# **Linux高性能服务器编程**

目录

[1. 基本的API](#_Toc667091203_WPSOffice_Level1) [3](#_Toc667091203_WPSOffice_Level1)

[1. SocketAddress 的Api](#_Toc911665300_WPSOffice_Level1) [3](#_Toc911665300_WPSOffice_Level1)

[2. Socket的 Api](#_Toc21115881_WPSOffice_Level1) [3](#_Toc21115881_WPSOffice_Level1)

[A. 主机字节序和网络字节序](#_Toc911665300_WPSOffice_Level2) [3](#_Toc911665300_WPSOffice_Level2)

[B. Socket创建](#_Toc21115881_WPSOffice_Level2) [4](#_Toc21115881_WPSOffice_Level2)

[C. 命名Socket](#_Toc1044099817_WPSOffice_Level2) [4](#_Toc1044099817_WPSOffice_Level2)

[3. 网络信息Api](#_Toc1044099817_WPSOffice_Level1) [5](#_Toc1044099817_WPSOffice_Level1)

基本的API

## SocketAddress 的Api

定义在bits/socket.h中,sa\_faimly 代表地址族类型，sa\_data 存放socket地址的值。

struct sockaddr{

sa\_family\_t sa\_family;

char sa\_data[14];

}

由于sa\_data的容量太小，无法容纳多数协议的地址值，Linux 有定义了新的socketAddress结构体，socketaddr\_storage,不仅提供了足够大的空间用于存放地址值，

而且内存对齐的，\_\_ss\_padding用于填充的作用。

strut sockaddr\_storage{

sa\_family\_t sa\_family;

unsighned long int \_\_ss\_align;

char \_\_ss\_padding[128-sizeof(\_\_ss\_align)];

}

以上SocketAddress的结构体并不方便获得IP地址和端口，因此Linux为各个协议族定义了自己专门的socketAddress结构体

PF\_UNIX协议族

地址族类型通常与协议族类型一致,地址族类型与协议族类型也都定义在bits/socket.h文件中。

|  |  |
| --- | --- |
| 协议族类 | 地址族类型 |
| PF\_UNIX | AF\_UNIX |
| PF\_INET | AF\_INET |
| PF\_INET6 | AF\_INET6 |

## Socket的 Api

定义在sys/socket.h 头文件中，包括创建socket ,命名socket,监听socket,接受连接，发起连接，读写数据，获取地址，检测外带标记；

### 主机字节序和网络字节序

现代CPU的累加器可以一次加载4个字节（32位的机器），那么四个字节在内存中的排序，将影响被装载的值；大端字节序是指一个整数的高位字节存储到内存的地低地址处；小端字节序是指整数的高位排在内存高地址处；现代的PC大部分采用小端字节序，网络采用大端字节序；

在netinet/in.h文件中提供了主机字节序和网络字节序之间的转换；

//长整数据主机转成网络字节序 （IP地址转换）

Unsighned long int htonl(unsighned long int hostLong);

//长整数据网络转成主机字节序 （IP地址转换）

Unsighned long int ntohl(unsighned long int hostLong);

//短整数据主机转成网络字节序 （port端口转换）

Unsighned short int htons(unsighned shortint hostShort);

//短整数据网络转成主机字节序 （port端口转换）

Unsighned short int ntohs(unsighned long int hostShort);

### Socket创建

定义在sys/socket.h文件中

@Param domain 表示协议PF\_UNIX,PF\_INET,PF\_INET6

@Param type 若是SOCKET\_STREAM（传输层是TCP协议）若是SOCKET\_DGRAM

表示传输层是UDP协议

@Param protocol 若为0表示选择默认的协议；

@return 返回一个文件描述符号

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

### 命名Socket

在创建socket时候，我们指定了地址族，但是并未指定使用该地址族的具体那个scoketaddress，我们将一个socket 与socket地址绑定称为socket命名；

定义在sys/socket.h

@Param socketfd socket文件描述符 创建socket的时候返回的文件描述符

@Param my\_addr 表示socketaddress分配给未命名的sockfd文件描述符号

@Param addrlen

@return 表示绑定的结果 0表示成功

int bind(int socketfd, const struct sockaddr \*my\_addr, socklen\_t addrlen);

### 监听socket

socket被命名之后，还不能立刻接受客户的连接，我们需要使用系统调用来创建一个监听队列以存放待处理的客户连接；

定义在sys/socket.h

@Param socketfd 表示文件描述符号

@backlog 内核监听队列的最大长度 在Linux 2.2版本之前包括半连接状态（SYN\_RCVD）和完全连接状态(ESTABLISHED)

@Return 0表示成功

int listen(int socketfd, int backlog)

### 接受连接

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

### 发起连接

客户端通过下面的方法主动与服务端发起连接

定义在sys/socket.h文件中

@param server\_addr 是服务器监听的socket地址

＠return 0表示连接成功

int connet(int sockfd, const struct sockaddr\* serv\_addr, socklen\_t addrlen);

### 关闭了连接

关闭一个连接实际就是关闭该连接对应的socket;

定义在unistd.h文件中

int close(int sockfd);

close并不是立即关闭一个连接，而是将sockfd的引用计数减少１，只有当引用计数为０的时候有才真正关闭连接；在多进程程序中，一次fork系统调用默认使父进程中打开的socket的引用计数加１；因此我们必须在父进程和子进程中都对该socket 执行close调用才能将连接关闭；如果无论如何都有立即终止连接可以使用shutdown系统调用；

@param howto SHUT\_RD 表示关闭读的操作；SHUT\_WR 表示关闭写的操作；SHUT\_RDWR 表示关闭读写；

@return ０表示关闭成功

int shutdown(int sockfd, int howto);

## 数据的读写

### TCP数据的读写

定义在sys/socket.ｈ文件中，

读取sockfd上的数据，

＠param buf 是指定的读缓存区的位置

＠len 缓存区的大小

@flags 通常设置为０

＠return 返回读取到的数据的长度；若为０对方已经关闭了连接；若为-1表示出错；

ssize\_t recv(int sockfd,void\* buf,size\_t len, int flags);

往sockfd上写入数据

＠param buf表示写缓存区的位置

＠len 表示写的缓存区的大小

＠flags 参考一下列表　可以是某项或者通过几项的或

ssize\_t send(int sockfd,const void\*buf,size\_t len, int flags);

Flags选项名称

|  |  |  |  |
| --- | --- | --- | --- |
| 选项名字 | 含义 | send | recv |
| MSG\_CONFIRM | 提示数据链路层持续监听对方响应，直到得到答复；它仅限于SOCK\_DGRAM 和SOCK\_RAW的socket | Y | N |
| MSG\_DONTROUTE | 不查看路由表，直接将数据发送给本地局域网主机。表示发送者确切知道目标主机就在本地局域网内 | Ｙ | N |
| MSG\_DONTWAIT | 对scoket的此次操作是非堵塞的 | Y | Y |
| MSG\_MORE | 告诉内核应用程序还有更多的数据发送，内核将超时等待新数据写入TCP发送缓存区后一并发送。这样可以防止ＴCP发送过多的小的报文段，从而提高传输效率。 | y | N |
| MSG\_PEEK | 探寻读缓存区中的数据，此次操作不会导致这些数据清除 | N | Y |
| MSG\_WAITALL | 读操作在读取到一定数量的数据才会返回 | N | Y |
| MSG\_OOB | 发送或者接收紧急数据 | Ｙ | Y |
| MSG\_NOSIGNAL | 往读端关闭的管道或者socket连接时候写数据不引发SIGPIPE | Y | N |

## 网络信息Api

定义在/sys/netdb.h,实现主机名和IP地址之间的转换，以及服务名称和端口之间的转换；