

ARP: Address Resolution Protocol [RFC 826]

IP: Internet Protocol [RFC 791]

ICMP: Internet Control Message Protocol [RFC 777]

IGMPv1: Internet Group Management Protocol version 1[RFC 1112]

TCP: Transmission Control Protocol [RFC 793]

UDP: User Datagram Protocol [RFC 768]

AutoIP: Auto Internet Protocol

DNS: Domain Name System [RFC-1034 and RFC 1035]

DHCP: Dynamic Host Configuration Protocol [RFC 2131]

Ping: This is an application that tests the connectivity of two network systems by

sending an ICMP request and then listening for ICMP response.

BSD Socket: Berkeley Software Distribution Sockets is a generalized networking

capability provided at the application layer that is used to perform inter process

communication across a computer network. Since the BSD sockets form a de facto

standard abstraction for network sockets, the network TCP/IP stack provided by

EBSnet, Inc. uses the same guidelines in its abstraction.

Reference Applications: Sample applications are provided. See Reference

applications documentation and source code for details.

要了解ARP的作用，首先要分清两个“地址”：

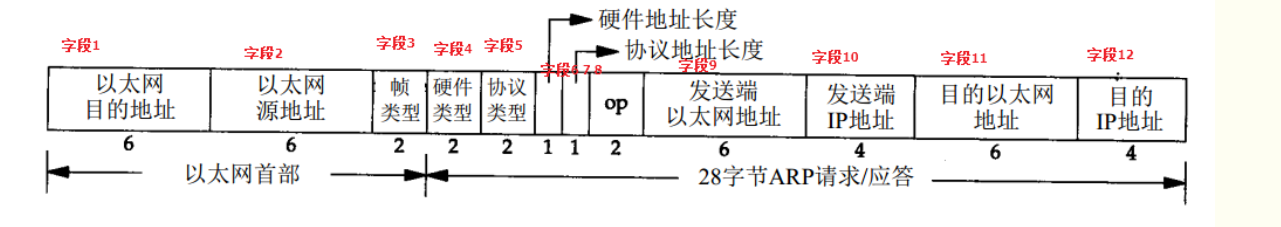
　　（1）TCP/IP的32bit IP地址。仅知道主机的IP地址不能让内核发送数据帧给主机。

　　（2）网络接口的硬件地址，它是一个48bit的值，用来标识不同的以太网或令牌环网络接口。在硬件层次上，进行数据交换必须有正确的接口地址，内核必须知道目的端的硬件地址才能发送数据。

　　简言之，就是在以太网中，一台主机要把数据帧发送到同一局域网上的另一台主机时，设备驱动程序必须知道以太网地址（**MAC地址***（****Media Access Control Address）***，直译为**媒体访问控制地址**，也称为**局域网地址**（LAN Address），**以太网地址**（Ethernet Address）或**物理地址**（Physical Address））才能发送数据。而我们只知道IP地址，这时就需要采用ARP协议将IP地址映射为以太网地址。

要注意一点，一般认为ARP协议只使适用于局域网。

ARP分组的格式如下图：



字段1是ARP请求的目的以太网地址，全1时代表广播地址。

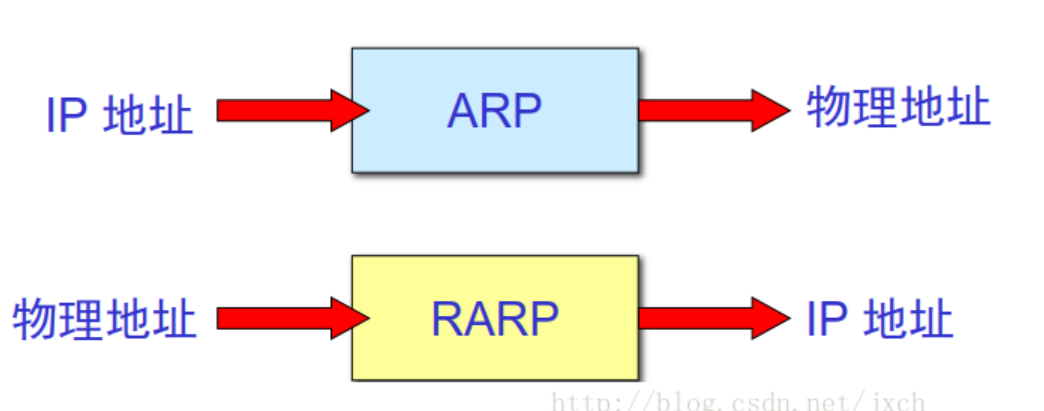
字段2是发送ARP请求的以太网地址。

字段3以太网帧类型表示的是后面的数据类型，ARP请求和ARP应答这个值为0x0806。

字段4表示硬件地址的类型，硬件地址不只以太网一种，是以太网类型时此值为1。

字段5表示要映射的协议地址的类型，要对IPv4地址进行映射，此值为0x0800。

字段6和7表示硬件地址长度和协议地址长度，MAC地址占6字节，IP地址占4字节。

字段8是操作类型字段，值为1，表示进行ARP请求；值为2，表示进行ARP应答；值为3，表示进行RARP请求；值为4，表示进行RARP应答。

字段9是发送端ARP请求或应答的硬件地址，这里是以太网地址，和字段2相同。

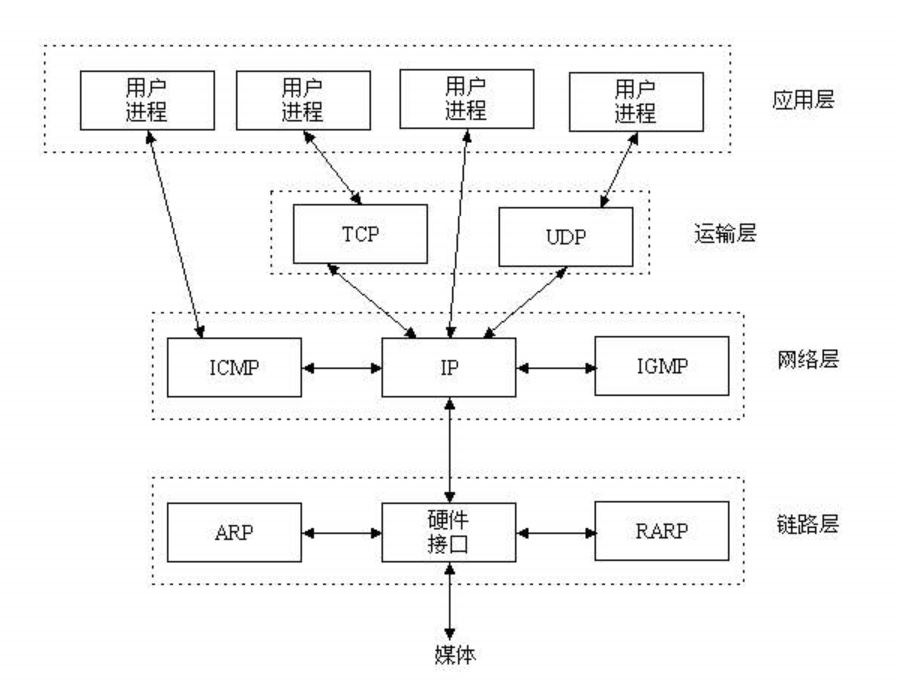
字段10是发送ARP请求或应答的IP地址。

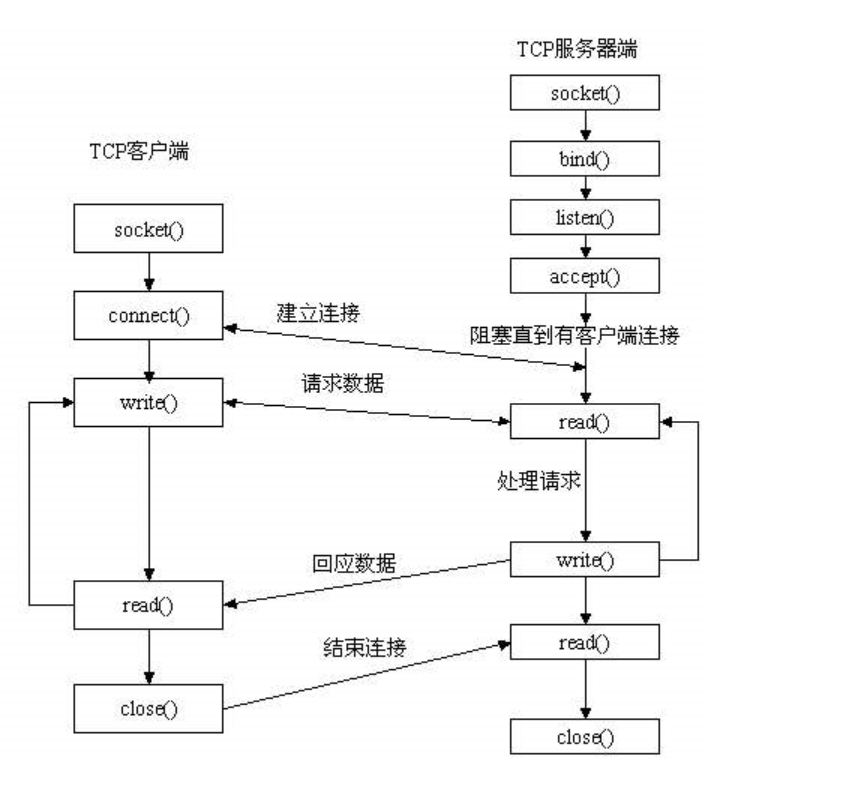
字段11和12是目的端的硬件地址和协议地址。

在windows下用gcc编译socket库时，出现错误：

**MingW上编译WinSocket程序undefined reference to `WSAStartup@8'报错的解决办法**

解决办法：需要链接一个库 gcc xx.c –l wsock32





Socket套接字几个重要函数：

1. **socket()函数**

int **socket**(int protofamily, int type, int protocol);//返回sockfd

sockfd是描述符。

  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。这个socket描述字跟文件描述字一样，后续的操作都有用到它，把它作为参数，通过它来进行一些读写操作。

      正如可以给fopen的传入不同参数值，以打开不同的文件。创建socket的时候，也可以指定不同的参数创建不同的socket描述符，socket函数的三个参数分别为：

* protofamily：即协议域，又称为协议族（family）。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。协议族决定了socket的地址类型，在通信中必须采用对应的地址，如AF\_INET决定了要用ipv4地址（32位的）与端口号（16位的）的组合、AF\_UNIX决定了要用一个绝对路径名作为地址。
* type：指定socket类型。常用的socket类型有，SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等（socket的类型有哪些？）。
* protocol：故名思意，就是指定协议。常用的协议有，IPPROTO\_TCP、IPPTOTO\_UDP、IPPROTO\_SCTP、IPPROTO\_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议（这个协议我将会单独开篇讨论！）。

**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。

当我们调用**socket**创建一个socket时，返回的socket描述字它存在于协议族（address family，AF\_XXX）空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用bind()函数，否则就当调用connect()、listen()时系统会自动随机分配一个端口。

SOCK\_STREAM是有保障的(即能保证数据正确传送到对方)面向连接的SOCKET，多用于资料(如文件)传送。

SOCK\_DGRAM 是无保障的面向消息的socket ， 主要用于在网络上发广播信息。

SOCK\_STREAM是基于TCP的，数据传输比较有保障。SOCK\_DGRAM是基于UDP的，专门用于局域网，基于广播SOCK\_STREAM是数据流,一般是tcp/ip协议的编程,SOCK\_DGRAM分是数据抱,是udp协议网络编程

### 2，bind()函数

正如上面所说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。

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

函数的三个参数分别为：

* sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。bind()函数就是将给这个描述字绑定一个名字。
* addr：一个const struct sockaddr \*指针，指向要绑定给sockfd的协议地址。这个地址结构根据地址创建socket时的地址协议族的不同而不同，如ipv4对应的是：
* struct sockaddr\_in {
* sa\_family\_t sin\_family; /\* address family: AF\_INET \*/
* in\_port\_t sin\_port; /\* port in network byte order \*/
* struct in\_addr sin\_addr; /\* internet address \*/
* };
* /\* Internet address. \*/
* struct in\_addr {
* uint32\_t s\_addr; /\* address in network byte order \*/

};

ipv6对应的是：

struct sockaddr\_in6 {

sa\_family\_t sin6\_family; /\* AF\_INET6 \*/

in\_port\_t sin6\_port; /\* port number \*/

uint32\_t sin6\_flowinfo; /\* IPv6 flow information \*/

struct in6\_addr sin6\_addr; /\* IPv6 address \*/

uint32\_t sin6\_scope\_id; /\* Scope ID (new in 2.4) \*/

};

struct in6\_addr {

unsigned char s6\_addr[16]; /\* IPv6 address \*/

};

Unix域对应的是：

#define UNIX\_PATH\_MAX 108

struct sockaddr\_un {

sa\_family\_t sun\_family; /\* AF\_UNIX \*/

char sun\_path[UNIX\_PATH\_MAX]; /\* pathname \*/

};

* addrlen：对应的是地址的长度。

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；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。这就是为什么通常服务器端在listen之前会调用bind()，而客户端就不会调用，而是在connect()时由系统随机生成一个。

**网络字节序与主机字节序**

**主机字节序**就是我们平常说的大端和小端模式：不同的CPU有不同的字节序类型，这些字节序是指整数在内存中保存的顺序，这个叫做主机序。引用标准的Big-Endian和Little-Endian的定义如下：

　　a) Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

　　b) Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

**网络字节序**：4个字节的32 bit值以下面的次序传输：首先是0～7bit，其次8～15bit，然后16～23bit，最后是24~31bit。这种传输次序称作大端字节序。**由于TCP/IP首部中所有的二进制整数在网络中传输时都要求以这种次序，因此它又称作网络字节序。**字节序，顾名思义字节的顺序，就是大于一个字节类型的数据在内存中的存放顺序，一个字节的数据没有顺序的问题了。

所以：在将一个地址绑定到socket的时候，请先将主机字节序转换成为网络字节序，而不要假定主机字节序跟网络字节序一样使用的是Big-Endian。由于这个问题曾引发过血案！公司项目代码中由于存在这个问题，导致了很多莫名其妙的问题，所以请谨记对主机字节序不要做任何假定，务必将其转化为网络字节序再赋给socket。

### 3、listen()、connect()函数

如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

int listen(int sockfd, int backlog);

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

listen函数的第一个参数即为要监听的socket描述字，第二个参数为相应socket可以排队的最大连接个数。socket()函数创建的socket默认是一个主动类型的，listen函数将socket变为被动类型的，等待客户的连接请求。

connect函数的第一个参数即为客户端的socket描述字，第二参数为服务器的socket地址，第三个参数为socket地址的长度。客户端通过调用connect函数来建立与TCP服务器的连接。

**4、accept()函数**

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。TCP客户端依次调用socket()、connect()之后就向TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen); //返回连接connect\_fd

参数sockfd

参数sockfd就是上面解释中的监听套接字，这个套接字用来监听一个端口，当有一个客户与服务器连接时，它使用这个一个端口号，而此时这个端口号正与这个套接字关联。当然客户不知道套接字这些细节，它只知道一个地址和一个端口号。

参数addr

这是一个结果参数，它用来接受一个返回值，这返回值指定客户端的地址，当然这个地址是通过某个地址结构来描述的，用户应该知道这一个什么样的地址结构。如果对客户的地址不感兴趣，那么可以把这个值设置为NULL。

参数len

如同大家所认为的，它也是结果的参数，用来接受上述addr的结构的大小的，它指明addr结构所占有的字节个数。同样的，它也可以被设置为NULL。

如果accept成功返回，则服务器与客户已经正确建立连接了，此时服务器通过accept返回的套接字来完成与客户的通信。

注意：

      accept默认会阻塞进程，直到有一个客户连接建立后返回，它返回的是一个新可用的套接字，这个套接字是连接套接字。

此时我们需要区分两种套接字，

       监听套接字: 监听套接字正如accept的参数sockfd，它是监听套接字，在调用listen函数之后，是服务器开始调用socket()函数生成的，称为监听socket描述字(监听套接字)

       连接套接字：一个套接字会从主动连接的套接字变身为一个监听套接字；而accept函数返回的是已连接socket描述字(一个连接套接字)，它代表着一个网络已经存在的点点连接。

        一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。

        自然要问的是：为什么要有两种套接字？原因很简单，如果使用一个描述字的话，那么它的功能太多，使得使用很不直观，同时在内核确实产生了一个这样的新的描述字。

连接套接字socketfd\_new 并没有占用新的端口与客户端通信，依然使用的是与监听套接字socketfd一样的端口号

1．为什么建立连接协议是三次握手，而关闭连接却是四次握手呢？

这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的建连请求后，它可以把ACK和SYN（ACK起应答作用，而SYN起同步作用）放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。

2．为什么TIME\_WAIT状态还需要等2MSL后才能返回到CLOSED状态？

这是因为虽然双方都同意关闭连接了，而且握手的4个报文也都协调和发送完毕，按理可以直接回到CLOSED状态（就好比从SYN\_SEND状态到ESTABLISH状态那样）；但是因为我们必须要假想网络是不可靠的，你无法保证你最后发送的ACK报文会一定被对方收到，因此对方处于LAST\_ACK状态下的SOCKET可能会因为超时未收到ACK报文，而重发FIN报文，所以这个TIME\_WAIT状态的作用就是用来重发可能丢失的ACK报文。

**IP地址的单播**

主机之间**“一对一”**的通讯模式，网络中的交换机和路由器对数据只进行转发不进行复制。如果10个客户机需要相同的数据，则服务器需要逐一传送，重复10次相同的工作。但由于其能够针对每个客户的及时响应，所以现在的网页浏览全部都是采用IP单播协议。网络中的路由器和交换机根据其目标地址选择传输路径，将IP单播数据传送到其指定的目的地。

**IP地址的广播**

主机之间**“一对所有”**的通讯模式，网络对其中每一台主机发出的信号都进行无条件复制并转发，所有主机都可以接收到所有信息（不管你是否需要），由于其不用路径选择，所以其网络成本可以很低廉。有线电视网就是典型的广播型网络，我们的电视机实际上是接受到所有频道的信号，但只将一个频道的信号还原成画面。在数据网络中也允许广播的存在，但其被限制在二层交换机的局域网范围内，禁止广播数据穿过路由器，防止广播数据影响大面积的主机。

1. [TCP/IP协议](https://baike.baidu.com/item/TCP%2FIP%E5%8D%8F%E8%AE%AE)栈中, [传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)只有[UDP](https://baike.baidu.com/item/UDP)可以广播.

2. 只能对同一[子网](https://baike.baidu.com/item/%E5%AD%90%E7%BD%91)内部广播, [广播数据包](https://baike.baidu.com/item/%E5%B9%BF%E6%92%AD%E6%95%B0%E6%8D%AE%E5%8C%85)不经过[路由器](https://baike.baidu.com/item/%E8%B7%AF%E7%94%B1%E5%99%A8).

3. UDP的广播地址为255.255.255.255

4，UDP都是以数据报的形式进行发送和接收的，而TCP是以数据流的形式进行发送和接收的。数据报和数据流，这两者要区分开来。

5，作为接收端，当你调用bind()函数绑定IP时使用INADDR\_ANY，表明接收来自任意IP、任意网卡的发给指定端口的数据。作为发送端，当用调用bind()函数绑定IP时使用INADDR\_ANY，表明使用网卡号最低的网卡进行发送数据，也就是UDP数据广播。

**IP地址的组播**

主机之间**“一对一组”**的通讯模式，也就是加入了同一个组的主机可以接受到此组内的所有数据，网络中的交换机和路由器只向有需求者复制并转发其所需数据。主机可以向路由器请求加入或退出某个组，网络中的路由器和交换机有选择的复制并传输数据，即只将组内数据传输给那些加入组的主机。这样既能一次将数据传输给多个有需要（加入组）的主机，又能保证不影响其他不需要（未加入组）的主机的其他通讯。

1，组播协议允许将一台主机发送的数据通过网络路由器和交换机复制到多个加入此组播的主机，是一种一对多的通讯方式。

2，与广播协议相比，只有组播接收方向路由器发出请求后，网络路由器才复制一份数据给接收方，从而节省接收方的带宽。而广播方式无论接收方是否需要，网络设备都将所有广播信息向所有设备发送，从而大量占据接收方的接入带宽。

**select函数来实现多路复用输入/输出模型，非阻塞式。**

int select(int maxfd,fd\_set \*rdset,fd\_set \*wrset,fd\_set \*exset,struct timeval \*timeout);

返回值：返回对应位仍然为1的fd的总数。注意啦：只有那些可读，可写以及有异常条件待处理的fd位仍然为1。否则为0哦。举个例子，比如recv(), 在没有数据到来调用它的时候,你的线程将被阻塞,如果数据一直不来,你的线程就要阻塞很久.这样显然不好。所以采用select来查看套节字是否可读(也就是是否有数据读了) 。

具体解释select的参数：

（1）maxfd是一个整数值，是指集合中所有文件描述符的范围，即所有文件描述符的最大值加1，不能错。

说明：对于这个原理的解释可以看上边fd\_set的详细解释，fd\_set是以位图的形式来存储这些文件描述符。maxfd也就是定义了位图中有效的位的个数。

（2）fd\_set \*rdset是指向fd\_set结构的指针，这个集合中应该包括文件描述符，我们是要监视这些文件描述符的读变化的，即我们关心是否可以从这些文件中读取数据了，如果这个集合中有一个文件可读，select就会返回一个大于0的值，表示有文件可读；如果没有可读的文件，则根据timeout参数再判断是否超时，若超出timeout的时间，select返回0，若发生错误返回负值。可以传入NULL值，表示不关心任何文件的读变化。

（3）fd\_set \*wrset是指向fd\_set结构的指针，这个集合中应该包括文件描述符，我们是要监视这些文件描述符的写变化的，即我们关心是否可以向这些文件中写入数据了，如果这个集合中有一个文件可写，select就会返回一个大于0的值，表示有文件可写，如果没有可写的文件，则根据timeout参数再判断是否超时，若超出timeout的时间，select返回0，若发生错误返回负值。可以传入NULL值，表示不关心任何文件的写变化。

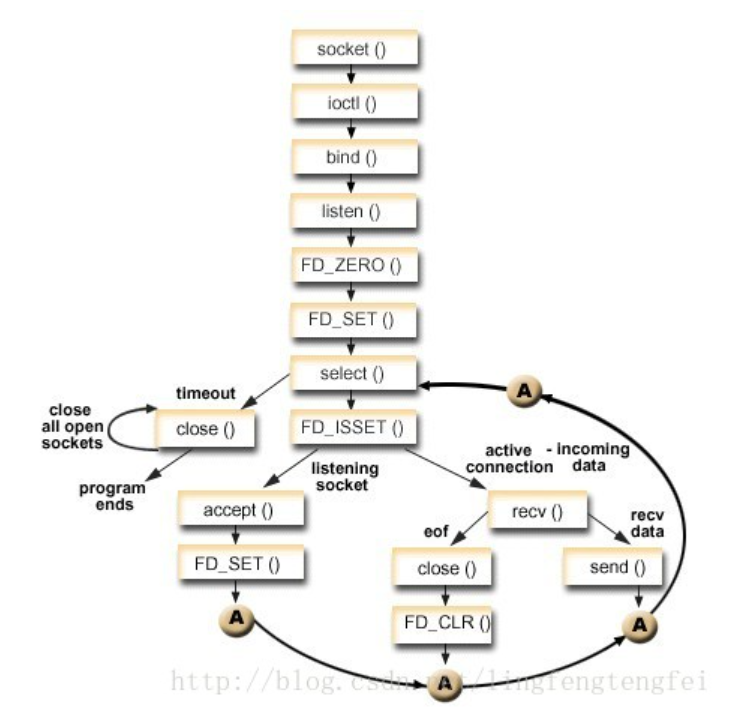
（4）fd\_set \*exset同上面两个参数的意图，用来监视文件错误异常文件。

（5）struct timeval \*timeout是select的超时时间，这个参数至关重要，它可以使select处于三种状态，第一，若将NULL以形参传入，即不传入时间结构，就是将select置于阻塞状态，一定等到监视文件描述符集合中某个文件描述符发生变化为止；第二，若将时间值设为0秒0毫秒，就变成一个纯粹的非阻塞函数，不管文件描述符是否有变化，都立刻返回继续执行，文件无变化返回0，有变化返回一个正值；第三，timeout的值大于0，这就是等待的超时时间，即 select在timeout时间内阻塞，超时时间之内有事件到来就返回了，否则在超时后不管怎样一定返回，返回值同上述。

（1）当监视的相应的文件描述符集中满足条件时，比如说读文件描述符集中有数据到来时，内核(I/O)根据状态修改文件描述符集，并返回一个大于0的数。

（2）当没有满足条件的文件描述符，且设置的timeval监控时间超时时，select函数会返回一个为0的值。

（3）当select返回负值时，发生错误。



mDNS

mdns 即多播dns（Multicast DNS），mDNS主要实现了在没有传统DNS服务器的情况下使局域网内的主机实现相互发现和通信，使用的端口为5353，遵从dns协议，使用现有的DNS信息结构、名语法和资源记录类型。并且没有指定新的操作代码或响应代码。

在局域网中，设备和设备之前相互通信需要知道对方的ip地址的，大多数情况，设备的ip不是静态ip地址，而是通过dhcp 协议动态分配的ip 地址，如何设备发现呢，就是要mdns大显身手，例如：现在物联网设备和app之间的通信，要么app通过广播，要么通过组播，发一些特定信息，感兴趣设备应答，实现局域网设备的发现，当然mdns 比这强大的多

组播地址: 组播地址使用的是D类地址，地址范围为：224.0.0.0—239.255.255.255

mdns 工作原理简单描述：

mdns 使用组播地址为： 224.0.0.251 （ipv6： FF02::FB） 端口为5353，mdns 是用于局域网内部的，并且主机的域名为.local 结尾,每个进入局域网的主机，如果开启了mDNS服务的话，都会向局域网内的所有主机组播一个消息，我是谁（域名），和我的IP地址是多少。然后其他有mdns服务的主机就会响应，也会告诉你，它是谁（域名），它的IP地址是多少。 当然设备需要服务时，就是使用mdns 查询域名对对应的ip地址，对应的设备收到该报文后同样通过组播方式应答，此时其他主机设备也是可以收到该应答报文，其他主机也会记录域名和ip 以及ttl 等，更新缓存

 使用Bonjour的设备在网络中自动组播它们自己的服务信息并监听其它设备的服务信息。设备之间就像在打招呼，这也是该技术命名为Bonjour的原因。Bonjour使得局域网中的系统和服务即使在没有网络管理员的情况下也很容易被找到。

从Bonjour角度来看，该技术主要解决了三个问题：

* Addressing：即为主机分配IP。Bonjour的Addressing处理比较简单，即每个主机在网络内部的地址可选范围内找一个IP，然后查看网络内部是否有其他主机再用。如果该IP没有被分配的话，它将使用此IP。
* Naming：Naming解决的是host名和IP地址的对应关系。Bonjour采用的是Multiple DNS技术，即DNS查询消息将通过UDP组播方式发送。一旦网络内部某个机器发现查询的机器名和自己设置的一样，就回复这条请求。此外，Bonjour还拓展了MDNS的用途，即除了能查找host外，还支持对service的查找。不过，Bonjour的Naming有一个限制，即网络内部不能有重名的host或service。
* Service Discovery：SD基于上面的Naming工作，它使得应用程序能查找到网络内部的服务，并解析该服务对应的IP地址和端口号。应用程序一旦得到服务的IP地址和端口号，就可以直接和该服务建立交互关系。

SSL

Linux内部的notify注册如何实现？

Linux内部的路由模块？

什么是token？

token的意思是“令牌”，是服务端生成的一串字符串，作为客户端进行请求的一个标识。

当用户第一次登录后，服务器生成一个token并将此token返回给客户端，以后客户端只需带上这个token前来请求数据即可，无需再次带上用户名和密码。

简单token的组成；uid(用户唯一的身份标识)、time(当前时间的时间戳)、sign（签名，token的前几位以哈希算法压缩成的一定长度的十六进制字符串。为防止token泄露）。

WFD

WiFI Display（WFD）是WiFI Alliance 开发出的一种规范，使多媒体设备之间建立和维持一个基于WiFi的连接，并且利用这个连接推进视频/音频的在目标设备的呈现播放。

WFD建立在wifi p2p连接基础上的，支持以下两种连接方式

-wifi direct:（必须支持）

   设备无需通过无线路由器即可相互连接的技术，需要一台设备作为组织者建立一个类似ap功能的网络，其他设备可以搜索到并用wifi连接上

Wi-Fi Display经常和Miracast联系在一起。实际上，Miracast是Wi-Fi联盟（Wi-Fi Alliance）对支持Wi-Fi Display功能的设备的认证名称。通过Miracast认证的设备将在最大程度内保持对Wi-Fi Display功能的支持和兼容。由此可知，Miracast考察的就是Wi-Fi Display。而Wi-Fi Display的核心功能就是让设备之间通过Wi-Fi无线网络来分享视音频数据。以一个简单的应用场景为例：有了Wi-Fi Display后，手机和电视机之间可以直接借助Wi-Fi，而无需硬连线（如HDMI）就可将手机中的视频投递到TV上去显示[[①]](http://write.blog.csdn.net/postedit" \l "_ftn1" \o "" \t "_blank)。以目前智能设备的发展趋势来看，Wi-Fi Display极有可能在较短时间内帮助我们真正实现多屏互动。

DLNA技术

DLNA(Digital Living Network Alliance)是由Sony、Intel、Microsoft等发起成立的一个认证组织，其目的也是实现消费电子产品通过有线/无线网络实现设备互联、数据互通。相比于Miracast认证项目，DLNA有着自己的一套框架和相应标准：首先DLNA将市场上几乎所有的电子产品进行了分类(图30)，一方面可以便于DLNA对众多的设备进行规范认证，另一方面也是DLNA为不同设备间进行交互制定标准的基础。

## AirPlay技术

    AirPlay以[组播DNS](http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt)(Multicast DomainName Server—mDNS)协议和[DNS服务发现](http://tools.ietf.org/html/draft-cheshire-dnsext-dns-sd-11)(DNS Service Discovery，简称DNS-SD)协议为基础，它们是[IETFn Zeroconf工作组](http://www.zeroconf.org/)提出的用于自动寻找设备及服务的网络协议，苹果公司以这两个协议为基础，实现了苹果公司数字家庭网络框架。

    AirPlay协议消息发送格式及规则基于mDNS协议，mDNS协议基于组播技术，定义了家庭各个设备之间的消息的基本格式和接收/发送规则。该协议以DNS协议为基础，并对其消息格式和消息收发顺序作出了一些修改。例如对DNS消息包头进行了简化，使其专注于实现家庭设备的互相发现；另外，考虑到使用组播技术，mDNS在降低网络拥塞和消息冗余方面也作出了很多改进，使得局域网内设备和服务的发现不会引起过多的消息交互。

     在mDNS协议的基础上，DNS-SD协议规定了一个服务宣告及使用的完整过程。即设备必须发送什么样的mDNS消息才能完整地宣告并描述自己服务。DNS-SD协议使用PTR、SRV和TXT三种类型的记录全面描述了一个服务的类型，名称以及所在主机的IP和端口号等。  
当使用DNS-SD协议实现了对设备及服务的发现和描述后，苹果公司的AirPlay协议规定了图片、音频及视频的传输和控制消息格式，从而实现了智能设备之间的媒体共享和协同动作。在通过DNS-SD获得了其他设备及服务的信息（即设备或服务的IP地址及端口号）之后，AirPlay使用HTTP消息实现了图片和视频的传输及控制，使用RSTP协议实现了音频的传输和控制

wpa\_supplicant