Linux 的 socket 编程

一、基本概念

1、本质

在 Linux 环境下，socket 是表示进程间网络通信的特殊文件类型。本质为内核借助缓冲区形成的伪文件，我们可以使用文件描述符引用套接字。与管道类似的，Linux 系统将其封装成文件的目的是为了统一接口，使得读写套接字和读写文件的操作一致。区别是管道主要应用于本地进程间通信，而套接字多应用于网络进程间数据的传递。
注：在网络中，“IP 地址+端口号”唯一标识网络通讯中的一个进程。

2、特点

在网络通信中，套接字一定是成对出现的。一端的发送缓冲区对应对端的接收缓冲区。套接字的一个文件描述符对应发送缓冲区和接收缓冲区两个缓冲区。

二、网络字节序

1、大端/小端字节序

内存中的多字节数据相对于内存地址有大端和小端之分，磁盘文件中的多字节数据相对于文件中的偏移地址也有大端小端之分。网络数据流同样有大端小端之分，TCP/IP 协议规定，网络数据流应采用大端字节序，即低地址高字节。

2、网络字节序和主机字节序的转换

为使网络程序具有可移植性，使同样的 C 代码在大端和小端计算机上编译后都能正常运行，可以调用以下库函数做网络字节序和主机字节序的转换：

```
#include <arpa/inet.h>

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

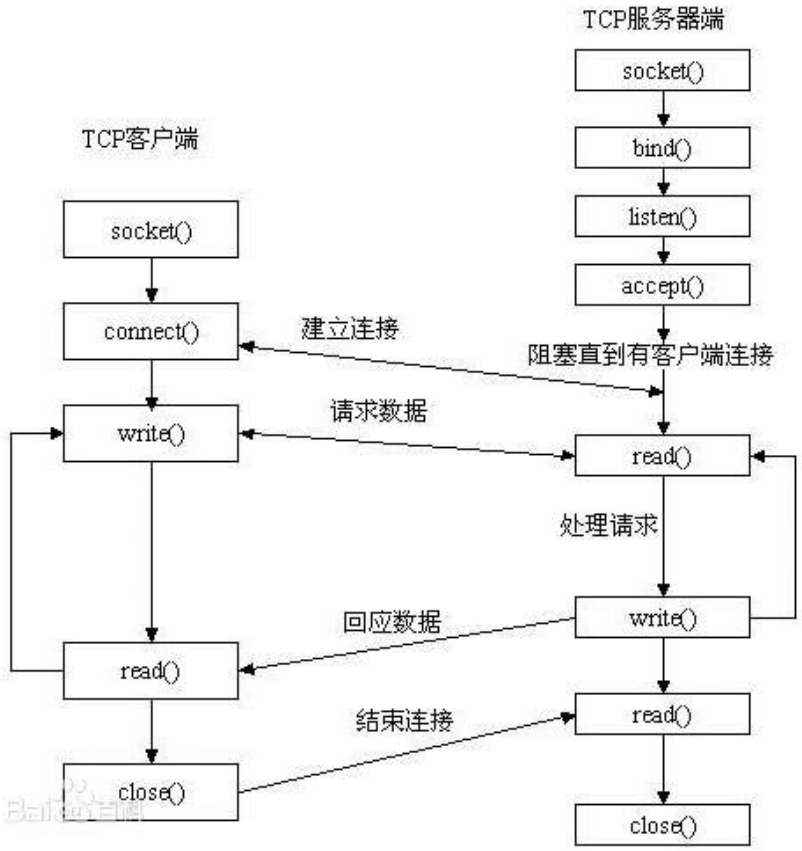
（h 表示 host，n 表示 network，l 表示 32 位长整数，s 表示 16 位短整数。）

3、点分十进制 IP 地址与网络字节序转换函数

```
#include <arpa/inet.h>
int inet_pton(int af, const char *src, void *dst);
const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
```

三、基于 TCP 协议的 socket 编程

1、模型创建流程图



2、sockaddr 结构体

早期的网络编程函数中使用 sockaddr 结构体，后来为了适应网络的发展修改为 sockaddr_in 和 sockaddr_in6 结构体，但是函数参数任然为 sockaddr，所以在编程时使用 sockaddr_in 结构体定义变量，在给函数传参时需要强转为 sockaddr *，例如 bind、accept、connect 等函数。

```
struct sockaddr_in {
    __kernel_sa_family_t sin_family;    //地址结构类型
    __be16 sin_port;                    //端口号（网络字节序）
```

```
    struct in_addr sin_addr;            //IP 地址（网络字节序）
    ......
};
struct in_addr {
    __be32 s_addr;
};
```

IPv4、IPv6 和 Unix Domain Socket 的地址类型分别定义为常数 AF_INET、AF_INET6、AF_UNIX。

3、函数介绍

1）socket 函数：

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

domain：

AF_INET 这是大多数用来产生 socket 的协议，使用 TCP 或 UDP 来传输，用 IPv4 的地址

AF_INET6 与上面类似，不过是来用 IPv6 的地址

AF_UNIX 本地协议，使用在 Unix 和 Linux 系统上，一般都是当客户端和服务器在同一台及其上的时候使用

type：

SOCK_STREAM 这个协议是按照顺序的、可靠的、数据完整的基于字节流的连接。这是一个使用最多的 socket 类型，这个 socket 是使用 TCP 来进行传输。

SOCK_DGRAM 这个协议是无连接的、固定长度的传输调用。该协议是不可靠的，使用 UDP 来进行它的连接。

SOCK_SEQPACKET 该协议是双线路的、可靠的连接，发送固定长度的数据包进行传输。必须把这个包完整的接受才能进行读取。

SOCK_RAW socket 类型提供单一的网络访问，这个 socket 类型使用 ICMP 公共协议。（ping、traceroute 使用该协议）

SOCK_RDM 这个类型是很少使用的，在大部分的操作系统上没有实现，它是提供给数据链路层使用，不保证数据包的顺序

protocol：传 0 表示使用默认协议。

返回值：成功：返回指向新创建的 socket 的文件描述符，失败：返回-1，设置 errno

描述：socket()打开一个网络通讯端口，如果成功的话，就像 open()一样返回一个文件描述符，应用程序可以像读写文件一样用 read/write 在网络上收发数据，如果 socket()调用出错则返回-1。对于 IPv4，domain 参数指定为 AF_INET。对于 TCP 协议，type 参数指定为 SOCK_STREAM，表示面向流的传输协议。如果是 UDP 协议，则 type 参数指定为 SOCK_DGRAM，表示面向数据报的传输协议。protocol 参数指定为 0 即可。

2）bind 函数：

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

描述：服务器程序所监听的网络地址和端口号通常是固定不变的，客户端程序得知服务器程序的地址和端口号后就可以向服务器发起连接，因此服务器需要调用 bind 绑定一个固定的网络地址和端口号。bind()的作用是将参数 sockfd 和 addr 绑定在一起，使 sockfd 这个用于网络通讯的文件描述符监听 addr 所描述的地址和端口号。struct sockaddr *是一个通用指针类型，addr 参数实际上可以接受多种协议的 sockaddr 结构体，而它们的长度各不相同，所以需要第三个参数 addrlen 指定结构体的长度。如：

struct sockaddr_in servaddr;

```
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(6666);
```

首先将整个结构体清零，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 6666。

3）listen 函数：

```
int listen(int sockfd, int backlog);
```

描述：典型的服务器程序可以同时服务于多个客户端，当有客户端发起连接时，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 个客户端处于待连接状态（并不是最多可以连接 backlog 个客户端），如果接收到更多的连接请求就忽略。listen()成功返回 0，失败返回-1。

4）accept 函数：

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

描述：三方握手完成后，服务器调用 accept()接受连接，如果服务器调用 accept()时还没有客户端的连接请求，就阻塞等待直到有客户端连接上来。addr 是一个传出参数，accept()返回时传出客户端的地址和端口号。addrlen 参数是一个传入传出参数（value-result argument），传入的是调用者提供的缓冲区 addr 的长度以避免缓冲区溢出问题，传出的是客户端地址结构体的实际长度（有可能没有占满调用者提供的缓冲区）。如果给 addr 参数传 NULL，表示不关心客户端的地址。

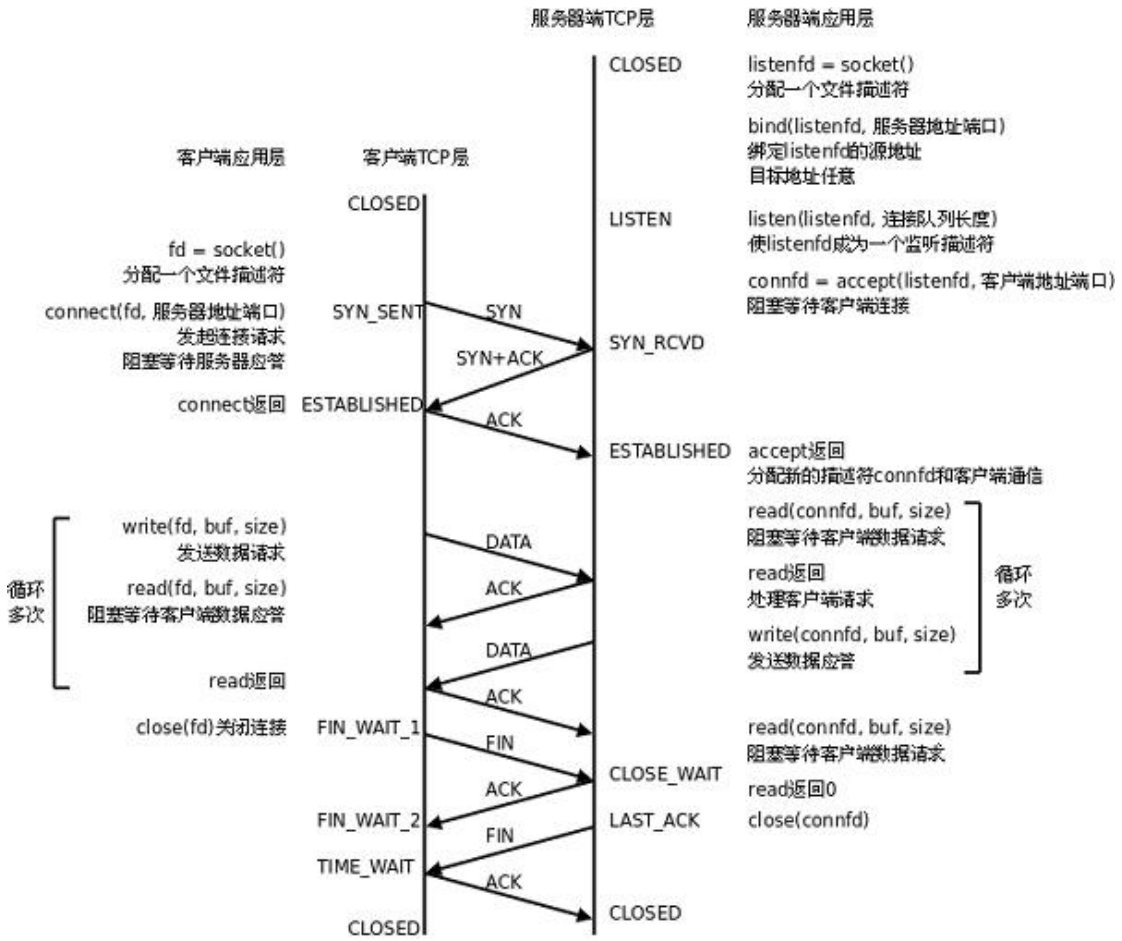
5）connect 函数：

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

描述：客户端需要调用 connect()连接服务器，connect 和 bind 的参数形式一致，区别在于 bind 的参数是自己的地址，而 connect 的参数是对方的地址。connect()成功返回 0，出错返回-1。

4、C/S 模型-TCP

1）基于 TCP 协议的客户端/服务器程序的一般流程：



服务器调用 socket()、bind()、listen()完成初始化后，调用 accept()阻塞等待，处于监听端口的状态，客户端调用 socket()初始化后，调用 connect()发出 SYN 段并阻塞等待服务器应答，服务器应答一个 SYN-ACK 段，客户端收到后从 connect()返回，同时应答一个 ACK 段，服务器收到后从 accept()返回。

2）server：作用是从客户端读字符，然后将每个字符转换为大写并回送给客户端。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <ctype.h>

#define MAXLINE 80
#define SERV_PORT 6666

int main(void)
{
    struct sockaddr_in servaddr, cliaddr;
```

```
    socklen_t cliaddr_len;
    int listenfd, connfd;
    char buf[MAXLINE];
    char str[INET_ADDRSTRLEN];
    int i, n;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(SERV_PORT);

    bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr));
    listen(listenfd, 20);

    printf("Accepting connections ...\n");
    cliaddr_len = sizeof(cliaddr);
    connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &cliaddr_len);
    printf("received from %s at PORT %d\n",
                    inet_ntop(AF_INET, &cliaddr.sin_addr, str, sizeof(str)),
                    ntohs(cliaddr.sin_port));

    while (1)
    {
            n = read(connfd, buf, MAXLINE);
            for (i = 0; i < n; i++)
                    buf[i] = toupper(buf[i]);
            write(connfd, buf, n);
            if (n == 2 && buf[0] == 'Q')
                break;
    }
    close(listenfd);
    close(connfd);
    return 0;
}
```

3）client：作用是从标准输入读入一行字符串发给服务器，然后接收服务器返回的字符串并打印，输入‘q’或者‘Q’断开连接。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>

#define MAXLINE 80
#define SERV_PORT 6666

int main(int argc, char *argv[])
{
        struct sockaddr_in servaddr;
        char buf[MAXLINE];
        int sockfd, n;
        char str[1024];

        sockfd = socket(AF_INET, SOCK_STREAM, 0);

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr);
        servaddr.sin_port = htons(SERV_PORT);

        connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

        while(1)
        {
                fgets(str, sizeof(str), stdin);
                write(sockfd, str, strlen(str));

                n = read(sockfd, buf, MAXLINE);
                if (n == 2 && buf[0] == 'Q')
                        break;
                write(STDOUT_FILENO, buf, n);
                printf("\n");
        }
        close(sockfd);
        return 0;
}
```

四、基于 UDP 协议的套接字编程

五、本地套接字编程（socket IPC）