UDP

UDP 是User Datagram Protocol的简称， 中文名是用户数据报协议，是[OSI](https://baike.baidu.com/item/OSI)（Open System Interconnection，[开放式系统互联](https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E5%BC%8F%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%81%94)） 参考模型中一种无连接的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)协议，提供面向事务的简单不可靠信息传送服务，IETF RFC 768是UDP的正式规范。UDP在IP报文的协议号是17。

UDP协议全称是用户数据报协议[1]  ，在[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C)中它与[TCP](https://baike.baidu.com/item/TCP)协议一样用于处理数据包，是一种无连接的协议。在[OSI](https://baike.baidu.com/item/OSI)模型中，在第四层——[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82)，处于IP协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。UDP用来支持那些需要在[计算机](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA)之间传输数据的网络应用。包括[网络视频会议](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E8%A7%86%E9%A2%91%E4%BC%9A%E8%AE%AE)系统在内的众多的客户/服务器模式的网络应用都需要使用UDP协议。UDP协议从问世至今已经被使用了很多年，虽然其最初的光彩已经被一些类似协议所掩盖，但是即使是在今天UDP仍然不失为一项非常实用和可行的网络传输层协议。

与所熟知的TCP（[传输控制协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE)）协议一样，UDP协议直接位于IP（网际协议）协议的顶层。根据OSI（[开放系统互连](https://baike.baidu.com/item/%E5%BC%80%E6%94%BE%E7%B3%BB%E7%BB%9F%E4%BA%92%E8%BF%9E)）参考模型，UDP和TCP都属于传输层协议。UDP协议的主要作用是将[网络数据](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E6%95%B0%E6%8D%AE)流量压缩成数据包的形式。一个典型的数据包就是一个二进制数据的传输单位。每一个数据包的前8个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

协议

UDP是[OSI](https://baike.baidu.com/item/OSI)参考模型中一种无连接的传输层协议，它主要用于不要求分组顺序到达的传输中，分组传输顺序的检查与排序由应用层完成[2]  ，提供面向事务的简单不可靠信息传送服务。UDP 协议基本上是[IP协议](https://baike.baidu.com/item/IP%E5%8D%8F%E8%AE%AE)与上层协议的接口。UDP协议适用[端口](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3)分别运行在同一台设备上的多个[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)。

UDP提供了无连接通信，且不对传送数据包进行可靠性保证，适合于一次传输少量数据，UDP传输的可靠性由应用层负责。常用的UDP端口号有：

应用协议 端口号

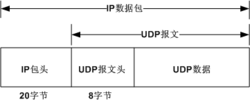
DNS 53

TFTP 69

SNMP 161

UDP报文没有可靠性保证、顺序保证和流量控制字段等，可靠性较差。但是正因为UDP协议的控制选项较少，在数据传输过程中延迟小、数据传输效率高，适合对可靠性要求不高的应用程序，或者可以保障可靠性的应用程序，如DNS、TFTP、SNMP等。

UDP在IP报文中的位置如图所示。

[](https://baike.baidu.com/pic/UDP/571511/0/4d086e061d950a7b8028e76b09d162d9f2d3c904?fr=lemma&ct=single)

使用

在选择使用协议的时候，选择UDP必须要谨慎。在[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C)质量令人十分不满意的环境下，UDP协议数据包丢失会比较严重。但是由于UDP的特性：它不属于连接型协议，因而具有资源消耗小，处理速度快的优点，所以通常音频、视频和普通数据在传送时使用UDP较多，因为它们即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。比如我们聊天用的ICQ和[QQ](https://baike.baidu.com/item/QQ)就是使用的UDP协议。

报头

UDP报头由4个域组成，其中每个域各占用2个字节，具体如下：

[](https://baike.baidu.com/pic/UDP/571511/0/b2de9c82d158ccbf4049fb531bd8bc3eb03541e5?fr=lemma&ct=single)UDP

源[端口号](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3%E5%8F%B7)

目标端口号

[数据报](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%8A%A5)长度

校验值

UDP协议使用端口号为不同的应用保留其各自的数据传输通道。UDP和[TCP协议](https://baike.baidu.com/item/TCP%E5%8D%8F%E8%AE%AE)正是采用这一机制实现对同一时刻内多项应用同时发送和接收数据的支持。数据发送一方（可以是客户端或服务器端）将UDP数据包通过源端口发送出去，而数据接收一方则通过目标端口接收数据。有的网络应用只能使用预先为其预留或注册的静态端口；而另外一些网络应用则可以使用未被注册的动态端口。因为UDP报头使用两个字节存放端口号，所以端口号的有效范围是从0到65535。一般来说，大于49151的端口号都代表动态端口。

数据报的长度是指包括报头和数据部分在内的总字节数。因为报头的长度是固定的，所以该域主要被用来计算可变长度的数据部分（又称为数据负载）。数据报的最大长度根据操作环境的不同而各异。从理论上说，包含报头在内的数据报的最大长度为65535字节。不过，一些实际应用往往会限制数据报的大小，有时会降低到8192字节。

UDP协议使用报头中的校验值来保证数据的安全。校验值首先在数据发送方通过特殊的算法计算得出，在传递到接收方之后，还需要再重新计算。如果某个数据报在传输过程中被第三方篡改或者由于线路噪音等原因受到损坏，发送和接收方的校验计算值将不会相符，由此UDP协议可以检测是否出错。这与TCP协议是不同的，后者要求必须具有校验值。

许多链路层协议都提供错误检查，包括流行的[以太网](https://baike.baidu.com/item/%E4%BB%A5%E5%A4%AA%E7%BD%91)协议，也许你想知道为什么UDP也要提供检查和校验。其原因是链路层以下的协议在源端和[终端](https://baike.baidu.com/item/%E7%BB%88%E7%AB%AF)之间的某些通道可能不提供错误检测。虽然UDP提供有错误检测，但检测到错误时，UDP不做错误校正，只是简单地把损坏的消息段扔掉，或者给应用程序提供警告信息。

UDP Helper是实现对指定UDP端口广播报文的中继转发，即将指定UDP端口的广播报文转换为单播报文发送给指定的服务器，起到中继的作用.

功能

为了在给定的主机上能识别多个目的地址，同时允许多个应用程序在同一台主机上工作并能独立地进行数据包的发送和接收，设计[用户数据报协议](https://baike.baidu.com/item/%E7%94%A8%E6%88%B7%E6%95%B0%E6%8D%AE%E6%8A%A5%E5%8D%8F%E8%AE%AE)UDP。

使用UDP协议包括：[TFTP](https://baike.baidu.com/item/TFTP)、[SNMP](https://baike.baidu.com/item/SNMP)、NFS、DNS、BOOTP  
　　UDP使用底层的互联网协议来传送报文，同IP一样提供不可靠的无连接数据包传输服务。它不提供报文到达确认、排序、及流量控制等功能。

报文格式

每个UDP报文分UDP报头和UDP数据区两部分。报头由四个16位长（2字节）字段组成，分别说明该报文的源端口、目的端口、报文长度以及校验值。

分层封装

在TCP/IP协议层次模型中，UDP位于IP层之上。应用程序访问UDP层然后使用IP层传送数据报。IP层的报头指明了源主机和目的主机地址，而UDP层的报头指明了主机上的源端口和目的端口。

分解操作

**UDP的复用、分解与端口**

UDP软件应用程序之间的复用与分解都要通过端口机制来实现。每个应用程序在发送数据报之前必须与操作系统协商以获得协议端口和相应的端口号。  
　　UDP分解操作：从IP层接收了数据报之后，根据UDP的目的端口号进行分解操作。  
　　UDP端口号指定有两种方式：由管理机构指定端口和动态绑定的方式。

配置命令

**1.1.1 display udp-helper server**

【命令】

**display udp-helper server**[**interface***interface-type interface-number*]

【视图】

任意视图

【缺省级别】

2：系统级

【参数】

**interface***interface-type interface-number*：显示指定接口的UDP中继转发相关信息。*interface-type interface-number*为接口类型和接口编号。

【描述】

**display udp-helper server**命令用来显示UDP中继转发的相关信息。

如果不选择参数，则显示所有接口的UDP中继转发相关信息。

【举例】

# 显示VLAN接口1的UDP中继转发相关信息。

<Sysname> display udp-helper server interface vlan-interface 1

Interface name Server address Packets sent

Vlan-interface1 192.1.1.2 0

以上显示信息表示VLAN接口1对应的目的服务器的IP地址为192.1.1.2，中继转发到该目的服务器的报文数为0。

**1.1.2 reset udp-helper packet**

【命令】

**reset udp-helper packet**

【视图】

用户视图

【缺省级别】

2：系统级

【参数】

无

【描述】

**reset udp-helper packet**命令用来清除UDP中继转发的报文统计数目。

【举例】

# 清除UDP中继转发的报文统计数目。

<Sysname> reset udp-helper packet

**1.1.3 udp-helper enable**

【命令】

**udp-helper enable**

**undo udp-helper enable**

【视图】

系统视图

【缺省级别】

2：系统级

【参数】

无

【描述】

**udp-helper enable**命令用来使能UDP Helper功能，设备会将指定UDP端口的广播报文转换为单播报文发送给指定的目的服务器，起到中继的作用。**undo udp-helper enable**命令用来关闭UDP Helper功能。

缺省情况下，UDP Helper功能处于关闭状态。

【举例】

# 使能UDP Helper功能。

<Sysname> system-view

[Sysname] udp-helper enable

**1.1.4 udp-helper port**

【命令】

**udp-helper port**{*port-number*|**dns**|**netbios-ds**|**netbios-ns**|**tacacs**|**tftp**|**time**}

**undo udp-helper port**{*port-number*|**dns**|**netbios-ds**|**netbios-ns**|**tacacs**|**tftp**|**time**}

【视图】

系统视图

【缺省级别】

2：系统级

【参数】

*port-number*：需要中继转发的UDP端口号，取值范围为1～65535（不支持67和68）。

**dns**：对DNS的数据报文进行中继转发，对应的UDP端口号为53。

**netbios-ds**：对NetBIOS数据报服务的数据报文进行中继转发，对应的UDP端口号为138。

**netbios-ns**：对NetBIOS名字服务的数据报文进行中继转发，对应的UDP端口号为137。

**tacacs**：对终端访问控制器访问控制系统的数据报文进行中继转发，对应的UDP端口号为49。

**tftp**：对简单文件传输协议的数据报文进行中继转发，对应的UDP端口号为69。

**time**：对时间服务的数据报文进行中继转发，对应的UDP端口号为37。

【描述】

**udp-helper port**命令用来配置需要中继转发的UDP端口。**undo udp-helper port**命令用来取消对需要中继转发的UDP端口的配置。

缺省情况下，没有配置中继转发的UDP端口。

关闭UDP Helper功能后，所有已配置的UDP端口都被取消。

【举例】

# 配置对目的UDP端口号为100的广播报文进行中继转发。

<Sysname> system-view

[Sysname] udp-helper port 100

**1.1.5 udp-helper server**

【命令】

**udp-helper server***ip-address*

**undo udp-helper server**[*ip-address*]

【视图】

三层以太网接口视图/VLAN接口视图

【缺省级别】

2：系统级

【参数】

*ip-address*：目的服务器的IP地址，为点分十进制形式。

【描述】

**udp-helper server**命令用来配置中继转发的目的服务器。**undo udp-helper server**命令用来删除中继转发的目的服务器。

缺省情况下，没有配置中继转发的目的服务器。

目前，一个接口上最多可以配置20个中继转发的目的服务器。

需要注意的是，**undo udp-helper server**命令后不加参数时，将会删除该接口下配置的所有目的服务器。

相关配置可参考命令**display udp-helper server**。[3]

特性

（1） UDP是一个[无连接协议](https://baike.baidu.com/item/%E6%97%A0%E8%BF%9E%E6%8E%A5%E5%8D%8F%E8%AE%AE)，传输数据之前源端和终端不建立连接，当

[](https://baike.baidu.com/pic/UDP/571511/0/d019d2bf97921a4a19d81ff1?fr=lemma&ct=single)UDP

它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

（2） 由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。

（3） UDP信息包的标题很短，只有8个字节，相对于TCP的20个字节信息包的额外开销很小。

（4） 吞吐量不受拥挤控制算法的调节，只受[应用软件](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E8%BD%AF%E4%BB%B6)生成数据的速率、传输带宽、源端和终端主机性能的限制。

（5）UDP使用**尽最大努力交付，**即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。

（6）UDP是**面向报文**的。发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付给IP层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。

虽然UDP是一个不可靠的协议，但它是分发信息的一个理想协议。例如，在屏幕上报告股票市场、在屏幕上显示航空信息等等。UDP也用在[路由信息协议](https://baike.baidu.com/item/%E8%B7%AF%E7%94%B1%E4%BF%A1%E6%81%AF%E5%8D%8F%E8%AE%AE)RIP（Routing Information Protocol）中修改路由表。在这些应用场合下，如果有一个消息丢失，在几秒之后另一个新的消息就会替换它。UDP广泛用在多媒体应用中，例如，Progressive Networks公司开发的RealAudio[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6)，它是在[因特网](https://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91)上把预先录制的或者现场音乐实时传送给客户机的一种软件，该软件使用的RealAudio audio-on-demand protocol协议就是运行在UDP之上的协议，大多数因特网电话软件产品也都运行在UDP之上。

对比

UDP和TCP协议的主要区别是两者在如何实现信息的可靠传递方面不同。

[](https://baike.baidu.com/pic/UDP/571511/0/d4239b353cc87d5190ef39f3?fr=lemma&ct=single)UDP

TCP协议中包含了专门的传递保证机制，当数据接收方收到发送方传来的信息时，会自动向发送方发出确认消息；发送方只有在接收到该确认消息之后才继续传送其它信息，否则将一直等待直到收到确认信息为止。与TCP不同，UDP协议并不提供数据传送的保证机制。如果在从发送方到接收方的传递过程中出现数据报的丢失，协议本身并不能做出任何检测或提示。因此，通常人们把UDP协议称为**不可靠的**[传输协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE)。

相对于TCP协议，UDP协议的另外一个不同之处在于如何接收突发性的多个数据报。不同于TCP，UDP并不能确保数据的发送和接收顺序。例如，一个位于客户端的应用程序向服务器发出了以下4个数据报

D1

D22

D333

D4444

但是UDP有可能按照以下顺序将所接收的数据提交到服务端的应用：

D333

D1

D4444

D22

事实上，UDP协议的这种乱序性基本上很少出现，通常只会在网络非常拥挤的情况下才有可能发生。

应用

既然UDP是一种不可靠的[网络协议](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%8D%8F%E8%AE%AE)，那么还有什么使用价值或必要呢？其实不然，在有些情况下UDP协议可能会变得非常有用。因为UDP具有TCP所望尘莫及的速度优势。虽然TCP协议中植入了各种安全保障功能，但是在实际执行的过程中会占用大量的[系统开销](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E5%BC%80%E9%94%80)，无疑使速度受到严重的影响。反观UDP由于排除了信息可靠传递机制，将安全和排序等功能移交给上层应用来完成，极大降低了执行时间，使速度得到了保证。

关于UDP协议的最早规范是[RFC](https://baike.baidu.com/item/RFC)768，1980年发布。尽管时间已经很长，但是UDP协议仍然继续在主流应用中发挥着作用。包括视频[电话会议系统](https://baike.baidu.com/item/%E7%94%B5%E8%AF%9D%E4%BC%9A%E8%AE%AE%E7%B3%BB%E7%BB%9F)在内的许多应用都证明了UDP协议的存在价值。因为相对于可靠性来说，这些应用更加注重实际性能，所以为了获得更好的使用效果（例如，更高的画面帧刷新速率）往往可以牺牲一定的可靠性（例如，画面质量）。这就是UDP和TCP两种协议的权衡之处。根据不同的环境和特点，两种传输协议都将在今后的网络世界中发挥更加重要的作用。

程序设计

UDP Server程序

编写程序

(1)使用socket()来建立一个UDP socket，第二个参数为SOCK\_DGRAM。

(2)初始化sockaddr\_in结构的变量，并赋值。sockaddr\_in结构定义：

struct sockaddr\_in {

uint8\_t sin\_len ;

sa\_family\_t sin\_family;

in\_port\_t sin\_port ;

struct[in\_addr](https://baike.baidu.com/item/in_addr)sin\_addr;

char sin\_zero[8];

};

这里使用“8888”作为服务程序的端口，使用“[INADDR\_ANY](https://baike.baidu.com/item/INADDR_ANY)”作为绑定的IP地址即任何主机上的地址。

(3)使用[bind()](https://baike.baidu.com/item/bind%28%29)把上面的[socket](https://baike.baidu.com/item/socket)和定义的IP地址和端口绑定。这里检查bind()是否执行成功，如果有错误就退出。这样可以防止服务程序重复运行的问题。

(4)进入无限循环程序，使用recvfrom()进入等待状态，直到接收到客户程序发送的数据，就处理收到的数据，并向客户程序发送反馈。这里是直接把收到的数据发回给客户程序。

程序内容

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include<sys/socket.h>  #include<unistd.h>  #include<string.h>  #include<stdio.h>  #include<arpa/inet.h>  #include<stdlib.h>  #defineMAXLINE80  #defineSERV\_PORT8888    voiddo\_echo(intsockfd,structsockaddr\*pcliaddr,socklen\_tclilen)  {  intn;  socklen\_tlen;  charmesg[MAXLINE];  for(;;)  {  len=clilen;  /\*waitingforreceivedata\*/  n=recvfrom(sockfd,mesg,MAXLINE,0,pcliaddr,&len);  /\*sentdatabacktoclient\*/  sendto(sockfd,mesg,n,0,pcliaddr,len);  }  }  intmain(intargc,char\*argv[])  {  intsockfd;  structsockaddr\_inservaddr,cliaddr;  sockfd=socket(AF\_INET,SOCK\_DGRAM,0);/\*createasocket\*/  /\*initservaddr\*/  bzero(&servaddr,sizeof(servaddr));  servaddr.sin\_family=AF\_INET;  servaddr.sin\_addr.s\_addr=htonl(INADDR\_ANY);  servaddr.sin\_port=htons(SERV\_PORT);  /\*bindaddressandporttosocket\*/  if(bind(sockfd,(structsockaddr\*)&servaddr,sizeof(servaddr))==-1)  {  perror("binderror");  exit(1);  }  do\_echo(sockfd,(structsockaddr\*)&cliaddr,sizeof(cliaddr));  return0;  } |

UDP Client程序

**1、编写UDP Client程序的步骤**

(1)初始化sockaddr\_in结构的变量，并赋值。这里使用“8888”作为连接的服务程序的端口，从命令行参数读取IP地址，并且判断IP地址是否符合要求。

(2)使用socket()来建立一个UDP socket，第二个参数为SOCK\_DGRAM。

(3)使用connect()来建立与服务程序的连接。与TCP协议不同，UDP的connect()并没有与服务程序三次握手。上面说了UDP是非连接的，实际上也可以是连接的。使用连接的UDP，kernel可以直接返回错误信息给用户程序，从而避免由于没有接收到数据而导致调用recvfrom()一直等待下去，看上去好像客户程序没有反应一样。

(4)向服务程序发送数据，因为使用连接的UDP，所以使用write()来替代sendto()。这里的数据直接从标准输入读取用户输入。

(5)接收服务程序发回的数据，同样使用read()来替代recvfrom()。

(6)处理接收到的数据，这里是直接输出到标准输出上。

**udpclient.c程序内容**：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | #include<sys/socket.h>  #include<unistd.h>  #include<string.h>  #include<stdio.h>  #include<arpa/inet.h>  #include<stdlib.h>  #defineMAXLINE80  #defineSERV\_PORT8888    voiddo\_cli(FILE\*fp,intsockfd,structsockaddr\*pservaddr,socklen\_tservlen)  {  intn;  charsendline[MAXLINE],recvline[MAXLINE+1];  /\*connecttoserver\*/  if(connect(sockfd,(structsockaddr\*)pservaddr,servlen)==-1)  {  perror("connecterror");  exit(1);  }  while(fgets(sendline,MAXLINE,fp)!=NULL)  {  /\*readalineandsendtoserver\*/  write(sockfd,sendline,strlen(sendline));  /\*receivedatafromserver\*/  n=read(sockfd,recvline,MAXLINE);  if(n==-1)  {  perror("readerror");  exit(1);  }  recvline[n]=0;/\*terminatestring\*/  fputs(recvline,stdout);  }  }  intmain(intargc,char\*\*argv)  {  intsockfd;  structsockaddr\_inservaddr;  /\*checkargs\*/  if(argc!=2)  {  printf("usage:udpclient\n");  exit(1);  }  /\*initservaddr\*/  bzero(&servaddr,sizeof(servaddr));  servaddr.sin\_family=AF\_INET;  servaddr.sin\_port=htons(SERV\_PORT);  if(inet\_pton(AF\_INET,argv[1],&servaddr.sin\_addr)<=0)  {  printf("[%s]isnotavalidIPaddress\n",argv[1]);  exit(1);  }  sockfd=socket(AF\_INET,SOCK\_DGRAM,0);  do\_cli(stdin,sockfd,(structsockaddr\*)&servaddr,sizeof(servaddr));  return0;  } |

例子程序

**1、编译例子程序**

使用如下命令来编译例子程序：

gcc -Wall -o udpserv udpserv.c

gcc -Wall -o udpclient udpclient.c

编译完成生成了udpserv和udpclient两个可执行程序。

**2、运行UDP Server程序**

执行./udpserv &命令来启动服务程序。我们可以使用netstat -ln命令来观察服务程序绑定的IP地址和端口，部分输出信息如下：

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 0.0.0.0:32768 0.0.0.0:\* LISTEN

tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN

tcp 0 0 0.0.0.0:6000 0.0.0.0:\* LISTEN

tcp 0 0[127.0.0.1](https://baike.baidu.com/item/127.0.0.1):631 0.0.0.0:\* LISTEN

udp 0 0 0.0.0.0:32768 0.0.0.0:\*

udp 0 0 0.0.0.0:8888 0.0.0.0:\*

udp 0 0 0.0.0.0:111 0.0.0.0:\*

udp 0 0 0.0.0.0:882 0.0.0.0:\*

可以看到udp处有“0.0.0.0:8888”的内容，说明服务程序已经正常运行，可以接收主机上任何IP地址且端口为8888的数据。

如果这时再执行./udpserv &命令，就会看到如下信息：

bind error: Address already in use

说明已经有一个服务程序在运行了。

运行程序

执行./udpclient 127.0.0.1命令来启动客户程序，使用127.0.0.1来连接服务程序，执行效果如下：

Hello,World!

Hello,World!

this is a test

this is a test

^d

输入的数据都正确从服务程序返回了，按ctrl+d可以结束输入，退出程序。

如果服务程序没有启动，而执行客户程序，就会看到如下信息：

$ ./udpclient 127.0.0.1

test

read error: Connection refused

说明指定的IP地址和端口没有服务程序绑定，客户程序就退出了。这就是使用connect()的好处，注意，这里错误信息是在向服务程序发送数据后收到的，而不是在调用connect()时。如果使用[tcpdump](https://baike.baidu.com/item/tcpdump)程序来[抓包](https://baike.baidu.com/item/%E6%8A%93%E5%8C%85)，会发现收到的是ICMP的错误信息。