Socket 基本编程

Andrew Huang< bluedrum@163.com>

内容

- Socket编程基本知识
- Socket 常用函数
  - udp处理流程
  - tcp创建流程处理流程
- Windows下socket库的使用
  - Windows socket

Socket编程基本知识

Socket 简介

- BSD Socket接口是TCP/IP网络的API
- 在Linux,Unix和Windows均实现这个接口.BSD Socket的是目前开发网络应用主要接口.绝大部分网络应用均可Socket来开发
- 一个Socket队列是IP应用的基本单位.两个机器通讯相当于两个机器的两个Socket互相通讯的过程.
- Socket 的本意是插座.每一个激活的socket可以看成是一个跟本地某个IP端口绑定的IP包队列.
- 接口设计者最先是将接口放在Unix操作系统里面的。因此一个激活的Socket被设计成特殊的I/O文件, Socket也是一种文件描述符。 .因此操作类似对一个普通文件操作

Socket如何表示IP地址

- **IP**协议规定地址**,ipv4**是**4**个**byte,ipv6**是**16byte,**所有的形式的地址最终都要转换成这样格式**,socket**库才能处理**.**
- 但**socket**库的地址结构的出发点**,**是试图兼容和处理不同网络的地址形式**.**而不是仅仅针对**IP**地址**,**因此设计成一个复杂联合结构 **,**以下是通用地址结构**,**用**sa_family**来指示是哪种类型的网络地址

```
struct sockaddr {
        unsigned short sa_family;   /* 地址族，   AF_xxx */
        char sa_data[14];           /* 14  字节的协议地址  */
        };
```

- **sa_family**为**AF_INET** 表示**Ipv4**地址**,**此时**sockaddr**的定义变成如下形式**.ipv4**大部分处理**struct sockaddr_in,**定义在**netinet/in.h**

```
struct sockaddr_in {
    short int sin_family;              /* 地址族 */
    unsigned short int sin_port;       /* 端口号 */
    struct in_addr sin_addr;           /* IP地址 */
    unsigned char sin_zero[8];         /* 填充0 以保持与
                                       struct sockaddr同样大小 */
};
```

- in_addr定义如下

```
typedef struct in_addr {
    union {
    struct {u_char s_b1,s_b2,s_b3,s_b4;} s_un_b;
    struct {u_short s_w1,s_w2;} S_un_w;
            u_long s_addr;
    }s_un;
  } in_addr;
```

  - 这一形式把4个byte 的IP地址拆成了三种形式,以便各种情况都能处理,但实际绝大部分是使用u_long s_addr
- 综上所述,设置一个IPV4的地址需要设socket_in 三个参数
  - sa_family = AF_INET, sin_port设为 端口, sin_addr.s_addr设为整数形式地址
  - 但是由于socketaddr的先天性的原因,把这个结构搞得非常复杂,这是开发者一定要注意的.

其它形式的地址表示

- 人们常用的是点分表示法的格式.即形如"192.168.0.1"这样字符串表示法,这样格式需要转成整数才能被socket库使用,因此socket库提供如下转换函数
  - #include <arpe/inet.h>
  - **int inet_aton(const char *strptr,struct in_addr *addrptr);**
    - 点分字符串转in_addr结构
    - 返回：1——串有效，0——串出错
  - in_addr_t inet_addr(const char *strptr);
    - 点分字符串转in_addr结构
    - 返回：若成功，返回32位二进制的网络字节序地址；若出错，则返回INADDR_NONE
    - 类似inet_aton,但inet_addr更为通用
  - char *inet_ntoa (struct in_addr inaddr);
    - 将in_addr结构转为点分法字符串
    - 返回：指向点分十进制数串指针
    - 注意转换后点分格式是带端口的 202.116.34.194.4000 ,表示 IP:202.116.34.194 端口4000

1)inet_addr

	inet_addr（将网络地址转成二进制的数字）
相关函数	inet_aton，inet_ntoa
表头文件	#include<sys/socket.h> #include<netinet/in.h> #include<arpa/inet.h>
定义函数	unsigned long int inet_addr(const char *cp);
函数说明	inet_addr()用来将参数 cp 所指的网络地址字符串转换成网络所使用的二进制数字。网络地址字符串是以数字和点组成的字符串，例如:“163.13.132.68”。
返回值	成功则返回对应的网络二进制的数字，失败返回-1。

2)inet_aton

	inet_aton（将网络地址转成网络二进制的数字）
相关函数	inet_addr，inet_ntoa
表头文件	#include<sys/scoket.h> #include<netinet/in.h> #include<arpa/inet.h>
定义函数	int inet_aton(const char * cp,struct in_addr *inp);

函数说明	inet_aton()用来将参数cp所指的网络地址字符串转换成网络使用的二进制的数字，然后存于参数inp所指的in_addr结构中。
	结构in_addr定义如下
	struct in_addr
	{
	unsigned long int s_addr;
	};

返回值	成功则返回非0值，失败则返回0。

3)inet_ntoa

	inet_ntoa（将网络二进制的数字转换成网络地址）
相关函数	inet_addr，inet_aton
表头文件	#include<sys/socket.h> #include<netinet/in.h> #include<arpa/inet.h>
定义函数	char * inet_ntoa(struct in_addr in);
函数说明	inet_ntoa()用来将参数 in 所指的网络二进制的数字转换成网络地址，然后将指向此网络地址字符串的指针返回。
返回值	成功则返回字符串指针，失败则返回 NULL。

其他形式的地址表示(二)

- 新版转换函数,支持 ipv6 地址转换
  - Ipv6 的点分形式地址以[]包括起来,以 ipv4 区别开来
  - int inet_pton(int family,const char *strptr,void *addrptr);
  - const char *inet_ntop(int family,const void *addrptr,char *strptr,size_t len);
  - 函数中 p 表示 presentation ,即点分地址示 202.116.34.194
  - 函数中 n 表示 numeric 数值格式
  - 这两个函数里的端口数字都是网络序.

网络字节序,本机字节序

- 不同的计算机系统采用不同的字节序存储数据,同样一个4字节的32位整数,在内存中存储的方式就不同. 字节序分为小端字节序(Little Endian)和大端字节序(Big Endian), Intel处理器大多数使用小端字节序, Motorola处理器大多数使用大端(Big Endian)字节序;
- 小端字节序:低序字节存储在起始地址
- 大端字节序:高序字节存储在起始地址

- Ip协议是定义在可以在任何操作系统或CPU传输数据的协议,在ip传输一个数字,必须统一规定的字节序,否则在传输时会发混乱.这个统一数字字节序叫网络序.TCP/IP规定网络序采用大端字节序, 相对的, CPU本身数字表示顺序称为本机序.
- IP包在发送前,必须把本机序转换为网络序.在接收后,也需要把网络序转为本地序,这样才能正常使用.
- 在各个操作系统都会要实现四个转换函数
  - include <netinet/in.h>
  - unit16_t htons(uint16_t host);
  - unit32_t htonl(uint32_t host);
  - unit16_t ntohs(uint16_t net);
  - unit32_t ntohl(uint32_t net);

1)ntohl

	ntohl（将32位网络字符顺序转换成主机字符顺序）
相关函数	htonl，htons，ntohs
表头文件	#include<netinet/in.h>
定义函数	unsigned long int ntohl(unsigned long int netlong);
函数说明	ntohl()用来将参数指定的32位netlong转换成主机字符顺序。
返回值	返回对应的主机字符顺序。
范例	参考getservent()。

2)ntohs

	ntohs（将16位网络字符顺序转换成主机字符顺序）
相关函数	htonl，htons，ntohl
表头文件	#include<netinet/in.h>
定义函数	unsigned short int ntohs(unsigned short int netshort);
函数说明	ntohs()用来将参数指定的16位netshort转换成主机字符顺序。
返回值	返回对应的主机顺序。
范例	参考getservent()。

3)htonl

	htonl（将 32 位主机字符顺序转换成网络字符顺序）
相关函数	htons，ntohl，ntohs
表头文件	#include<netinet/in.h>
定义函数	unsigned long int htonl(unsigned long int hostlong);
函数说明	htonl（）用来将参数指定的 32 位 hostlong 转换成网络字符顺序。
返回值	返回对应的网络字符顺序。
范例	参考 getservbyport()或 connect

4)htons

	htons（将16位主机字符顺序转换成网络字符顺序）
相关函数	htonl，ntohl，ntohs
表头文件	#include<netinet/in.h>
定义函数	unsigned short int htons(unsigned short int hostshort);
函数说明	htons()用来将参数指定的16位hostshort转换成网络字符顺序。
返回值	返回对应的网络字符顺序。
范例	参考connect()。

使用域名

- 很多用户习惯于采用容易记忆的域名来访问远端机器.但Socket只采用数字形式的IP地址来访问
- 因此必须一个函数来完成这一转换,以便程序界面更加友好.
  - 如ie的地址栏,就是IE自行在内部把域名作了转换,后然后使用IP地址去联接

- 不同的操作系统采用不同函数来作域名转换
  - 一般采用gethostbyname()作域名转换

Linux 通用处理IP/域名的代码

```
/* 用到如下声明
   typedef OS_STATUS int;
   #define OS_OK 0
   #define OS_ERROR -1
   #define OS_NOT_FOUND -2
  必须使用头文件  #include <netdb.h>
*/
```

```
OS_STATUS   LinuxGetIpByHost(char * host_name,char * ip_addr,int ip_len)
{
        struct sockaddr_in addr;
         struct hostent *host;
        char **alias;

    /* 这里我们先假设是点分形式 IP，通过 IP 获得主机信息*/
         if(inet_aton(host_name,&(addr.sin_addr))!=0)
         { /* 如果成功,重新以点分字符串形式,写到返回 BUFFER*/
            snprintf(ip_addr,ip_len,"%s", inet_ntoa(addr.sin_addr));
            return OS_OK;
         }
         else
         { /* 如果转换不成功,把 host_name 域名来查找一次，通过域名找主机信息*/
             host=gethostbyname(host_name);
         }
     /* 查询域名失败,可能是输入错误地址,返回*/
        if(host == NULL)
          return OS_ERROR;
    /* 查询结果可能存有多 IP 地址,即一个域名可能对应多个 IP,一般采用第一个*/
        alias=host->h_addr_list;
        if(alias == NULL)
                return OS_NOT_FOUND;

          /* 以点分字符串形式返回*/
             snprintf(ip_addr,ip_len,"%s",inet_ntoa(*(struct in_addr *)(*alias)));
            return (OS_OK);
}
```

Win32通用处理IP/域名的代码

```
/* 用到如下声明
   typedef OS_STATUS int;
   #define OS_OK 0
   #define OS_ERROR -1
   #define OS_NOT_FOUND -2
  必须使用头文件  #include <winsock.h>
*/
```

```
OS_STATUS   _Win32GetIpByHost(char * host_name,char * ip_addr,int ip_len)
{
    struct sockaddr_in dest;
    struct hostent      *hp = NULL;

     memset(&dest, 0, sizeof(dest));
     dest.sin_family = AF_INET;
 /* 首先假设是一个点分形式 IP 地址,用转换函数转换一次*/
     if ((dest.sin_addr.s_addr = inet_addr(host_name)) != INADDR_NONE)
   { /* 如果成功,直接 IP 地址转换回去,多余空格,0 字符将被去掉*/
            snprintf(ip_addr,ip_len,"%s", inet_ntoa(dest.sin_addr));
            return (OS_OK);
     }
  /* 然后假设是一个域名,尝试转换一次*/
if ((hp = gethostbyname(host_name)) != NULL)
          {
             memcpy(&(dest.sin_addr), hp->h_addr, hp->h_length);
             dest.sin_family = hp->h_addrtype;
              snprintf(ip_addr,ip_len,"%s", inet_ntoa(dest.sin_addr));
            return OS_OK;
          }
return OS_ERROR;
}
```

Socket 基本接口函数

Socket的理解

- Socket从字面上理解就是插头,翻译成术语就是套接字.
- 可以用水龙头/漏斗来与Socket来作类比.
  - 一个网络通道(TCP或UDP)可以看成是一个水管.
  - 在两个网络设备之间交换的数据看成是水.
  - 在网络上传输数据相当于是通过水管在送水.

- 这样一个机器要接到水,必须先要开一个水头龙.这个龙头就是SOCKET,因为一个机器可能有多个水龙头,为了区别开,必须要编号,这个编号就是端口号(PORT)
- 然后这个机器要建一个水管延伸到另外一台机器上的指定的漏斗上(远端的指定端口的SOCKET).这个过程就是创建网络联接的过程
- 建立成功后,对方在它的机器上的漏斗(SOCKET)放水,通过水管,接到本机的水龙头(SOCKET),就可以得到水
- TCP管道相当于是一点水都不漏的水管,对方的水要不一点都送不过来,要不完整的把所有水都送过来
- UDP相当于是一个有可能漏水的水管,接收方必须要自己用量杯来检查才可以知道送过来的水是不是跟发送方的水一样多.

Socket的创建流程

- 因此一个台机器首先要知道远端机器的IP地址和端口才能与对方通讯.
- 首先要创建一个SOCKET(水龙头)
  - 用socket()函数
- 为这个SOCKET绑定端口(为水龙头指定一个编号)
  - 用bind()函数
- 然后通过这个socket跟远方机器建立联接
  - TCP用connect()函数,UDP没有连接概念,在发送时指定
- 建立联系成功后,通过接,发函数来交换数据
  - send(),recv(),sendto(),recvfrom(),write(),read()
- 使用完毕联接,要关闭水管
  - close()
  - Windows下使用 closesocket();
- TCP服务器有一些特殊要求,必须要先建一个侦听的SOCKET/水龙头,当有一个TCP客户端发来请求,还要再建专门一个水龙头与这个客户端建立水管与这个一客户端来交换数据.
  - listen(),accept()函数

socket（建立一个socket通信）

	socket（建立一个 socket 通信）
相关函数	accept，bind，connect，listen
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int socket(int domain,int type,int protocol);
函数说明	socket()用来建立一个新的 socket，也就是向系统注册，通知系统建立一通信端口。

参数	type 有下列几种数值: **SOCK_STREAM 提供双向连续且可信赖的数据流，即 TCP**。支持 OOB 机制，在所有数据传送前必须使用 connect()来建立连线状态。 **SOCK_DGRAM 使用不连续不可信赖的数据包连接** **即 UDP** SOCK_SEQPACKET 提供连续可信赖的数据包连接 **SOCK_RAW 提供原始网络协议存取** SOCK_RDM 提供可信赖的数据包连接 SOCK_PACKET 提供和网络驱动程序直接通信。 protocol 用来指定 socket 所使用的传输协议编号，通常此参考不用管它，设为 0 即可。
返回值	成功则返回 socket 描述符，失败返回-1。
错误代码	EPROTONOSUPPORT 参数 domain 指定的类型不支持参数 type 或 protocol 指定的协议 ENFILE 核心内存不足，无法建立新的 socket 结构 EMFILE 进程文件表溢出，无法再建立新的 socket EACCESS 权限不足，无法建立 type 或 protocol 指定的协议 ENOBUFS/ENOMEM 内存不足 EINVAL 参数 domain/type/protocol 不合法

socket() 参数-domain

- 完整的定义在/usr/include/bits/socket.h 内，底下是常见的协议:
  - PF_UNIX/PF_LOCAL/AF_UNIX/AF_LOCAL UNIX 进程通信协议
  - PF_INET/AF_INET Ipv4网络协议
  - PF_INET6/AF_INET6 Ipv6 网络协议
  - PF_IPX/AF_IPX IPX-Novell协议
  - PF_NETLINK/AF_NETLINK 核心用户接口装置
  - PF_X25/AF_X25 ITU-T X.25/ISO-8208 协议
  - PF_AX25/AF_AX25 业余无线AX.25协议
  - PF_ATMPVC/AF_ATMPVC 存取原始ATM PVCs
  - PF_APPLETALK/AF_APPLETALK appletalk（DDP）协议
  - PF_PACKET/AF_PACKET 初级封包接口

bind()

	bind（对 socket 定位）
相关函数	socket，accept，connect，listen
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int bind(int sockfd,struct sockaddr * my_addr,int addrlen);
函数说明	bind()用来给已经打开的 Socket sockfd,指定本地地址和端口,一个 sockaddr 结构，用于 UDP 收发两端,和 TCP 客户端的使用

参数	addrlen 为 sockaddr 的结构长度。
返回值	成功则返回 0，失败返回-1，错误原因存于 errno 中。
错误代码	EBADF 参数 sockfd 非合法 socket 处理代码。 EACCESS 权限不足 ENOTSOCK 参数 sockfd 为一文件描述词，非 socket。

- bind (),把一个socket跟端口绑定,这样socket才能被外部访问.
  - int bind(int sockfd, struct sockaddr *my_addr, int addrlen)
  - sockfd:是由socket调用返回的文件描述符.
  - addrlen:是sockaddr结构的长度.
  - my_addr:是一个指向sockaddr的指针. 但一般使用sockaddr_in 定义,来处理ipv4地址
  - struct sockaddr_in{
    unsigned short sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero[8]; };
  - 我们主要使用Internet所以sin_family一般为AF_INET,sin_addr设置为INADDR_ANY表示可以 和任何的主机通信,sin_port是我们要监听的端口号.sin_zero[8]是用来填充的. bind将本地的端口同socket返回的文件描述符捆绑在一起.成功是返回0,失败的情况和socket一样

connect()

	connect（建立 socket 连线）
相关函数	socket，bind，listen
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int connect (int sockfd,struct sockaddr * serv_addr,int addrlen);
函数说明	connect()用来将参数 sockfd 的 socket 连至参数 serv_addr 指定的网络地址。结构 sockaddr 请参考 bind()。参数 addrlen 为 sockaddr 的结构长度。
返回值	成功则返回 0，失败返回-1，错误原因存于 errno 中。
错误代码	EBADF 参数 sockfd 非合法 socket 处理代码 EFAULT 参数 serv_addr 指针指向无法存取的内存空间 ENOTSOCK 参数 sockfd 为一文件描述词，非 socket。 EISCONN 参数 sockfd 的 socket 已是连线状态 ECONNREFUSED 连线要求被 server 端拒绝。 ETIMEDOUT 企图连线的操作超过限定时间仍未有响应。 ENETUNREACH 无法传送数据包至指定的主机。 EAFNOSUPPORT sockaddr 结构的 sa_family 不正确。 EALREADY socket 为不可阻断且先前的连线操作还未完成。

范例

```
/*  利用 socket 的 TCP client
此程序会连线 TCP server，并将键盘输入的字符串传送给 server。
TCP server 范例请参考 listen（）。
*/
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define PORT 1234
#define SERVER_IP “127.0.0.1”
main()
{
int s;
struct sockaddr_in addr;
char buffer[256];
if((s = socket(AF_INET,SOCK_STREAM,0))<0){
perror(“socket”);
exit(1);
}
/*  填写 sockaddr_in 结构*/
bzero(&addr,sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port=htons(PORT);
addr.sin_addr.s_addr = inet_addr(SERVER_IP);
/*  尝试连线*/
if(connect(s,&addr,sizeof(addr))<0){
perror(“connect”);
exit(1);
}
/*  接收由 server 端传来的信息*/
recv(s,buffer,sizeof(buffer),0);
printf(“%s\n”,buffer);
while(1){
bzero(buffer,sizeof(buffer));
/*  从标准输入设备取得字符串*/
read(STDIN_FILENO,buffer,sizeof(buffer));
/*  将字符串传给 server 端*/
if(send(s,buffer,sizeof(buffer),0)<0){
perror(“send”);
exit(1);
```

	} } }
执行	$ ./connect Welcome to server! hi I am client! /*键盘输入*/ /*<Ctrl+C>中断程序*/

listen()

	listen（等待连接）
相关函数	socket，bind，accept，connect
表头文件	#include<sys/socket.h>
定义函数	int listen(int s,int backlog);
函数说明	listen()用来等待参数 s 的 socket 连线。参数 backlog 指定同时能处理的最大连接要求，如果连接数目达此上限则 client 端将收到 ECONNREFUSED 的错误。 Listen()并未开始接收连线，只是设置 socket 为 listen 模式，真正接收 client 端连线的是 accept()。通常 listen()会 在 socket()，bind()之后调用，接着才调用 accept()。
返回值	成功则返回 0，失败返回-1，错误原因存于 errno
附加说明	listen()只适用 SOCK_STREAM 或 SOCK_SEQPACKET 的 socket 类型。如果 socket 为 AF_INET 则参数 backlog 最大值可设至 128。 backlog:设置请求排队的最大长度.当有多个客户端程序和服务端相连时，使用这个表示可以介绍的排队长度
错误代码	EBADF 参数 sockfd 非合法 socket 处理代码 EACCESS 权限不足 EOPNOTSUPP 指定的 socket 并未支援 listen 模式。

范例	#include<sys/types.h>
	#include<sys/socket.h>
	#include<netinet/in.h>
	#include<arpa/inet.h>
	#include<unistd.h>
	#define PORT 1234
	#define MAXSOCKFD 10
	
	main()
	{
	int sockfd,newsockfd,is_connected[MAXSOCKFD],fd;
	struct sockaddr_in addr;
	int addr_len = sizeof(struct sockaddr_in);
	fd_set readfds;
	char buffer[256];

```
char msg[ ] =”Welcome to server!”;
if ((sockfd = socket(AF_INET,SOCK_STREAM,0))<0){
perror(“socket”);
exit(1);
}

bzero(&addr,sizeof(addr));
addr.sin_family =AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(sockfd,&addr,sizeof(addr))<0){
perror(“connect”);
exit(1);
}
if(listen(sockfd,3)<0){
perror(“listen”);
exit(1);
}

for(fd=0;fd<MAXSOCKFD;fd++)
is_connected[fd]=0;
while(1){
FD_ZERO(&readfds);
FD_SET(sockfd,&readfds);
for(fd=0;fd<MAXSOCKFD;fd++)
if(is_connected[fd]) FD_SET(fd,&readfds);
if(!select(MAXSOCKFD,&readfds,NULL,NULL,NULL))continue;
for(fd=0;fd<MAXSOCKFD;fd++)
if(FD_ISSET(fd,&readfds)){
if(sockfd = =fd){
if((newsockfd = accept (sockfd,&addr,&addr_len))<0)
perror(“accept”);
write(newsockfd,msg,sizeof(msg));
is_connected[newsockfd] =1;
printf(“cnnect from %s\n”,inet_ntoa(addr.sin_addr));
}else{
bzero(buffer,sizeof(buffer));
if(read(fd,buffer,sizeof(buffer))<=0){
printf(“connect closed.\n”);
is_connected[fd]=0;
close(fd);
}else
printf(“%s”,buffer);
```

	} } } }
执行	$ ./listen connect from 127.0.0.1 hi I am client connected closed.

accept

	accept（ listen 侦听后,有客户端联接时,创建 socket）
相关函数	socket，bind，listen，connect
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int accept(int s,struct sockaddr * addr,int * addrlen);
函数说明	accept()用来接受参数 s 的 socket 连线。参数 s 的 socket 必需先经 bind()、listen()函数处理过，当有连线进来时 accept()会返回一个新的 socket 处理代码，往后的数据传送与读取就是经由新的 socket 处理，而原来参数 s 的 socket 能继续使用 accept()来接受新的连线要求。连线成功时，参数 addr 所指的结构会被系统填入远程主机的地址数据，参数 addrlen 为 scokaddr 的结构 长度。关于结构 sockaddr 的定义请参考 bind()。
返回值	成功则返回新的 socket 处理代码，失败返回-1，错误原因存于 errno 中。
错误代码	EBADF 参数 s 非合法 socket 处理代码。 EFAULT 参数 addr 指针指向无法存取的内存空间。 ENOTSOCK 参数 s 为一文件描述词，非 socket。 EOPNOTSUPP 指定的 socket 并非 SOCK_STREAM。 EPERM 防火墙拒绝此连线。 ENOBUFS 系统的缓冲内存不足。 ENOMEM 核心内存不足。
范例	参考 listen()。

accept函数,addrlen参数问题

- int accept(int s,struct sockaddr * addr,int * addrlen);的第三个参数addrlen,是一个输出值,即联接TCP服务器的客户端的sockaddr地址的长度.由accept函数增写,并传给外部调用者
- 在Linux这一函数工作正常
- 在Windows下有一个BUG,即这个输入值在调用前必须赋与正确的长度值,否则accept函数阻塞失败,立刻返回一个错误的socket
- 如下代码在Windows下accept将执行失败,会始终返回INVLID_SOCKET,因为addrlen没有赋值,除非把注释一行代码打开

recv()接收TCP数据

	recv（经 socket 接收数据）
相关函数	recvfrom，recvmsg，send，sendto，socket
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int recv(int s,void *buf,int len,unsigned int flags);
函数说明	recv()用来接收远端主机经指定的 socket 传来的数据，并把数据存到由参数 buf 指向的内存空间，参数 len 为可接收数据的最大长度。
参数	flags 一般设 0。其他数值定义如下: MSG_OOB 接收以 out-of-band 送出的数据。 MSG_PEEK 返回来的数据并不会在系统内删除，如果再调用 recv()会返回相同的数据内容。 MSG_WAITALL 强迫接收到 len 大小的数据后才能返回，除非有错误或信号产生。 MSG_NOSIGNAL 此操作不愿被 SIGPIPE 信号中断返回值成功则返回接收到的字符数，失败返回-1，错误原因存于 errno 中。
错误代码	EBADF 参数 s 非合法的 socket 处理代码 EFAULT 参数中有一指针指向无法存取的内存空间 ENOTSOCK 参数 s 为一文件描述词，非 socket。 EINTR 被信号所中断 EAGAIN 此动作会令进程阻断，但参数 s 的 socket 为不可阻断 ENOBUFS 系统的缓冲内存不足。 ENOMEM 核心内存不足 EINVAL 传给系统调用的参数不正确。
范例	参考 listen()。

send 发送TCP数据

	send（经 socket 传送数据）
相关函数	sendto，sendmsg，recv，recvfrom，socket
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int send(int s,const void * msg,int len,unsigned int falgs);
函数说明	send()用来将数据由指定的 socket 传给对方主机。参数 s 为已建立好连接的 socket。参数 msg 指向欲连线的数据内容，参数 len 则为数据长度。参数 flags 一般设 0，其他数值定义如下 MSG_OOB 传送的数据以 out-of-band 送出。 MSG_DONTROUTE 取消路由表查询 MSG_DONTWAIT 设置为不可阻断运作 MSG_NOSIGNAL 此动作不愿被 SIGPIPE 信号中断。
返回值	成功则返回实际传送出去的字符数，失败返回-1。错误原因存于 errno

错误代码	EBADF 参数 s 非合法的 socket 处理代码。 EFAULT 参数中有一指针指向无法存取的内存空间 ENOTSOCK 参数 s 为一文件描述词，非 socket。 EINTR 被信号所中断。 EAGAIN 此操作会令进程阻断，但参数 s 的 socket 为不可阻断。 ENOBUFS 系统的缓冲内存不足 ENOMEM 核心内存不足 EINVAL 传给系统调用的参数不正确。
范例	参考 connect()

recvfrom()接收UDP数据

	recvfrom（经 socket 接收数据）
相关函数	recv，recvmsg，send，sendto，socket
表头文件	#include<sys/types.h> #include<sys/socket.h>
定义函数	int recvfrom(int s,void *buf,int len,unsigned int flags ,struct sockaddr *from ,int *fromlen);
函数说明	recv()用来接收远程主机经指定的 socket 传来的数据，并把数据存到由参数 buf 指向的内存空间，参数 len 为可接收数据的最大长度。参数 flags 一般设 0，其他数值定义请参考 recv()。参数 from 用来指定欲传送的网络地址，结构 sockaddr 请参考 bind()。参数 fromlen 为 sockaddr 的结构长度。
返回值	成功则返回接收到的字符数，失败则返回-1，错误原因存于 errno 中。
错误代码	EBADF 参数 s 非合法的 socket 处理代码 EFAULT 参数中有一指针指向无法存取的内存空间。 ENOTSOCK 参数 s 为一文件描述词，非 socket。 EINTR 被信号所中断。 EAGAIN 此动作会令进程阻断，但参数 s 的 socket 为不可阻断。 ENOBUFS 系统的缓冲内存不足 ENOMEM 核心内存不足 EINVAL 传给系统调用的参数不正确。
范例	/*利用 socket 的 UDP client 此程序会连线 UDP server，并将键盘输入的字符串传给 server。 UDP server 范例请参考 sendto（）。 */ #include<sys/stat.h> #include<fcntl.h> #include<unistd.h> #include<sys/typs.h> #include<sys/socket.h> #include<netinet/in.h> #include<arpa/inet.h>

	#define PORT 2345
	#define SERVER_IP “127.0.0.1”
	main()
	{
	int s,len;
	struct sockaddr_in addr;
	int addr_len =sizeof(struct sockaddr_in);
	char buffer[256];
	/* 建立 socket*/
	if((s = socket(AF_INET,SOCK_DGRAM,0))<0){
	perror(“socket”);
	exit(1);
	}
	/* 填写 sockaddr_in*/
	bzero(&addr,sizeof(addr));
	addr.sin_family = AF_INET;
	addr.sin_port = htons(PORT);
	addr.sin_addr.s_addr = inet_addr(SERVER_IP);
	while(1){
	bzero(buffer,sizeof(buffer));
	/* 从标准输入设备取得字符串*/
	len =read(STDIN_FILENO,buffer,sizeof(buffer));
	/* 将字符串传送给 server 端*/
	sendto(s,buffer,len,0,&addr,addr_len);
	/* 接收 server 端返回的字符串*/
	len = recvfrom(s,buffer,sizeof(buffer),0,&addr,&addr_len);
	printf(“receive: %s”,buffer);
	}
	}

执行	(先执行 udp server 再执行 udp client) hello /*从键盘输入字符串*/ receive: hello /*server 端返回来的字符串*/

sendto发送UDP数据

	sendto（经 socket 传送数据）
相关函数	send , sendmsg,recv , recvfrom , socket
表头文件	#include < sys/types.h > #include < sys/socket.h >
定义函数	int sendto (int s , const void * msg, int len, unsigned int flags, const struct sockaddr * to , int tolen) ;

函数说明	sendto() 用来将数据由指定的 socket 传给对方主机。参数 s 为已建好连线的 socket, 如果利用 UDP 协议则不需经过连线操作。参数 msg 指向欲连线的数据内容，参数 flags 一般设 0，详细描述请参考 send()。参数 to 用来指定欲传送的网络地址，结构 sockaddr 请参考 bind()。参数 tolen 为 sockaddr 的结果长度。
返回值	成功则返回实际传送出去的字符数，失败返回－1，错误原因存于 errno 中。
错误代码	EBADF 参数 s 非法的 socket 处理代码。 EFAULT 参数中有一指针指向无法存取的内存空间。 WNOTSOCK canshu s 为一文件描述词，非 socket。 EINTR 被信号所中断。 EAGAIN 此动作会令进程阻断，但参数 s 的 soket 为补课阻断的。 ENOBUFS 系统的缓冲内存不足。 EINVAL 传给系统调用的参数不正确。

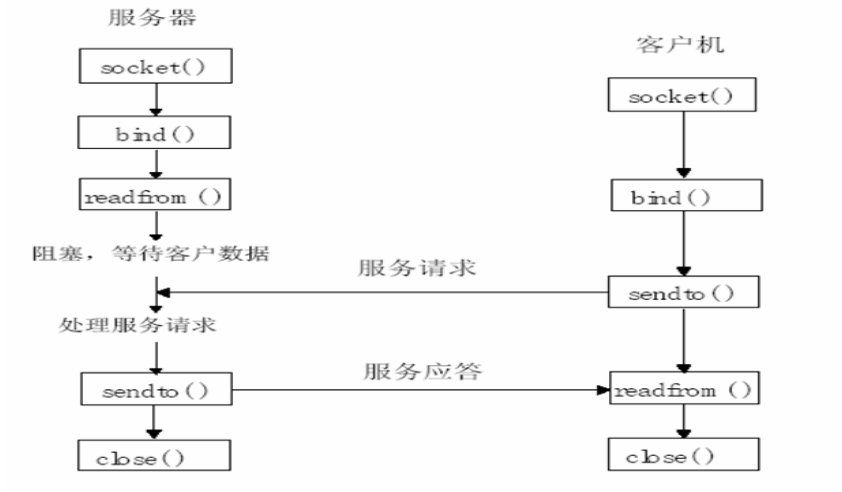
范例	#include < sys/types.h >
	#include < sys/socket.h >
	# include <netinet.in.h>
	#include <arpa.inet.h>
	#define PORT 2345 /*使用的 port*/
	main(){
	int sockfd,len;
	struct sockaddr_in addr;
	char buffer[256];
	/*建立 socket*/
	if(sockfd=socket (AF_INET,SOCK_DGRAM,0))<0){
	perror (“socket”);
	exit(1);
	}
	/*填写 sockaddr_in 结构*/
	bzero (&addr, sizeof(addr));
	addr.sin_family=AF_INET;
	addr.sin_port=htons(PORT);
	addr.sin_addr=hton1(INADDR_ANY) ;
	if (bind(sockfd, &addr, sizeof(addr))<0){
	perror(“connect”);
	exit(1);
	}
	while(1){
	bezro(buffer,sizeof(buffer));
	len = recvfrom(socket,buffer,sizeof(buffer), 0 , &addr &addr_len);
	/*显示 client 端的网络地址*/
	printf(“receive from %s\n “ , inet_ntoa(addr.sin_addr));
	/*将字串返回给 client 端*/
	sendto(sockfd,buffer,len,0,&addr,addr_len);”

	} }
执行	请参考 recvfrom()

- 数据读取,这是Linux专用函数
  - ssize_t read(int fd,void *buf,size_t nbyte)
  - read函数是负责从fd中读取内容.当读成功时,read返回实际所读的字节数,如果返回的值是0 表示已经读到文件的结束了,小于0表示出现了错误.如果错误为EINTR说明读是由中断引起的, 如果是ECONNREST表示网络连接出了问题
- 数据写入,这是Linux专用函数
  - ssize_t write(int fd,const void *buf,size_t nbytes)
  - write函数将buf中的nbytes字节内容写入文件描述符fd.成功时返回写的字节数.失败时返回-1. 并设置errno变量. 在网络程序中,当我们向套接字文件描述符写时有俩种可能.
  - 1)write的返回值大于0,表示写了部分或者是全部的数据.
  - 2)返回的值小于0,此时出现了错误.我们要根据错误类型来处理.
  - 如果错误为EINTR表示在写的时候出现了中断错误.
  - 如果为EPIPE表示网络连接出现了问题(对方已经关闭了连接).

UDP 编程

UDP处理流程



UDP服务器处理流程-初始化

```
/* 打开一个 socket ,AF_INET 表示是 IPV4 版本,
    SOCK_DGRAM 表示 UDP*/
     if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
          perror("socket");
          exit(1);
     }

#define MYPORT 4950

/* 初始化本地地址和端口 */
     my_addr.sin_family = AF_INET;          /* host byte order */
     my_addr.sin_port = htons(MYPORT);      /* short, network byte order */
     my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
     bzero(&(my_addr.sin_zero), 8);         /* zero the rest of the struct */
     /* 跟指定端口绑定 */
     if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) \
         == -1) {
          perror("bind");
          exit(1);
     }
```

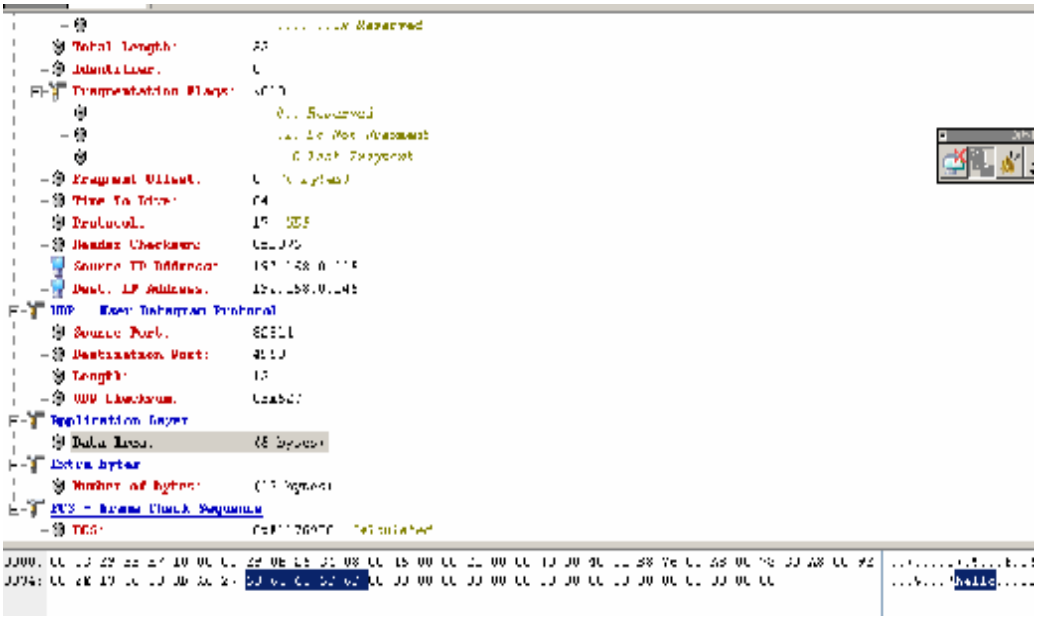
UDP服务器处理流程-接收数据

```
/* 遁环接数据 */
     while(1)
     {
          addr_len = sizeof(struct sockaddr);
          /* 不断接接收客户端数据,their_addr是对端的包含对端的IP和地址*/
          if ((numbytes=recvfrom(sockfd, buf, MAXBUFLEN, 0, (struct sockaddr *)&their_addr,
               &addr_len)) == -1)
          {
               perror("recvfrom");
               exit(1);
          }

          printf("got packet from %s\n",inet_ntoa(their_addr.sin_addr));
          printf("packet is %d bytes long\n",numbytes);
          buf[numbytes] = '\0';
          printf("packet contains \"%s\"\n",buf);
```

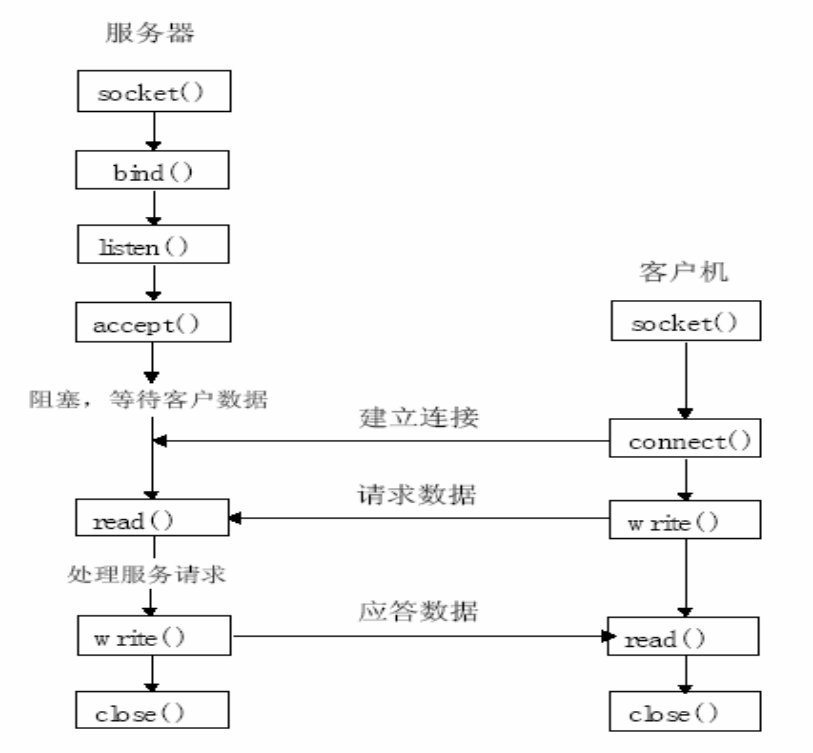
```
        /*接上一页*/

        if ((numbytes=sendto(sockfd, buf, numbytes, 0, (struct sockaddr *)&their_addr, sizeof(struct
            sockaddr))) == -1)
        {
                perror("sendto");
                exit(1);
        }
      }
     /* 关闭socket */
     close(sockfd);
    }
```



TCP编程

TCP 处理流程



ＴＣＰ服务器的流程处理

- TCP服务器的编程比较独特，不是只采用一个socket而是采用多个socket进行处理．可以用电话人工接线员来比喻
  - 首先接线员(TCP服务器）自己有个侦听的通道(socket),以接收用户请求
    - Socket()->Bind()->listen()
  - 当用一个用户接上来(connect())请接线员转接时，接线员会重新开一根线来接通这个用户与他想找的用户的通话
    - accept()会生成一个新的socket，用于服务器与这个客户端之间通话
  - 所有的包的交互都是这个新的socket进行传输的

Socket编程注意事项

- 端口不能被重复打开
  - 一个打开的端口再次绑定时会报错
  - bind: Address already in use
- 一般建议用singal来检测程序,在退出时关闭相应的socket

思考题

- 在自己数据包里传输数字必须要做字节序转换吗？
- TCP服务器，在线的客户端的socket会有多个，他们之间,socket库能保证客户端socket互相之间能通讯吗？

课堂练习

- 1.编写一个Linux ECHO程序 /UDP/TCP
  - 尝试用C语言结构去发送或接收一个包
- 2.编写一个Windows ECHO程序 /UDP/TCP

扩展练习

- TCP客户端,或服务器端的编写都是固定的流程
  - 请设计一组通用接口来简化socket程序的开发
  - 一般会有如下接口
    - open_tcp_server(short port);
    - connect_tcp_server(char * ipaddr,short port);

- TCP/UDP客户端,或服务器端的编写都是固定的流程
  - 请设计一组通用接口来简化socket程序的开发
  - 一般会有如下接口
    - open_tcp_server(short port);,open_udp_server()
    - connect_tcp_server(char * ipaddr,short port);
    - connect_udp_server(char * ipaddr,short port);

- 设一个基类 CSocket,
  - 它有CUDPSocket,CTCPSocket子类
  - 再扩展CUDPSServer/CUDPClient
  - CTCPServer/CTCPClient()
  - 至少如下公有成员,Host,port
  - 至少如下公有方法open(),close(),read(),write()