0215socket

编写udp代码

```
NAME
       socket - create an endpoint for communication

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

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

DESCRIPTION
       socket() creates an endpoint for communication and returns a descriptor.
```

```
struct sockaddr_in
{
  __SOCKADDR_COMMON (sin_);
  in_port_t sin_port;     /* Port number.  */
  struct in_addr sin_addr;    /* Internet address.  */
                  端口号和ip
  /* Pad to size of `struct sockaddr'.  */
  unsigned char sin_zero[sizeof (struct sockaddr) -
        __SOCKADDR_COMMON_SIZE -
        sizeof (in_port_t) -
        sizeof (struct in_addr)];
};
```

Name	Purpose	Man page
AF_UNIX, AF_LOCAL	Local communication	**unix**(7)
AF_INET	IPv4 Internet protocols	**ip**(7)
AF_INET6	IPv6 Internet protocols	**ipv6**(7)
AF_IPX	IPX - Novell protocols	
AF_NETLINK	Kernel user interface device	**netlink**(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	**x25**(7)
AF_AX25	Amateur radio AX.25 protocol	
AF_ATMPVC	Access to raw ATM PVCs	
AF_APPLETALK	Appletalk	**ddp**(7)
AF_PACKET	Low level packet interface	**packet**(7)

我们一般用这个

```
BIND(3P)                    POSIX Programmer's Manual                    BIND(3P)

PROLOG
       This  manual page is part of the POSIX Programmer's Manual.  The Linux implemen-
       tation of this interface may differ (consult the corresponding Linux manual page
       for  details  of  Linux  behavior),  or  the interface may not be implemented on
       Linux.

NAME
       bind - bind a name to a socket

SYNOPSIS
       #include <sys/socket.h>

       int bind(int socket, const struct sockaddr *address,
              socklen_t address_len);
```

// 2. bind绑定：将用户设置的ip和port在内核中和当前的进程强关联

```
// 2. bind绑定：将用户设置的ip和port在内核中和当前的进程强关联
// '192.168.1.1' -> 点分十进制风格的ip地址 其实每一个区间都是[0,255]的
// 所以存储这个理论上4字节就够了
// 这个点分十进制是给用户看的
// 所以内部必定存在：点分十进制风格的ip地址 <-> 4字节 的一个转化
struct sockaddr_in local;
// 把这个结构体清零一下 -- 用一些新的接口
bzero(&local, sizeof(local));
local.sin_family = AF_INET; // 一般就是填socket接口的第一个参数
// 服务器的ip和端口未来也是要发送对方主机的 -> 先要将数据发送到网络！
// 所以要考虑大小端问题！所以要调用一下 【主机序列转网络序列】 的接口！
local.sin_port = htons(_port);
// 这里也要将字符串风格的
// 1. ip地址 转化成 4字节的ip地址  2. 主机序列 -> 网络序列
// 有一套接口可以一次帮我们做完这些事情
```

```
PSIS
   #include <sys/socket.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>

   int inet_aton(const char *cp, struct in_addr *inp);

   in_addr_t inet_addr(const char *cp);

   in_addr_t inet_network(const char *cp);

   char *inet_ntoa(struct in_addr in);

   struct in_addr inet_makeaddr(int net, int host);

   in_addr_t inet_lnaof(struct in_addr in);

   in_addr_t inet_netof(struct in_addr in);
```

start 部分

```
}
void start()
{
    // 作为一款网络服务器，是永远不退出的！ -> 常驻进程，除非挂了
    for (;;)
    {
        // start. 读取数据
        // 分析数据和处理数据
        // end. 写回数据

    }
}
```

recvfrom后面两个参数是输出型参数

```
NAME
       recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);

       ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                        struct sockaddr *src_addr, socklen_t *addrlen);

       ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

发送数据

```
SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t send(int sockfd, const void *buf, size_t len, int flags);

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

       ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

启动服务器

```
 -rw-rw-r-- 1 yufc yufc 4284 Apr  3 11:15 udp_server.hpp
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ make
g++ -o udp_client udp_client.cc -std=c++11
g++ -o udp_server udp_server.cc -std=c++11
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ ./udp_server
Usage: ./udp_server ip port                                    启动udp服务器

(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ ./udp_server 0.0.0.0 8080
```

```
(base) [yufc@VM-12-12-centos:~/Core]$ sudo netstat -anup
[sudo] password for yufc:                                  用这个命令查看服务
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
udp        0      0 0.0.0.0:8080            0.0.0.0:*                           6452/./udp_server
udp        0      0 0.0.0.0:68              0.0.0.0:*                           1059/dhclient
udp        0      0 10.0.12.12:123          0.0.0.0:*                           668/ntpd
udp        0      0 127.0.0.1:123           0.0.0.0:*                           668/ntpd
udp6       0      0 fe80::5054:ff:fe18::123 :::*                                668/ntpd
udp6       0      0 ::1:123                 :::*                                668/ntpd
(base) [yufc@VM-12-12-centos:~/Core]$
```

```
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ ./udp_server 127.0.0.1 8080
```

127.0.0.1
叫做本地环回

通常用来进行本地网路服务器的测试

如果我们写的服务器ip是127.0.0.1 发过来的数据只会在本地的协议栈中走一圈
并不会放到网络中去

所以如果本地环回测试通过，放到网络上不行，大概率就是网络的问题

云服务器：

1. 云服务器无法bind公网ip，也不建议
2. Server来讲，我们也不推荐bind确定的ip
3. 我们推荐使用任意ip

```
    // 1. ip地址 转化成 4字节的ip地址  2. 主机序列 -> 网络序列
    // 有一套接口可以一次帮我们做完这些事情
    local.sin_addr.s_addr = _ip.empty() ? INADDR_ANY : inet_addr(_ip.c_str());
    if (bind(_sock, (struct sockaddr *)&local, sizeof local) < 0)
    {
        logMessage(FATAL, "%d: %s", errno, strerror(errno));
        exit(2);
    }
    logMessage(NORMAL, "init udp server done ... %s", strerror(errno));
    // done
    return true;
}
void start()
{
```

其实这个INADDR_ANY就是0
意思就是，让服务器在工作过程中，可以从任意IP中获取数据

```
终端
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ ./udp_server 8080
```

此时启动之后，我们的客户端的ip就是全0，表示可以从任意ip获取数据

```
Makefile  udp_client.cc  udp_server  udp_server.hpp
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ ./udp_client 127.0.0.1 8080
请输入你的信息# nihao
server  echo# nihao
请输入你的信息# halo
server  echo# halo
请输入你的信息# adsjf;kasd
server  echo# adsjf;kasd
请输入你的信息#
```

服务端一样可以了

```
SYNOPSIS
       #include <stdio.h>

       FILE *popen(const char *command, const char *type);

       int pclose(FILE *stream);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       popen(), pclose():
```

popen这个接口
1. 执行command -> pipi() fork() 调用exec系列的接口调用这个命令
2. 可以将执行结果通过FILE*类型的指针进行读取

```
                0, (struct sockaddr *)&peer, &len);
    /* These calls return the number of bytes received, or -1 if an error occurred. */
    // 返回的s是读到的字节数
    if (s > 0)
    {
        buffer[s] = 0; // 把我们当前数据当作字符串
        FILE *fp = popen(buffer, "r");
        if (nullptr == fp)
        {
            logMessage(ERROR, "popen: %d: %s", errno, strerror(errno));
            continue;
        }
        char result[256];
        std::string cmd_echo;
        while (fgets(result, sizeof(result), fp) != nullptr)
        {
            cmd_echo += result;
        }
    }
```

把我们输入命令的
最后输出的结果保存起来

再给客户端发过去

```
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$ make
g++ -o udp_client udp_client.cc -std=c++11 -g
g++ -o udp_server udp_server.cc -std=c++11 -g
(base) [yufc@VM-12-12-centos:~/Core/BitCodeField/0215/udp]$./udp_client 127.0.0.1 8080
please echo the command # ls
server  echo #
Log.hpp
Makefile
udp_client
udp_client.cc
udp.log
udp_server
udp_server.cc
udp_server.hpp
udp_server.hpp-backup

please echo the command #
```

这样就可以输入命令给客户端
让他执行了