**网络编程函数**

<http://www.cnblogs.com/skynet/archive/2010/12/12/1903949.html>

<http://blog.csdn.net/simba888888/article/details/9012521>

Unix/[**Linux**](http://lib.csdn.net/base/linux)基本哲学之一就是“一切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. #include<sys/types.h>
2. #include<sys/socket.h>

**3.4.1 函数socket**

创建套接字：socket函数对应于普通文件的打开操作。普通文件的打开操作返回一个文件描述字，而socket()用于创建一个socket描述符（socket descriptor），它唯一标识一个socket。

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int socket(int domain, int type, int protocol);
3. 参数说明：
4. domain：即协议域。常用的协议族有，AF\_INET(ipv4)、AF\_INET6(ipv6)、AF\_LOCAL（或称AF\_UNIX，Unix域socket）、AF\_ROUTE等等。
5. type：指定socket类型。常用的socket类型有，SOCK\_STREAM(TCP)、SOCK\_DGRAM(UDP)、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等。
6. protocol：指定协议。常用的协议有，IPPROTO\_TCP、IPPTOTO\_UDP、IPPROTO\_SCTP、IPPROTO\_TIPC等。
7. int：返回值为-1表示出错

注意：并不是上面的type和protocol可以随意组合的，如SOCK\_STREAM不可以跟IPPROTO\_UDP组合。当protocol为0时，会自动选择type类型对应的默认协议。

**3.4.2 函数bind**

命名套接字：bind()函数把一个地址族中的特定地址赋给socket。例如对应AF\_INET、AF\_INET6就是把一个ipv4或ipv6地址和端口号组合赋给socket。

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);
3. 参数说明：
4. sockfd：是由socket()调用返回的套接口文件描述符。
5. addr：传入数据结构sockaddr的指针，包括（IP，protocol，port）需要转换为通用地址类型struct sockaddr\*
6. addrlen：以设置成sizeof(struct sockaddr)
7. int：返回值，-1表示出错

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. ipv4地址结构【AF\_INET】：
2. struct sockaddr\_in {
3. sa\_family\_t    sin\_family; /\* address family: AF\_INET \*/
4. in\_port\_t      sin\_port;   /\* port in network byte order \*/
5. struct in\_addr sin\_addr;   /\* internet address \*/
6. };
8. /\* Internet address. \*/
9. struct in\_addr {
10. uint32\_t       s\_addr;     /\* address in network byte order \*/
11. };

ipv6地址结构【AF\_INET6】：

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. struct sockaddr\_in6 {
2. sa\_family\_t     sin6\_family;   /\* AF\_INET6 \*/
3. in\_port\_t       sin6\_port;     /\* port number \*/
4. uint32\_t        sin6\_flowinfo; /\* IPv6 flow information \*/
5. struct in6\_addr sin6\_addr;     /\* IPv6 address \*/
6. uint32\_t        sin6\_scope\_id; /\* Scope ID (new in 2.4) \*/
7. };
9. struct in6\_addr {
10. unsigned char   s6\_addr[16];   /\* IPv6 address \*/
11. };

Unix域地址结构【AF\_UNIX】：

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. #define UNIX\_PATH\_MAX    108
3. struct sockaddr\_un {
4. sa\_family\_t sun\_family;               /\* AF\_UNIX \*/
5. char        sun\_path[UNIX\_PATH\_MAX];  /\* pathname \*/ 使用strcpy(address.sun\_path,"server\_socket")
6. };

通常服务器在启动的时候都会绑定一个众所周知的地址（如ip地址+端口号），用于提供服务，客户就可以通过它来接连服务器；所以服务器端在listen之前会调用bind()。

而客户端就不用指定，有系统自动分配一个端口号和自身的ip地址组合，在connect()时由系统随机生成一个。由于客户端不需要固定的端口号，因此不必调用bind()，客户端的端口号由内核自动分配。注意，客户端不是不允许调用bind()，只是没有必要调用  
bind()固定一个端口号，服务器也不是必须调用bind()，但如果服务器不调用bind()，内核会自动给服务器分配监听端口，每次启动服务器时端口号都不一样，客户端要连接服务器就会遇到麻烦。

对server端的addr的初始化如下所示：

**[cpp]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. memset(&servaddr, 0, **sizeof**(servaddr));
2. servaddr.sin\_family = AF\_INET;
3. servaddr.sin\_port = htons(5188);
4. servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

首先将整个结构体清零（也可以用bzero函数），然后设置地址类型为AF\_INET，网络地址为INADDR\_ANY，这个宏表示本地的任意IP地址，因为服务器可能有多个网卡，每个网卡也可能绑定多个IP地址，这样设置可以在所有的IP地址上监听，直到与某个客户端建立了连接时才确定下来到底用哪个IP地址，端口号为5188。

**3.4.3 函数listen**

创建套接字队列

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int listen(int sockfd, int backlog);
3. 参数说明：
4. sockfd：即为要监听的socket描述字
5. backlog：相应socket可以排队的最大连接个数

第二个参数是进入队列中允许的连接的个数。进入的连接请求在使用系统调用accept()应答之前要在进入队列中等待。这个值是队列中最多可以拥有的请求的个数。大多数系统的缺省设置为20。你可以设置为5或者10。当出错时，listen()将会返回-1值。

这个函数对于backlog的解释《unix网络编程》P85是说已完成连接队列（ESTABLISHED）和未完成连接队列（SYN\_RCVD）之和（即等待连接数而非连接数）。服务器调用的accept()返回并接受这个连接，如果有大量的客户端发起连接而服务器来不及处理，尚未accept的客户端就处于连接等待状态，listen()声明sockfd处于监听状态，并且最多允许有backlog个客户端处于连接等待状态，如果接收到更多的连接请求就忽略。在计算机早期由于网络连接的处理速度很慢，几个并发的请求就可能导致系统处理不过来而引发错误，现在这个数字的意义已经不大了，现在软硬件性能几乎能保证这个队列不可能满。连接的状态变为ESTABLISHED之后（实际是连接由从半连接队列转移到了完成握手的完成队列）才表示可以被accpet()处理。

**3.4.4 函数accept**

接受连接

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);
3. 参数说明：
4. sockfd：是由socket()调用返回的套接口文件描述符。
5. addr：传出连接客户的sockaddr指针，包括（IP，protocol，port）
6. addrlen：传入指定客户结构addr的长度，返回时该值传出为连接客户地址addr的实际长度
7. int：返回值，-1表示出错

调用accept()之后，将会返回一个全新的套接口文件描述符来处理这个单个的连接。这样，对于同一个连接来说，你就有了两个文件描述符。原先的一个文件描述符正在监听你指定的端口，新的文件描述符可以用来调用send()和recv()。

可以通过对套接字文件描述符设置O\_NONBLOCK来改变其是否阻塞：

**[cpp]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. **int** flags = fcntl(socket, F\_GETFL,0);
2. fcntl(socket, F\_SETFL, O\_NONBLOCK| flags);

**3.4.5 函数connect**

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

客户端连接服务器，addr为传入参数，表示目的服务器的(ip,protocal,port)。

**3.4.6 函数read与write**

read()/write()  
recv()/send()  
readv()/writev()  
recvmsg()/sendmsg()  
recvfrom()/sendto()

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. #include <unistd.h>
2. ssize\_t read(int fd, void \*buf, size\_t count);
3. ssize\_t write(int fd, const void \*buf, size\_t count);
5. #include <sys/types.h>
6. #include <sys/socket.h>
7. ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);
8. ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);
10. ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,
11. const struct sockaddr \*dest\_addr, socklen\_t addrlen);
12. ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,
13. struct sockaddr \*src\_addr, socklen\_t \*addrlen);
15. ssize\_t sendmsg(int sockfd, const struct msghdr \*msg, int flags);
16. ssize\_t recvmsg(int sockfd, struct msghdr \*msg, int flags);

**3.4.7 函数close**

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int close(int fd);

注意：close操作只是使相应socket描述字的引用计数-1，只有当引用计数为0的时候，才会触发TCP客户端向服务器发送终止连接请求。

**3.4.8 函数getsockname**

函数返回与套接口关联的本地协议地址。

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int getsockname(int sockfd, struct sockaddr \* localaddr, socken\_t \* addrlen);

**3.4.9 函数getpeername**

函数返回与套接口关联的远程协议地址。

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int getpeername(int sockfd,struct sockaddr\* peeraddr,int\* addrlen);
3. 参数说明：
4. sockfd：是由socket()调用返回的套接口文件描述符。
5. peeraddr：传出数据结构sockaddr的指针，包括（IP，protocol，port）
6. addrlen：传出结构大小的指针
7. int：返回值，-1表示出错

**3.5 网络字节序与主机字节序转换**

将主机字节序转换为网络字节序（避免大端小端问题）

#include <netinet/in.h>

**3.5.1 函数htonl**

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. uint32\_t htonl(uint32\_t hostlong);

**3.5.2 函数htons**

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. uint16\_t htons(uint16\_t hostshort);

**3.5.3 函数ntohl**

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. uint32\_t ntohl(uint32\_t netlong);

**3.5.4 函数ntohs**

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. uint16\_t ntohs(uint16\_t netshort);

**3.6 IP地址与主机字节序转换**

**3.6.1 函数inet\_pton**

[将“点分十进制” －> “整数”]

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int inet\_pton(int af, const char \*src, void \*dst);
3. 参数说明：
4. af：地址族，AF\_INET为ipv4地址，AF\_INET6为ipv6地址
5. src：为ip地址（ipv4例如1.1.1.1）
6. dst：函数将该地址转换为in\_addr的结构体，并复制在\*dst中
7. int：返回值：如果函数出错将返回一个负值，并将errno设置为EAFNOSUPPORT，如果参数af指定的地址族和src格式不对，函数将返回0。

**3.6.2 函数inet\_ntop**

[将“整数” －> “点分十进制”]

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. const char \*inet\_ntop(int af, const void \*src, char \*dst, socklen\_t cnt);

**3.7 主机数据库函数**

**3.7.1 函数gethostname**

它返回（本地）计算机的名字，存储在hostname中，大小为size

**[plain]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. int gethostname(char\*hostname,size\_tsize);
3. 参数说明：
4. int：返回值gethostname将返回0。如果失败，它将返回-1。

**3.7.2 函数gethostbyname**

根据域名或者主机名获取信息

**[cpp]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. 函数原型：
2. **struct** hostent \*gethostbyname(**const** **char** \*name);
3. 参数说明：
4. name：主机名，如"www.baidu.com"
5. hostend：返回值。如果函数调用失败，将返回NULL。

hostend的结构如下：

**[cpp]** [view plain](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889) [copy](http://blog.csdn.net/gexiaobaohelloworld/article/details/9184889)

1. **struct** hostent
2. {
3. **char**  \*h\_name;            //表示的是主机的规范名，例如www.google.com的规范名其实是www.l.google.com
4. **char**  \*\*h\_aliases;        //表示的是主机的别名
5. **int**   h\_addrtype;         //IP地址的类型
6. **int**   h\_length;           //IP地址的长度
7. **char**  \*\*h\_addr\_list;      //主机的ip地址，注意这是以网络字节顺序保存的一个值，用inet\_ntop恢复
8. };