创建通信UDP的套接字

2、设置套接字属性为广播，setsockopt();

3、通信，发送广播的数据，(IP地址自动获取)

4、关闭通信的套接字

客服端：

1、创建通信的UDP套接字

2、绑定广播的端口(IP地址自动获取)

3、通信，接收广播数据

4、关闭套接字

代码示例：

1、服务器

[src\linux网络编程\UDP\UDP广播\servers.c](src/linux网络编程/UDP/UDP广播/servers.c)

2、客服端

[src\linux网络编程\UDP\UDP广播\client.c](file:///D:\zm\我的笔记\src\linux网络编程\UDP\UDP广播\client.c)

网上UDP广播资料

<https://blog.csdn.net/weixin_36750623/article/details/83575412>

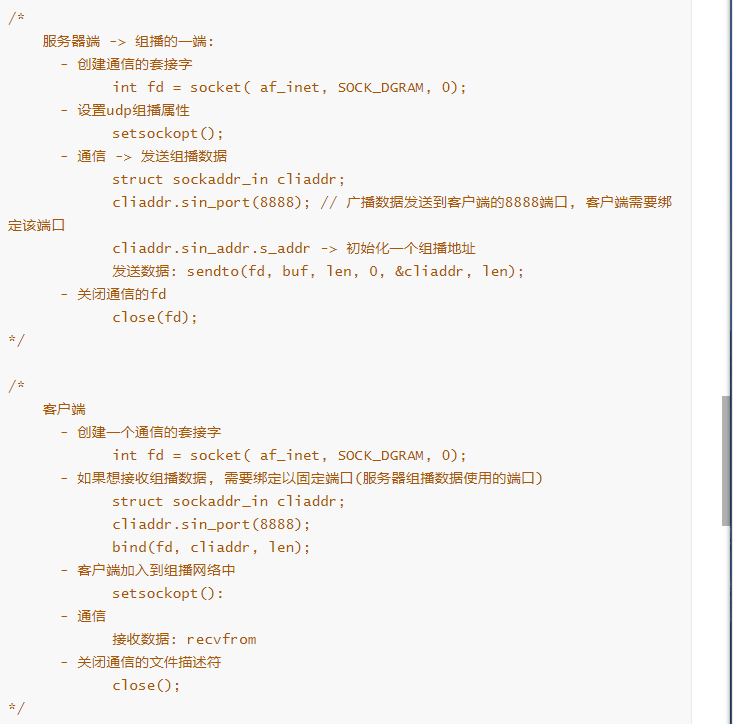
### UDP组播

不怎么懂

说明：



通信流程：



代码示例：

1、服务器

[src\linux网络编程\UDP\UDP组播\server.c](src/linux网络编程/UDP/UDP组播/server.c)

2、客服端

[src\linux网络编程\UDP\UDP广播\client.c](src/linux网络编程/UDP/UDP广播/client.c)

### TCP本地套接字

说明：

1、用于本机进程间通信

2、一般使用TCP通信

3、使用 套接字文件 替代端口和IP

代码示例：

1、服务器

[src\linux网络编程\本地套接字TCP实现\servers.c](src/linux网络编程/本地套接字TCP实现/servers.c)

2、客服端

[src\linux网络编程\本地套接字TCP实现\client.c](src/linux网络编程/本地套接字TCP实现/client.c)