

# Linux应用编程之Socket

## Socket介绍

Socket也称套接字，是一种通信机制。

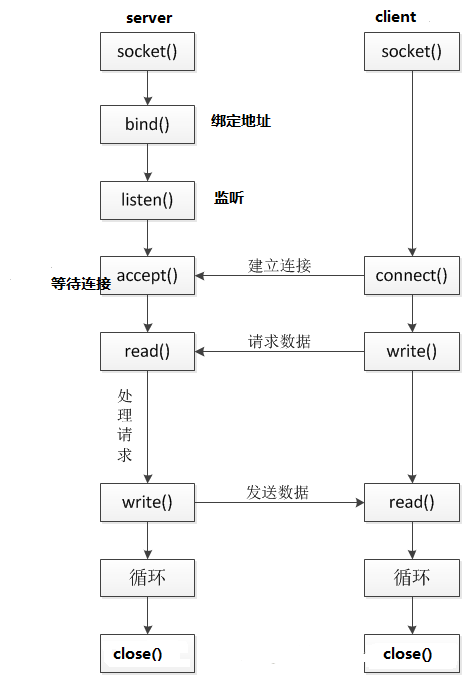
Socket就是我们建立网络连接的工具。

Socket明确的将客户和服务器区分开。

Socket机制可以实现将多个客户连接到服务器。

## Socket连接

Socket连接过程：



### 服务器端

1. 服务器应用程序用系统调用socket来创建一个套接字
2. 用系统调用bind给套接字命名
3. 服务器进程开始等待客户进行连接（**系统调用listen的作用是：创建一个队列并将其用于存放来自客户的进入连接**）
4. 服务器通过系统调用accept来接受客户的连接（**服务器调用accept时，它会创建一个与原有的命名socket不同的新socket。这个新socket只用于与这个特定的客户端进行通信**）

* #include <sys/types.h>  
  #include <sys/socket.h>  
  #include <stdio.h>  
  #include <sys/un.h>  
  #include <unistd.h>  
  #include <stdlib.h>  
    
  int main()  
  {  
   int server\_sockfd, client\_sockfd;  
   int server\_len, client\_len;  
   struct sockaddr\_un server\_address;  
   struct sockaddr\_un client\_address;  
    
  /\* 1.Remove any old socket and create an unnamed socket for the server. \*/  
    
   unlink("server\_socket");  
   server\_sockfd = socket(AF\_UNIX, SOCK\_STREAM, 0);  
    
  /\* 2.Name the socket. \*/  
    
   server\_address.sun\_family = AF\_UNIX;  
   strcpy(server\_address.sun\_path, "server\_socket");  
   server\_len = sizeof(server\_address);  
   bind(server\_sockfd, (struct sockaddr \*)&server\_address, server\_len);  
    
  /\* 3.Create a connection queue and wait for clients. \*/  
    
   listen(server\_sockfd, 5);  
   while(1)   
   {  
   char ch;  
   printf("server waiting\n");  
    
  /\* 4.Accept a connection. \*/  
    
   client\_len = sizeof(client\_address);  
   client\_sockfd = accept(server\_sockfd,   
   (struct sockaddr \*)&client\_address, &client\_len);  
    
  /\* 5.We can now read/write to client on client\_sockfd. \*/  
    
   read(client\_sockfd, &ch, 1);  
   ch++;  
   write(client\_sockfd, &ch, 1);  
   close(client\_sockfd);  
   }  
  }

### 客户端

1. 客户调用socket创建一个未命名的sokcet
2. 将服务器的命名套接字作为一个地址来调用connect与服务器建立连接

#include <sys/types.h>  
#include <sys/socket.h>  
#include <stdio.h>  
#include <sys/un.h>  
#include <unistd.h>  
#include <stdlib.h>  
  
int main()  
{  
 int sockfd;  
 int len;  
 struct sockaddr\_un address;  
 int result;  
 char ch = 'A';  
  
/\* 1.Create a socket for the client. \*/  
  
 sockfd = socket(AF\_UNIX, SOCK\_STREAM, 0);  
  
/\* 2.Name the socket, as agreed with the server. \*/  
  
 address.sun\_family = AF\_UNIX;  
 strcpy(address.sun\_path, "server\_socket");  
 len = sizeof(address);  
  
/\* Now connect our socket to the server's socket. \*/  
  
 result = connect(sockfd, (struct sockaddr \*)&address, len);  
  
 if(result == -1) {  
 perror("oops: client1");  
 exit(1);  
 }  
  
/\* 3.We can now read/write via sockfd. \*/  
  
 write(sockfd, &ch, 1);  
 read(sockfd, &ch, 1);  
 printf("char from server = %c\n", ch);  
 close(sockfd);  
 exit(0);  
}

### Socket属性

#### Socket的域

域指定Socket通信中使用的网络介质。

|  |  |
| --- | --- |
| 域 | 说明 |
| AF\_UNIX | UNIX域协议（文件系统套接字） |
| AF\_INET | ARPA因特网协议（UNIX网络套接字） |
| AF\_ISO | ISO标准协议 |
| AF\_NS | 施乐（Xerox）网络系统协议 |
| AF\_IPX | Novell IPX协议 |
| AF\_APPLETALK | Appletalk DDS |

#### Socket类型

一个Socket域可能有不同的通信方式，而每种通信方式又有不同的特性。

因特网协议提供了两种通讯协议：流（stream）和数据报（datagram）。

* 流套接字
* 流套接字（在某些方面类似于标准的输入/输出流）提供一个有序、可靠、双向字节流的连接。
* 由类型SOCK\_STREAM指定，它们是在AF\_INET域中通过**TCP/IP**连接实现的。
* 数据报套接字
* 与流套接字相反，数据报套接字不建立和维持一个连接。它对可以发送的数据报的长度有限制。数据报作为一个单独的网络消息被传输，它可能丢失、复制或乱序到达。
* 由类型SOCK\_DGRAM指定，它们是在AF\_INET域中通过UDP/IP连接实现的，提供的是一种无序的不可靠服务。

#### Socket协议

只要底层的传输机制允许不止一个协议来提供要求的套接字类型，我们就可以为套接字选择一个特定的协议。常用的UNIX网络套接字和文件系统套接字不需要选择一个特定的协议，使用默认值即可。

### 创建Socket

socket系统调用创建一个套接字并返回一个描述符，该描述符可用来访问该socket。

函数原型：

#include <sys/types.h>  
#include <sys/socket.h>  
  
int socket(int domain, int type, int protocol);

参数含义：

domain：指定协议族也就是socket域，可选择的参数见上面介绍域的表格。  
type：指定socket通信类型，SOCK\_STREAM或SOCK\_DGRAM。  
protocol：指定使用的协议，0表示使用默认协议。

### Socket地址

每个Socket域都有其自己的地址格式。

AF\_UNIX域Socket地址由sockaddr\_un描述，定义在sys/un.h中。

struct sockaddr\_un{  
 sa\_family\_t sun\_family; /\* AF\_UNIX \*/  
 char sun\_path[]; /\* pathname \*/  
};

AF\_UNIX域中Socket地址由sun\_path所指定，长度有限制（Linux规定是108个字符）。

AF\_INET域中，Socket地址由sockaddr\_in描述，定义在netinet/in.h中。

struct sockaddr\_un{  
 short int sin\_family; /\* AF\_INET \*/  
 unsigned short int sin\_port; /\* Port number \*/  
 struct in\_addr sin\_addr; /\* Internet address \*/  
};

IP地址结构in\_addr定义：

struct in\_addr{  
 unsigned long int s\_addr;  
}

### Socket命名

要想让通过socket调用创建的套接字可以被其他进程使用，服务器就必须给套接字命名。

AF\_UNIX套接字就会关联到一个文件系统的路径名。AF\_INET套接字就会关联到一个IP端口号。

#include <sys/socket.h>  
  
int bind(int socket, const struct sockaddr \*address, size\_t address\_len);

bind系统调用把参数address中的地址分配给与文件描述符socket关联的未命名套接字。地址结构的长度由参数address\_len传递。

bind调用时需要将一个特定的地址结构指针转化为指向通用地址类型（structsockaddr \*）。

### 创建Socket队列

为了能够在套接字上接受进入的连接，服务器程序必须创建一个队列来保存未处理的请求。使用listen系统调用来完成这一工作。

#include <sys/socket.h>  
  
int listen(int socket,int backlog);

backlog为可以容纳的未处理连接的最大数目。超过这个数字后，往后的连接会被拒绝。常用的值是5。

### 接受连接

服务器创建并命名套接字之后，可以通过accept系统调用来等待客户建立对该套接字的连接。

#include <sys/socket.h>  
  
int accept(int socket, struct sockaddr \*address, size\_t \*address\_len);

accept系统调用只有当有客户程序试图连接到socket参数指定的套接字上时才返回。这里客户是指在套接字队列中排第一个的未处理连接。accept函数将创建一个新套接字来与客户进行通讯，并且返回新套接字的描述符。

address参数应放入连接客户的地址，如果不关心客户地址，也可以将address参数指定为空指针。

address\_len参数指定客户地址结构体的长度。当这个调用返回时，address\_len将被设置为连接客户地址结构的实际长度。

### 请求连接

客户程序通过在一个未命名套接字和服务器监听套接字之间建立连接的方法来连接到服务器。是通过connect调用来完成这一工作。

#include <sys/socket.h>  
  
int connect(int socket, const struct sockaddr \*address, size\_t address\_len);

socket参数指定的套接字将连接到参数address指定的服务器套接字。address\_len参数指定address指向的结构的长度。

### 关闭Socket

可以调用close函数来终止服务器和客户上的套接字连接。

#include <unistd.h>  
  
int close(int fd);

应该在连接的两端都关闭套接字。

### 主机字节序与网络字节序

不同的计算机使用不同的字节序来表示整数（数据的大小端模式）。为了使不同类型的计算机通过网络传输的多字节整数达成一致，需要定义一个网络字节序（网络字节序为大端模式）。它们通过定义在头文件netinet/in.h中的函数来完成这一工作。

unsigned short int htons(uint16\_t hostshort);  
unsigned long int htonl(uint32\_t hostlong);  
unsigned short int ntohs(uint16\_t netshort);  
unsigned long int ntohl(uint32\_t netlong);

这些函数将16位和32位整数在主机字节序和标准网络字节序之间进行转换。函数名是与之对应的转换操作的简写形式。例如“host to network, short”（htons，短整数从主机字节序到网络字节序的转换）。

### Socket通信例程

/\*  
 \* 服务器端代码  
\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <sys/types.h>  
#include <unistd.h>  
#include <errno.h>  
#include <netinet/in.h>  
#include <netdb.h>  
#include <arpa/inet.h>  
  
const int MAX\_LINE = 2048;  
const int PORT = 6000;  
const int LISTENQ = 5;  
  
int main(int argc , char \*\*argv)  
{  
 /\*声明服务器地址和客户链接地址\*/  
 struct sockaddr\_in servaddr , cliaddr;  
  
 /\*声明服务器监听套接字和客户端链接套接字\*/  
 int listenfd , connfd;  
 pid\_t childpid;  
  
 /\*声明缓冲区\*/  
 char buf[MAX\_LINE];  
  
 socklen\_t clilen;  
  
 /\*(1) 初始化监听套接字listenfd\*/  
 if((listenfd = socket(AF\_INET , SOCK\_STREAM , 0)) < 0)  
 {  
 perror("socket error");  
 exit(1);  
 }//if  
  
 /\*(2) 设置服务器sockaddr\_in结构\*/  
 bzero(&servaddr , sizeof(servaddr));  
  
 servaddr.sin\_family = AF\_INET;  
 servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //表明可接受任意IP地址  
 servaddr.sin\_port = htons(PORT);  
  
 /\*(3) 绑定套接字和端口\*/  
 if(bind(listenfd , (struct sockaddr\*)&servaddr , sizeof(servaddr)) < 0)  
 {  
 perror("bind error");  
 exit(1);  
 }//if  
  
 /\*(4) 监听客户请求\*/  
 if(listen(listenfd , LISTENQ) < 0)  
 {  
 perror("listen error");  
 exit(1);  
 }//if  
  
 /\*(5) 接受客户请求\*/  
 for( ; ; )  
 {  
 clilen = sizeof(cliaddr);  
 if((connfd = accept(listenfd , (struct sockaddr \*)&cliaddr , &clilen)) < 0 )  
 {  
 perror("accept error");  
 exit(1);  
 }//if  
  
 //新建子进程单独处理链接  
 if((childpid = fork()) == 0)   
 {  
 close(listenfd);  
 //str\_echo  
 ssize\_t n;  
 char buff[MAX\_LINE];  
 while((n = read(connfd , buff , MAX\_LINE)) > 0)  
 {  
 write(connfd , buff , n);  
 }  
 exit(0);  
 }//if  
 close(connfd);  
 }//for  
   
 /\*(6) 关闭监听套接字\*/  
 close(listenfd);  
}

/\*  
 \* 客户端代码  
\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <sys/types.h>  
#include <unistd.h>  
#include <errno.h>  
#include <netinet/in.h>  
#include <netdb.h>  
#include <arpa/inet.h>  
  
const int MAX\_LINE = 2048;  
const int PORT = 6000;  
  
/\*readline函数实现\*/  
ssize\_t readline(int fd, char \*vptr, size\_t maxlen)  
{  
 ssize\_t n, rc;  
 char c, \*ptr;  
  
 ptr = vptr;  
 for (n = 1; n < maxlen; n++) {  
 if ( (rc = read(fd, &c,1)) == 1) {  
 \*ptr++ = c;  
 if (c == '\n')  
 break; /\* newline is stored, like fgets() \*/  
 } else if (rc == 0) {  
 \*ptr = 0;  
 return(n - 1); /\* EOF, n - 1 bytes were read \*/  
 } else  
 return(-1); /\* error, errno set by read() \*/  
 }  
  
 \*ptr = 0; /\* null terminate like fgets() \*/  
 return(n);  
}  
  
  
int main(int argc , char \*\* argv)  
{  
 /\*声明套接字和链接服务器地址\*/  
 int sockfd;  
 struct sockaddr\_in servaddr;  
  
 /\*判断是否为合法输入\*/  
 if(argc != 2)  
 {  
 perror("usage:tcpcli <IPaddress>");  
 exit(1);  
 }//if  
  
 /\*(1) 创建套接字\*/  
 if((sockfd = socket(AF\_INET , SOCK\_STREAM , 0)) == -1)  
 {  
 perror("socket error");  
 exit(1);  
 }//if  
  
 /\*(2) 设置链接服务器地址结构\*/  
 bzero(&servaddr , sizeof(servaddr));  
 servaddr.sin\_family = AF\_INET;  
 servaddr.sin\_port = htons(PORT);  
 if(inet\_aton(argv[1] , &servaddr.sin\_addr) < 0)  
 {  
 printf("inet\_aton error for %s\n",argv[1]);  
 exit(1);  
 }//if  
  
 /\*(3) 发送链接服务器请求\*/  
 if( connect(sockfd , (struct sockaddr \*)&servaddr , sizeof(servaddr)) < 0)  
 {  
 perror("connect error");  
 exit(1);  
 }//if  
  
 /\*(4) 消息处理\*/  
 char sendline[MAX\_LINE] , recvline[MAX\_LINE];  
 while(fgets(sendline , MAX\_LINE , stdin) != NULL)   
 {  
 write(sockfd , sendline , strlen(sendline));  
  
 if(readline(sockfd , recvline , MAX\_LINE) == 0)  
 {  
 perror("server terminated prematurely");  
 exit(1);  
 }//if  
  
 if(fputs(recvline , stdout) == EOF)  
 {  
 perror("fputs error");  
 exit(1);  
 }//if  
 }//while  
  
 /\*(5) 关闭套接字\*/  
 close(sockfd);  
}

## 网络信息

如果给定一个计算机的名字，那就可以通过调用解析地址的主机数据库函数来确定它的IP地址。这些函数是通过查询网络配置文件来完成这一工作，如/etc/hosts文件或网络信息服务。

主机库函数在接口头文件netdb.h中申明，如下：

#include <netdb.h>  
  
struct host \*gethostbyaddr(const void \*addr, size\_t len, int type);  
struct host \*gethostbyname(const char \*name);

这些函数返回的结构至少会包含以下几个成员：

struct hostent{  
 char \*h\_name; /\* 主机名称 \*/  
 char \*\*h\_aliases; /\* 别称的列表 \*/  
 int h\_addrtype; /\* 地址类型 \*/  
 int h\_length; /\* 地址字节长度 \*/  
 char \*\*h\_addr\_list; /\* 地址的列表 \*/  
}

如果没有与我们查询的主机或地址相关的数据项，这些信息函数将会返回一个空指针；

类似的，与服务及相关联端口号有关信息也可以通过一些服务信息函数来获取。如下：

#include <netdb.h>  
  
struct servent \*getservbyname(const char \*name, const char \*proto);  
struct servent \*getservbyport(int port, const char \*proto);

proto参数指定用于连接该服务的协议，取值为tcp或udp。

结构servent至少包含以下几个成员：

struct servent{  
 char \*s\_name; /\* 服务名称 \*/  
 char \*\*s\_aliases; /\* 别称列表 \*/  
 int s\_port; /\* IP端口 \*/  
 char \*s\_proto; /\* 服务协议 \*/  
}

相应的IP地址转换函数：

#include <arpa/inet.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
  
char \*inet\_ntoa(struct in\_addr in);  
int inet\_aton(const char \*string, struct in\_addr \*addr);

函数名是与之对应的转换操作的简写形式。例如“network to ascii ”（ntoa，网络字节序地址转换为acsii码地址）。

获取主机名称函数：

#include <unistd.h>  
  
int gethostname(char \*name, int namelength);

获取的主机名称会被写入参数name，而参数namelength指定了name的长度。成功返回0，失败返回-1。

### Socket选项

可以通过setsockopt函数设置选项来控制套接字的行为，它的定义如下：

#include <sys/socket.h>  
  
int setsockopt(int socket, int level, int option\_name,  
 const void \*option\_value, size\_t option\_len);

socket：标识一个套接口的描述字。

level：选项定义的层次；支持SOL\_SOCKET、IPPROTO\_TCP、IPPROTO\_IP和IPPROTO\_IPV6。

option\_name：需设置的选项。

option\_value：指针，指向存放选项待设置的新值的缓冲区。

option\_len：option\_value缓冲区长度。

## 多客户

在上面的Socket通信例程的服务器端代码就是一个利用多进程实现的多客户连接服务器。除了多进程外，还有一种方法实现多客户

### select系统调用

在网络程序中，一个进程同时处理多个文件描述符是很常见的情况。select()系统调用可以使进程检测同时等待的多个I/O设备，当没有设备准备好时，select()阻塞，其中任一设备准备好时，select()就返回。

select()的调用形式为：

#include <sys/select.h>  
#include <sys/time.h>  
  
int select(int maxfd, fd\_set \*readfds, fd\_set \*writefds, fe\_set \*exceptfds, const struct timeval \*timeout);

参数含义：  
maxfd：使用时记住这个值为待检测的最大文件描述符加1即可；--------》但是有坑啊！  
readfds：指定了被读监控的文件描述符集；Those listed in readfds will be watched to see if characters become available for reading；  
writefds：指定了被写监控的文件描述符集；those in writefds will be watched to see if a write will not block  
exceptfds：指定了被例外条件监控的文件描述符集；  
timeout：很常用的一个timeeval结构，起了定时器的作用：到了指定的时间，无论是否有设备准备好，都返回调用。timeval的结构定义如下：

struct timeval{  
 long tv\_sec; //表示几秒  
 long tv\_usec; //表示几微妙  
}

对以上的调用的函数各参数介绍后，说哈他的大致用法：简单粗暴，就是把需要读，写或者其他监控条件的fd加入相应的fd集用于select进行监控管理，并对这些fd进行轮循监控，当有相应的监控条件被触发后，select就会返回，接下来就是进行相应的操作；用法中说到加入，监控的字眼，于是就得说说select必须了解的几个宏：

void FD\_SET(int fd, fd\_set \*fdset);

宏FD\_SET设置文件描述符集fdset中对应于文件描述符fd的位(设置为1)，即将fd加入fdset集的操作；

void FD\_CLR(int fd, fd\_set \*fdset);

宏FD\_CLR清除文件描述符集fdset中对应于文件描述符fd的位（设置为0)，与上相反操作；

void FD\_ISSET(int fd, fd\_set \*fdset);

在调用select后使用FD\_ISSET来检测文件描述符集fdset中对应于文件描述符fd的位是否被设置,如果置位，就进行相应操作；

void FD\_ZERO(fd\_set \*fdset);

宏FD\_ZERO清除文件描述符集fdset中的所有位(既把所有位都设置为0)。

// About "fd\_set"  
typedef \_\_kernel\_fd\_set fd\_set;  
  
typedef struct{  
 unsigned long fds\_bits [\_\_FDSET\_LONGS];  
} \_\_kernel\_fd\_set;  
  
#define \_\_FDSET\_LONGS (\_\_FD\_SETSIZE/\_\_NFDBITS)  
#define \_\_FD\_SETSIZE 1024  
#define \_\_NFDBITS (8 \* sizeof(unsigned long))

**https://www.cnblogs.com/lp1129/articles/2705556.html** 宏相关深入理解；

操作系统通过宏FD*SETSIZE来声明在一个进程中select所能操作的文件描述符的最大数目。通常是1024，既定义FD*SETSIZE为1024，一个整数占4个字节，既32位，那么就是用包含32个元素的整数数组来表示文件描述符集（这是内核代码实现的思路，对理解内核select底层实现很有帮助）；

selest系统调用示例：

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <errno.h>  
#include <unistd.h>  
#include <time.h>  
#include <sys/socket.h>  
#include <sys/select.h>  
#include <sys/types.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <netdb.h>  
#include <fcntl.h>  
  
  
#define PORT 8888  
#define MAX\_LINE 2048  
#define LISTENQ 20  
  
  
int main(int argc , char \*\*argv)  
{  
 int i, maxi, maxfd, listenfd, connfd, sockfd;  
  
 int nready , client[FD\_SETSIZE];  
   
 ssize\_t n, ret;  
   
 fd\_set rset , allset;  
  
 char buf[MAX\_LINE];  
  
 socklen\_t clilen;  
  
 struct sockaddr\_in servaddr , cliaddr;  
  
 /\*(1) 得到监听描述符\*/  
 listenfd = socket(AF\_INET , SOCK\_STREAM , 0);  
  
 /\*(2) 绑定套接字\*/  
 bzero(&servaddr , sizeof(servaddr));  
 servaddr.sin\_family = AF\_INET;  
 servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 servaddr.sin\_port = htons(PORT);  
  
 bind(listenfd , (struct sockaddr \*)&servaddr , sizeof(servaddr));  
  
 /\*(3) 监听\*/  
 listen(listenfd , LISTENQ);  
  
 /\*(4) 设置select\*/  
 maxfd = listenfd;  
 maxi = -1;  
 for(i=0 ; i<FD\_SETSIZE ; ++i)  
 {  
 client[i] = -1;  
 }//for  
 FD\_ZERO(&allset);  
 FD\_SET(listenfd , &allset);  
  
 /\*(5) 进入服务器接收请求死循环\*/  
 while(1)  
 {  
 rset = allset;  
 nready = select(maxfd+1 , &rset , NULL , NULL , NULL);  
   
 if(FD\_ISSET(listenfd , &rset))  
 {  
 /\*接收客户端的请求\*/  
 clilen = sizeof(cliaddr);  
  
 printf("\naccpet connection~\n");  
  
 if((connfd = accept(listenfd , (struct sockaddr \*)&cliaddr , &clilen)) < 0)  
 {  
 perror("accept error.\n");  
 exit(1);  
 }//if   
  
 printf("accpet a new client: %s:%d\n", inet\_ntoa(cliaddr.sin\_addr) , cliaddr.sin\_port);  
  
 /\*将客户链接套接字描述符添加到数组\*/  
 for(i=0 ; i<FD\_SETSIZE ; ++i)  
 {  
 if(client[i] < 0)  
 {  
 client[i] = connfd;  
 break;  
 }//if  
 }//for  
  
 if(FD\_SETSIZE == i)  
 {  
 perror("too many connection.\n");  
 exit(1);  
 }//if  
  
 FD\_SET(connfd , &allset);  
 if(connfd > maxfd)  
 maxfd = connfd;  
 if(i > maxi)  
 maxi = i;  
  
 if(--nready < 0)  
 continue;  
 }//if  
  
 for(i=0; i<=maxi ; ++i)  
 {  
 if((sockfd = client[i]) < 0)  
 continue;  
 if(FD\_ISSET(sockfd , &rset))  
 {  
 /\*处理客户请求\*/  
 printf("\nreading the socket~~~ \n");  
   
 bzero(buf , MAX\_LINE);  
 if((n = read(sockfd , buf , MAX\_LINE)) <= 0)  
 {  
 close(sockfd);  
 FD\_CLR(sockfd , &allset);  
 client[i] = -1;  
 }//if  
 else  
 {  
 printf("clint[%d] send message: %s\n", i , buf);  
 if((ret = write(sockfd , buf , n)) != n)   
 {  
 printf("error writing to the sockfd!\n");  
 break;  
 }//if  
 }//else  
 if(--nready <= 0)  
 break;  
 }//if  
 }//for  
 }//while  
}

## 数据报

在AF\_INT域的的数据报由UDP完成，它的代码实现与TCP/IP有些许差异，在UDP连接中sendto和recvform代替了read和write。

int sendto( SOCKET sockfd, const char FAR\* buf, int len, int flags,const struct sockaddr FAR\* to, int tolen);

sockfd：一个标识套接口的描述字。  
buf：包含待发送数据的缓冲区。  
len：buf缓冲区中数据的长度。  
flags：调用方式标志位。  
to：（可选）指针，指向目的套接口的地址。  
tolen：to所指地址的长度。

int recvfrom( SOCKET sockfd, char FAR\* buf, int len, int flags, struct sockaddr FAR\* from, int FAR\* fromlen);

sockfd：标识一个已连接套接口的描述字。  
buf：接收数据缓冲区。  
len：缓冲区长度。  
flags：调用操作方式。  
from：（可选）指针，指向装有源地址的缓冲区。  
fromlen：（可选）指针，指向from缓冲区长度值。

如下示例：

/\* 客户端 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
  
  
const int SERV\_PORT = 6000;  
  
const int MAXLINE = 2048;  
void dg\_cli(FILE \*fp , int sockfd , const struct sockaddr \*pservaddr , socklen\_t servlen)  
{  
 int n;  
 char sendline[MAXLINE] , recvline[MAXLINE+1];  
   
 while(fgets(sendline , MAXLINE , fp) != NULL)  
 {  
 if(sendto(sockfd , sendline , strlen(sendline) , 0 , pservaddr , servlen) < 0)  
 {  
 perror("sendto error");  
 exit(1);  
 }//if  
  
 if( ( n = recvfrom(sockfd , recvline , MAXLINE , 0 , NULL , NULL)) < 0)  
 {  
 perror("recvfrom error");  
 exit(1);  
 }//if  
  
 recvline[n] = '\0';  
 fputs(recvline , stdout);  
  
 }//while  
}  
  
int main(int argc , char \*\*argv)  
{  
 int sockfd , t;  
 struct sockaddr\_in servaddr;  
 if(argc != 2)  
 {  
 perror("usage: udpcli <IPaddress>");  
 exit(1);  
 }//if  
  
 bzero(&servaddr , sizeof(servaddr));  
 servaddr.sin\_family = AF\_INET;  
 servaddr.sin\_port = htons(SERV\_PORT);  
 if((t = inet\_aton(argv[1], &servaddr.sin\_addr)) <= 0)  
 {  
 perror("inet\_aton error");  
 exit(1);  
 }//if  
   
 if((sockfd = socket(AF\_INET , SOCK\_DGRAM , 0)) < 0)  
 {  
 perror("socket error");  
 exit(1);  
 }//if  
  
 dg\_cli(stdin , sockfd , (struct sockaddr \*)&servaddr , sizeof(servaddr));  
 exit(0);  
}

/\* 服务端 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <sys/socket.h>  
#include <string.h>  
#include <sys/types.h>  
#include <netinet/ip.h>  
  
const int SERV\_PORT = 6000;  
const int MAXLINE = 2048;  
  
void dg\_echo(int sockfd , struct sockaddr \*pcliaddr , socklen\_t clilen)  
{  
 int n;  
 socklen\_t len;  
 char mesg[MAXLINE];  
  
 for( ; ;)  
 {  
 len = clilen;  
 if((n = recvfrom(sockfd , mesg , MAXLINE , 0 , pcliaddr , &len))<0)  
 {  
 perror("recvfrom error");  
 exit(1);  
 }//if  
  
 if((n = sendto(sockfd , mesg , n , 0 , pcliaddr , len)) < 0)  
 {  
 perror("sendto error");  
 exit(1);  
 }//if  
 }//for  
  
}  
  
int main(int argc , char \*\*argv)  
{  
 int sockfd;  
 struct sockaddr\_in servaddr , cliaddr;  
  
 bzero(&servaddr , sizeof(servaddr));  
 servaddr.sin\_family = AF\_INET;  
 servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 servaddr.sin\_port = htons(SERV\_PORT);  
  
 if((sockfd = socket(AF\_INET , SOCK\_DGRAM , 0)) < 0)  
 {  
 perror("socket error");  
 exit(1);  
 }//if  
  
 if(bind(sockfd , (struct sockaddr \*)&servaddr , sizeof(servaddr)))  
 {  
 perror("bind error");  
 exit(1);  
 }//if  
  
 dg\_echo(sockfd , (struct sockaddr \*)&cliaddr , sizeof(cliaddr));   
}