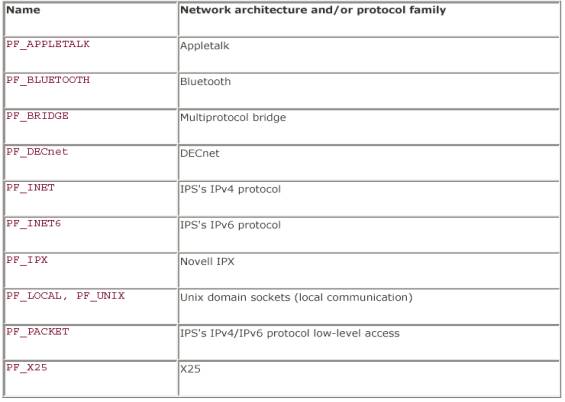
一个套接口可以看作是进程间通信的端点（endpoint），每个套接口的名字都是唯一的（唯一的含义是不言而喻的），其他进程可以发现、连接并且与之通信。通信域用来说明套接口通信的协议，不同的通信域有不同的通信协议以及套接口的地址结构等等，因此，创建一个套接口时，要指明它的通信域。比较常见的是unix域套接口（采用套接口机制实现单机内的进程间通信）及网际通信域。

**1、背景知识**

linux目前的网络内核代码主要基于伯克利的BSD的unix实现，整个结构采用的是一种面向对象的分层机制。层与层之间有严格的接口定义。这里我们引用[1]中的一个图表来描述linux支持的一些通信协议：



我们这里只关心IPS，即因特网协议族，也就是通常所说的TCP/IP网络。我们这里假设读者具有网络方面的一些背景知识，如了解网络的分层结构，通常所说的7层结构；了解IP地址以及路由的一些基本知识。

目前linux网络API是基于BSD套接口的（系统V提供基于流I/O子系统的用户接口，但是linux内核目前不支持流I/O子系统）。套接口可以说是网络编程中一个非常重要的概念，linux以文件的形式实现套接口，与套接口相应的文件属于sockfs特殊文件系统，创建一个套接口就是在sockfs中创建一个特殊文件，并建立起为实现套接口功能的相关数据结构。换句话说，对每一个新创建的BSD套接口，linux内核都将在sockfs特殊文件系统中创建一个新的inode。描述套接口的数据结构是socket，将在后面给出。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-ipc/part6/#ibm-pcon)

**2、重要数据结构**

下面是在网络编程中比较重要的几个数据结构，读者可以在后面介绍编程API部分再回过头来了解它们。

**（1）表示套接口的数据结构struct socket**

套接口是由socket数据结构代表的，形式如下：

|  |
| --- |
| struct socket  {  socket\_state state; /\* 指明套接口的连接状态，一个套接口的连接状态可以有以下几种  套接口是空闲的，还没有进行相应的端口及地址的绑定；还没有连接；正在连接中；已经连接；正在解除连接。 \*/  unsigned long flags;  struct proto\_ops ops; /\* 指明可对套接口进行的各种操作 \*/  struct inode inode; /\* 指向sockfs文件系统中的相应inode \*/  struct fasync\_struct \*fasync\_list; /\* Asynchronous wake up list \*/  struct file \*file; /\* 指向sockfs文件系统中的相应文件 \*/  struct sock sk; /\* 任何协议族都有其特定的套接口特性，该域就指向特定协议族的套接口对  象。 \*/  wait\_queue\_head\_t wait;  short type;  unsigned char passcred;  }; |

**（2）描述套接口通用地址的数据结构struct sockaddr**

由于历史的缘故，在bind、connect等系统调用中，特定于协议的套接口地址结构指针都要强制转换成该通用的套接口地址结构指针。结构形式如下：

|  |
| --- |
| struct sockaddr {  sa\_family\_t sa\_family; /\* address family, AF\_xxx \*/  char sa\_data[14]; /\* 14 bytes of protocol address \*/  }; |

**（3）描述因特网地址结构的数据结构struct sockaddr\_in（这里局限于IP4）：**

|  |
| --- |
| struct sockaddr\_in  {  \_\_SOCKADDR\_COMMON (sin\_); /\* 描述协议族 \*/  in\_port\_t sin\_port; /\* 端口号 \*/  struct in\_addr sin\_addr; /\* 因特网地址 \*/  /\* Pad to size of `struct sockaddr'. \*/  unsigned char sin\_zero[sizeof (struct sockaddr) -  \_\_SOCKADDR\_COMMON\_SIZE -  sizeof (in\_port\_t) -  sizeof (struct in\_addr)];  }; |

一般来说，读者最关心的是前三个域，即通信协议、端口号及地址。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-ipc/part6/#ibm-pcon)

**3、套接口编程的几个重要步骤：**

**（1）创建套接口，由系统调用socket实现：**

|  |
| --- |
| int socket( int domain, int type, int ptotocol); |

参数domain指明通信域，如PF\_UNIX（unix域），PF\_INET（IPv4），PF\_INET6(IPv6)等；type指明通信类型，如SOCK\_STREAM(面向连接方式)、SOCK\_DGRAM(非面向连接方式)等。一般来说，参数protocol可设置为0，除非用在原始套接口上（原始套接口有一些特殊功能，后面还将介绍）。

**注：**socket（）系统调用为套接口在sockfs文件系统中分配一个新的文件和dentry对象，并通过文件描述符把它们与调用进程联系起来。进程可以像访问一个已经打开的文件一样访问套接口在sockfs中的对应文件。但进程绝不能调用open()来访问该文件（sockfs文件系统没有可视安装点，其中的文件永远不会出现在系统目录树上），当套接口被关闭时，内核会自动删除sockfs中的inodes。

**（2）绑定地址**

根据传输层协议（TCP、UDP）的不同，客户机及服务器的处理方式也有很大不同。但是，不管通信双方使用何种传输协议，都需要一种标识自己的机制。

通信双方一般由两个方面标识：地址和端口号（通常，一个IP地址和一个端口号常常被称为一个套接口）。根据地址可以寻址到主机，根据端口号则可以寻址到主机提供特定服务的进程，实际上，一个特定的端口号代表了一个提供特定服务的进程。

对于使用TCP传输协议通信方式来说，通信双方需要给自己绑定一个唯一标识自己的套接口，以便建立连接；对于使用UDP传输协议，只需要服务器绑定一个标识自己的套接口就可以了，用户则不需要绑定(在需要时，如调用connect时[注1]，内核会自动分配一个本地地址和本地端口号)。绑定操作由系统调用bind()完成：

|  |
| --- |
| int bind( int sockfd, const struct sockaddr \* my\_addr, socklen\_t my\_addr\_len) |

第二个参数对于Ipv4来说，实际上需要填充的结构是struct sockaddr\_in，前面已经介绍了该结构。这里只想强调该结构的第一个域，它表明该套接口使用的通信协议，如AF\_INET。联系socket系统调用的第一个参数，读者可能会想到PF\_INET与AF\_INET究竟有什么不同？实际上，原来的想法是每个通信域（如PF\_INET）可能对应多个协议（如AF\_INET），而事实上支持多个协议的通信域一直没有实现。因此，在linux内核中，AF\_\*\*\*与PF\_\*\*\*被定义为同一个常数，因此，在编程时可以不加区分地使用他们。

**注1：**在采用非面向连接通信方式时，也会用到connect()调用，不过与在面向连接中的connect（）调用有本质的区别：在非面向连接通信中，connect调用只是先设置一下对方的地址，内核为本地套接口记下对方的地址，然后采用send()来发送数据，这样避免每次发送时都要提供相同的目的地址。其中的connect()调用不涉及握手过程；而在面向连接的通信方式中，connect()要完成一个严格的握手过程。

**（3）请求建立连接（由TCP客户发起）**

对于采用面向连接的传输协议TCP实现通信来说，一个比较重要的步骤就是通信双方建立连接（如果采用udp传输协议则不需要），由系统调用connect()完成：

|  |
| --- |
| int connect( int sockfd, const struct sockaddr \* servaddr, socklen\_t addrlen) |

第一个参数为本地调用socket后返回的描述符，第二个参数为服务器的地址结构指针。connect（）向指定的套接口请求建立连接。

**注：**与connect（）相对应，在服务器端，通过系统调用listen()，指定服务器端的套接口为监听套接口，监听每一个向服务器套接口发出的连接请求，并通过握手机制建立连接。内核为listen()维护两个队列：已完成连接队列和未完成连接队列。

**（4）接受连接请求（由TCP服务器端发起）**

服务器端通过监听套接口，为所有连接请求建立了两个队列：已完成连接队列和未完成连接队列（每个监听套接口都对应这样两个队列，当然，一般服务器只有一个监听套接口）。通过accept()调用，服务器将在监听套接口的已连接队列头中，返回用于代表当前连接的套接口描述字。

|  |
| --- |
| int accept( int sockfd, struct sockaddr \* cliaddr, socklen\_t \* addrlen) |

第一个参数指明哪个监听套接口，一般是由listen（）系统调用指定的（由于每个监听套接口都对应已连接和未连接两个队列，因此它的内部机制实质是通过sockfd指定在哪个已连接队列头中返回一个用于当前客户的连接，如果相应的已连接队列为空，accept进入睡眠）。第二个参数指明客户的地址结构，如果对客户的身份不感兴趣，可指定其为空。

**注：**对于采用TCP传输协议进行通信的服务器和客户机来说，一定要经过客户请求建立连接，服务器接受连接请求这一过程；而对采用UDP传输协议的通信双方则不需要这一步骤。

**（5）通信**

客户机可以通过套接口接收服务器传过来的数据，也可以通过套接口向服务器发送数据。前面所有的准备工作（创建套接口、绑定等操作）都是为这一步骤准备的。

常用的从套接口中接收数据的调用有：recv、recvfrom、recvmsg等，常用的向套接口中发送数据的调用有send、sendto、sendmsg等。

|  |
| --- |
| int recv(int s, void \*  *buf*, size\_t  *len*, int  *flags*)  int recvfrom(int s, void \*  *buf*, size\_t  *len*, int  *flags*, struct sockaddr \*  *from*, socklen\_t \*  *fromlen*)  int recvmsg(int s, struct msghdr \*  *msg*, int  *flags*)  int send(int s,const void \*  *msg*, size\_t  *len*, int  *flags*)  int sendto(int s, const void \*  *msg*, size\_t  *len*, int  *flags* const struct sockaddr \*  *to*, socklen\_t  *tolen*)  int sendmsg(int s, const struct msghdr \*  *msg*, int  *flags*) |

这里不再对这些调用作具体的说明，只想强调一下，recvfrom（）以及recvmsg（）可用于面向连接的套接口，也可用于面向非连接的套接口；而recv（）一般用于面向连接的套接口。另外，在调用了connect()之后，就应给调用send()而不是sendto()了，因为调用了connect之后，目标就已经确定了。

前面讲到，socket（）系统调用返回套接口描述字，实际上它是一个文件描述符。所以，可以对套接口进行通常的读写操作，即使用read()及write()方法。在实际应用中，由于面向连接的通信（采用TCP传输协议）是可靠的，同时又保证字节流原有的顺序，所以更适合用read及write方法。而非面向连接的通信（采用UDP传输协议）是不可靠的，字节流也不一定保持原有的顺序，所以一般不宜用read及write方法。

**（6）通信的最后一步是关闭套接口**

由close()来完成此项功能，它唯一的参数是套接口描述字，不再赘述。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-ipc/part6/#ibm-pcon)

**4、典型调用代码：**

到处可以发现基于套接口的客户机及服务器程序，这里不再给出完整的范例代码，只是给出它们的典型调用代码，并给出简要说明。

**（1）典型的TCP服务器代码：**

|  |
| --- |
| ... ...  int listen\_fd, connect\_fd;  struct sockaddr\_in serv\_addr, client\_addr;  ... ...  listen\_fd = socket ( PF\_INET, SOCK\_STREAM, 0 );  /\* 创建网际Ipv4域的（由PF\_INET指定）面向连接的（由SOCK\_STREAM指定，  如果创建非面向连接的套接口则指定为SOCK\_DGRAM）  的套接口。第三个参数0表示由内核确定缺省的传输协议，  对于本例，由于创建的是可靠的面向连接的基于流的套接口，  内核将选择TCP作为本套接口的传输协议） \*/  bzero( &serv\_addr, sizeof(serv\_addr) );  serv\_addr.sin\_family = AF\_INET ; /\* 指明通信协议族 \*/  serv\_addr.sin\_port = htons( 49152 ) ; /\* 分配端口号 \*/  inet\_pton(AF\_INET, " 192.168.0.11", &serv\_addr.sin\_sddr) ;  /\* 分配地址，把点分十进制IPv4地址转化为32位二进制Ipv4地址。 \*/  bind( listen\_fd, (struct sockaddr\*) serv\_addr, sizeof ( struct sockaddr\_in )) ;  /\* 实现绑定操作 \*/  listen( listen\_fd, max\_num) ;  /\* 套接口进入侦听状态，max\_num规定了内核为此套接口排队的最大连接个数 \*/  for( ; ; ) {  ... ...  connect\_fd = accept( listen\_fd, (struct sockaddr\*)client\_addr, &len ) ; /\* 获得连接fd. \*/  ... ... /\* 发送和接收数据 \*/  } |

**注：**端口号的分配是有一些惯例的，不同的端口号对应不同的服务或进程。比如一般都把端口号21分配给FTP服务器的TCP/IP实现。端口号一般分为3段，0-1023（受限的众所周知的端口，由分配数值的权威机构IANA管理），1024-49151（可以从IANA那里申请注册的端口），49152-65535（临时端口，这就是为什么代码中的端口号为49152）。

对于多字节整数在内存中有两种存储方式：一种是低字节在前，高字节在后，这样的存储顺序被称为低端字节序（little-endian）；高字节在前，低字节在后的存储顺序则被称为高端字节序（big-endian）。网络协议在处理多字节整数时，采用的是高端字节序，而不同的主机可能采用不同的字节序。因此在编程时一定要考虑主机字节序与网络字节序间的相互转换。这就是程序中使用htons函数的原因，它返回网络字节序的整数。

**（2）典型的TCP客户代码：**

|  |
| --- |
| ... ...  int socket\_fd;  struct sockaddr\_in serv\_addr ;  ... ...  socket\_fd = socket ( PF\_INET, SOCK\_STREAM, 0 );  bzero( &serv\_addr, sizeof(serv\_addr) );  serv\_addr.sin\_family = AF\_INET ; /\* 指明通信协议族 \*/  serv\_addr.sin\_port = htons( 49152 ) ; /\* 分配端口号 \*/  inet\_pton(AF\_INET, " 192.168.0.11", &serv\_addr.sin\_sddr) ;  /\* 分配地址，把点分十进制IPv4地址转化为32位二进制Ipv4地址。 \*/  connect( socket\_fd, (struct sockaddr\*)serv\_addr, sizeof( serv\_addr ) ) ; /\* 向服务器发起连接请求 \*/  ... ... /\* 发送和接收数据 \*/  ... ... |

对比两段代码可以看出，许多调用是服务器或客户机所特有的。另外，对于非面向连接的传输协议，代码还有简单些，没有连接的发起请求和接收请求部分。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-ipc/part6/#ibm-pcon)

**5、网络编程中的其他重要概念**

下面列出了网络编程中的其他重要概念，基本上都是给出这些概念能够实现的功能，读者在编程过程中如果需要这些功能，可查阅相关概念。

**（1）、I/O复用的概念**

I/O复用提供一种能力，这种能力使得当一个I/O条件满足时，进程能够及时得到这个信息。I/O复用一般应用在进程需要处理多个描述字的场合。它的一个优势在于，进程不是阻塞在真正的I/O调用上，而是阻塞在select()调用上，select()可以同时处理多个描述字，如果它所处理的所有描述字的I/O都没有处于准备好的状态，那么将阻塞；如果有一个或多个描述字I/O处于准备好状态，则select()不阻塞，同时会根据准备好的特定描述字采取相应的I/O操作。

**（2）、Unix通信域**

前面主要介绍的是PF\_INET通信域，实现网际间的进程间通信。基于Unix通信域（调用socket时指定通信域为PF\_LOCAL即可）的套接口可以实现单机之间的进程间通信。采用Unix通信域套接口有几个好处：Unix通信域套接口通常是TCP套接口速度的两倍；另一个好处是，通过Unix通信域套接口可以实现在进程间传递描述字。所有可用描述字描述的对象，如文件、管道、有名管道及套接口等，在我们以某种方式得到该对象的描述字后，都可以通过基于Unix域的套接口来实现对描述字的传递。接收进程收到的描述字值不一定与发送进程传递的值一致（描述字是特定于进程的），但是特们指向内核文件表中相同的项。

**（3）、原始套接口**

原始套接口提供一般套接口所不提供的功能：

* 原始套接口可以读写一些用于控制的控制协议分组，如ICMPv4等，进而可实现一些特殊功能。
* 原始套接口可以读写特殊的IPv4数据包。内核一般只处理几个特定协议字段的数据包，那么一些需要不同协议字段的数据包就需要通过原始套接口对其进行读写；
* 通过原始套接口可以构造自己的Ipv4头部，也是比较有意思的一点。

创建原始套接口需要root权限。

**（4）、对数据链路层的访问**

对数据链路层的访问，使得用户可以侦听本地电缆上的所有分组，而不需要使用任何特殊的硬件设备，在linux下读取数据链路层分组需要创建SOCK\_PACKET类型的套接口，并需要有root权限。

**（5）、带外数据（out-of-band data）**

如果有一些重要信息要立刻通过套接口发送（不经过排队），请查阅与带外数据相关的文献。

**（6）、多播**

linux内核支持多播，但是在默认状态下，多数linux系统都关闭了对多播的支持。因此，为了实现多播，可能需要重新配置并编译内核。具体请参考[4]及[2]。

**结论：**linux套接口编程的内容可以说是极大丰富，同时它涉及到许多的网络背景知识，有兴趣的读者可在[2]中找到比较系统而全面的介绍。

至此，本专题系列（linux环境进程间通信）全部结束了。实际上，进程间通信的一般意义通常指的是消息队列、信号灯和共享内存，可以是posix的，也可以是SYS v的。本系列同时介绍了管道、有名管道、信号以及套接口等，是更为一般意义上的进程间通信机制。

**参考资料**

* Understanding the Linux Kernel, 2nd Edition, By Daniel P. Bovet, Marco Cesati , 对各主题阐述得重点突出，脉络清晰。网络部分分析集中在TCP/IP协议栈的数据连路层、网络层以及传输层。
* UNIX网络编程第一卷：套接口API和X/Open传输接口API，作者：W.Richard Stevens，译者：杨继张，清华大学出版社。不仅对套接口网络编程有极好的描述，而且极为详尽的阐述了相关的网络背景知识。不论是入门还是深入研究，都是不可多得的好资料。
* Linux内核源代码情景分析（下），毛德操、胡希明著，浙江大学出版社，给出了unix域套接口部分的内核代码分析。
* GNU/Linux编程指南，入门、应用、精通，第二版，Kurt Wall等著，张辉译