



电子科技大学
University of Electronic Science and Technology of China

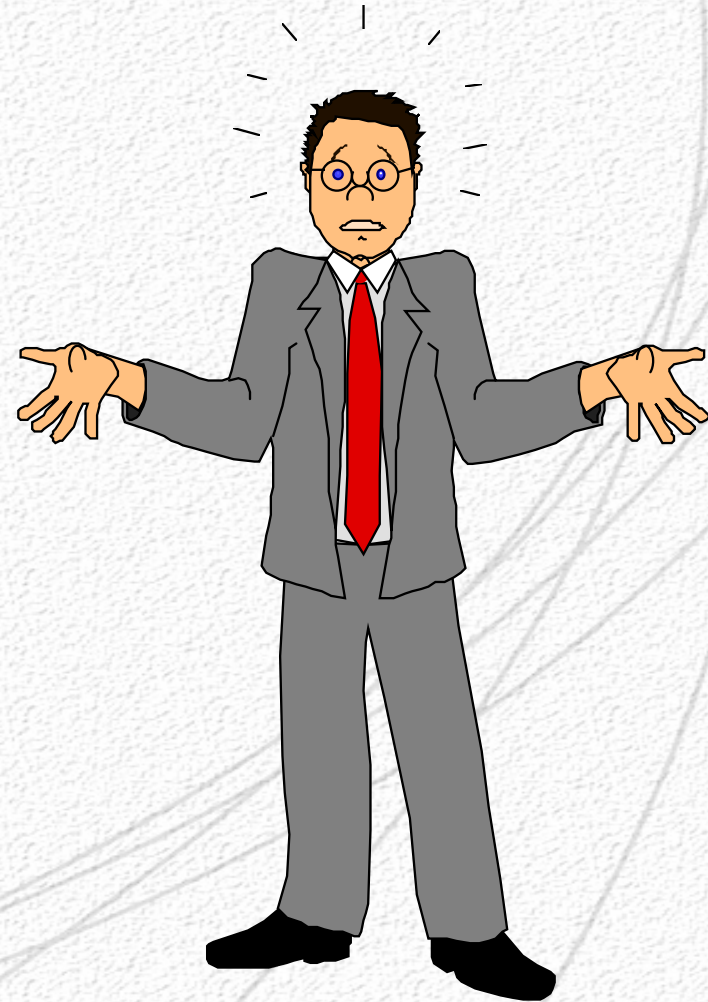
Lecture 2: Elementary Sockets

Ren Liyong

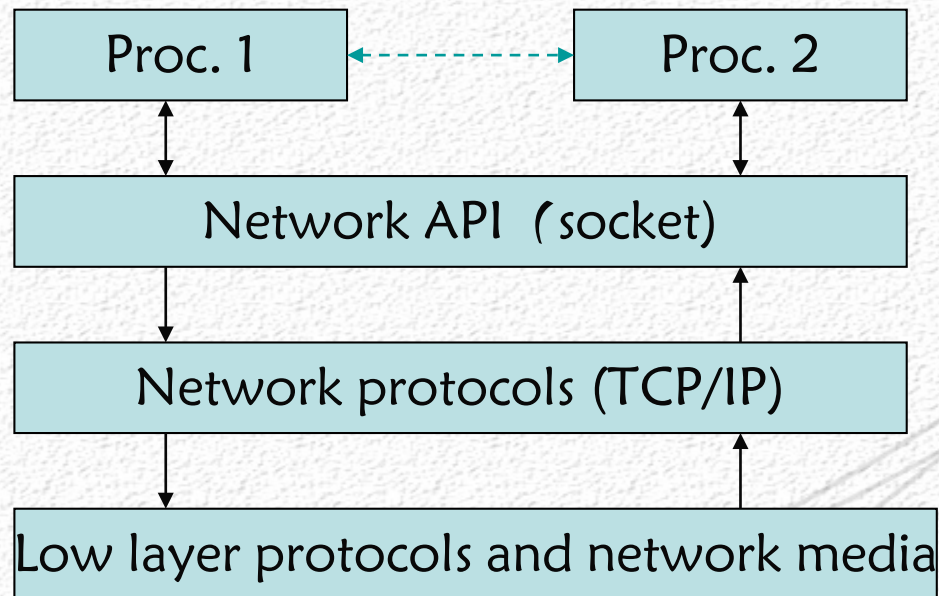
School of Software

www.uestc.edu.cn

1. Basic of Sockets
2. Structure of Sockets
3. Socket APIs
4. TCP C/S example
5. UDP C/S example
6. Name and Address Conversions



1. UNIX网络编程的两个方向：Socket, TLI
2. 套接字是一种网络API，程序员可以用之开发网络程序。



1. 套接字支持多种通信协议：

- ① **Unix**: Unix系统内部协议
- ② **INET**: IP版本4
- ③ **INET6**: IP版本6

2. 套接字类型，即应用程序希望的通信服务类型

- ① **SOCKET_STREAM**: 双向可靠数据流，对应TCP
- ② **SOCKET_DGRAM**: 双向不可靠数据报，对应UDP
- ③ **SOCKET_RAW**: 是低于传输层的低级协议或物理网络提供的套接字类型，可以访问内部网络接口。

Socket Address Structures (IPv4)

Most socket functions require a pointer to a socket address structure as an argument. **Each** supported protocol suite defines **its own** socket address structure. (netinet/in.h)

```
typedef uint32_t in_addr_t;
typedef uint16_t in_port_t;
typedef unsigned short sa_family_t;

struct in_addr{
    in_addr_t s_addr;
};
```

```
struct sockaddr_in {
    uint8_t      sin_len;
    sa_family_t  sin_family;
    in_port_t    sin_port;
    struct in_addr sin_addr;
    char         sin_zero[8];
};
```

IPv6 address length is 128. (netinet/in.h)

```
typedef uint16_t in_port_t;  
typedef unsigned short sa_family_t;  
  
struct in6_addr{  
    uint8_t      s6_addr[16];  
};
```

```
struct sockaddr_in6{  
    uint8_t      sin6_len;  
    sa_family_t  sin6_family;  
    in_port_t    sin6_port;  
    uint32_t     sin6_flowinfo;  
    struct in6_addr sin6_addr;  
};
```


Comparison of socket address Sockets

sockaddr_in{ }

长度	AF_INET
16位端口号	
32位IP地址	
未用	

固定长度（**16**字节）

sockaddr_in6{ }

长度	AF_INET6
16位端口号	
32位流标签	
128位IPv6地址	

固定长度（**24**字节）

Generic Socket Address Structures

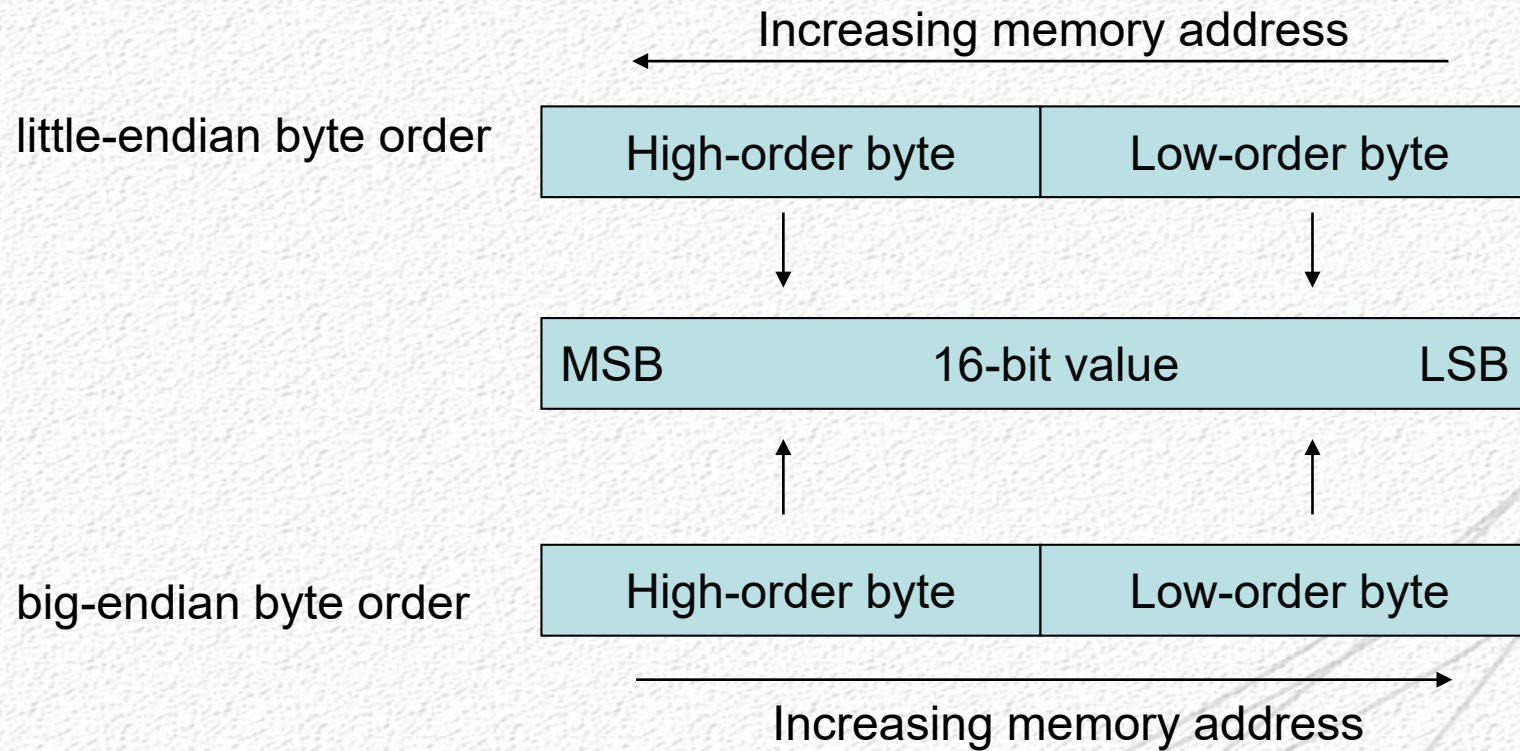
1. 由于套接字函数需接收来自不同协议的地址结构，ANSI的办法是使用通用的指针类型，即（`void *`），但套接字函数先于ANSI之前定义，其方法是定义一个通用的套接字地址结构。<sys/socket.h>

```
struct sockaddr{  
    uint8_t      sa_len;  
    sa_family_t  sa_family;  
    char         sa_data[14];  
};
```

2. This requires that any calls to these functions must **cast** the pointer to the protocol-specific to be a generic. Examp.:

```
struct sockaddr_in serv  
  
bind(sockfd, (struct sockaddr *)&serv, sizeof(serv));
```


Byte Ordering Functions



Byte Ordering Functions (cont.)

```
#include <netinet/in.h>
```

```
uint16_t htons(uint16_t hostshort)
```

```
uint32_t htonl(uint32_t hostlong)
```

both return: value in network byte order

```
uint16_t ntohs(uint16_t netshort)
```

```
uint32_t ntohl(uint32_t netlong)
```

both return: value in host byte order

Byte Manipulation Functions

```
#include <string.h>
```

```
void bzero(void *dest, size_t nbyte);
```

```
void bcopy(const void *src, void *dest, size_t nbyte);
```

```
int  bcmp(const void *src, void *dest, size_t nbyte);
```

these functions come from BSD

```
void *memset(void *dest, int c, size_t len);
```

```
void *memcpy(void *dest, const void *src, size_t nbyte);
```

```
int  memcmp(const void *ptr1, const void *ptr2, size_t nbyte)
```

these functions come from ANSI

Address Transform Functions

```
#include <arpa/inet.h>
```

```
int inet_aton(const char *cp, struct in_addr *inp);
```

返回：1-串有效，0-串有错

```
in_addr_t inet_addr(const char *cp);
```

返回：若成功，返回32位二进制的网络字节序地址，若有错，则返回INADDR_NONE. 过时函数

- **inet_aton**函数将cp所指的字符串转换成32位的网络字节序二进制，并通过指针inp来存储。这个函数需要对字符串所指的地址进行有效性验证。但如果cp为空，函数仍然成功，但不存储任何结果。
- **inet_addr**进行相同的转换，但不进行有效性验证，也就是说，所有 2^{32} 种可能的二进制值对**inet_addr**函数都是有效的。

`char *inet_ntoa(struct in_addr in);`

返回：指向点分十进制数串的指针

- 函数`inet_ntoa`将32位的网络字节序二进制IPv4地址转换成相应的点分十进制数串。但由于返回值所指向的串留在静态内存中，这意味着函数是不可重入的。
- 需要注意的是这个函数是以结构为参数的，而不是指针。
- 上述三个地址转换函数都只能处理IPv4协议，而不能处理IPv6地址。

■ `#include <arpa/inet.h>`

`int inet_pton(int family, const char *src, void *dst);`

返回：1-成功，0—输入无效，-1:出错

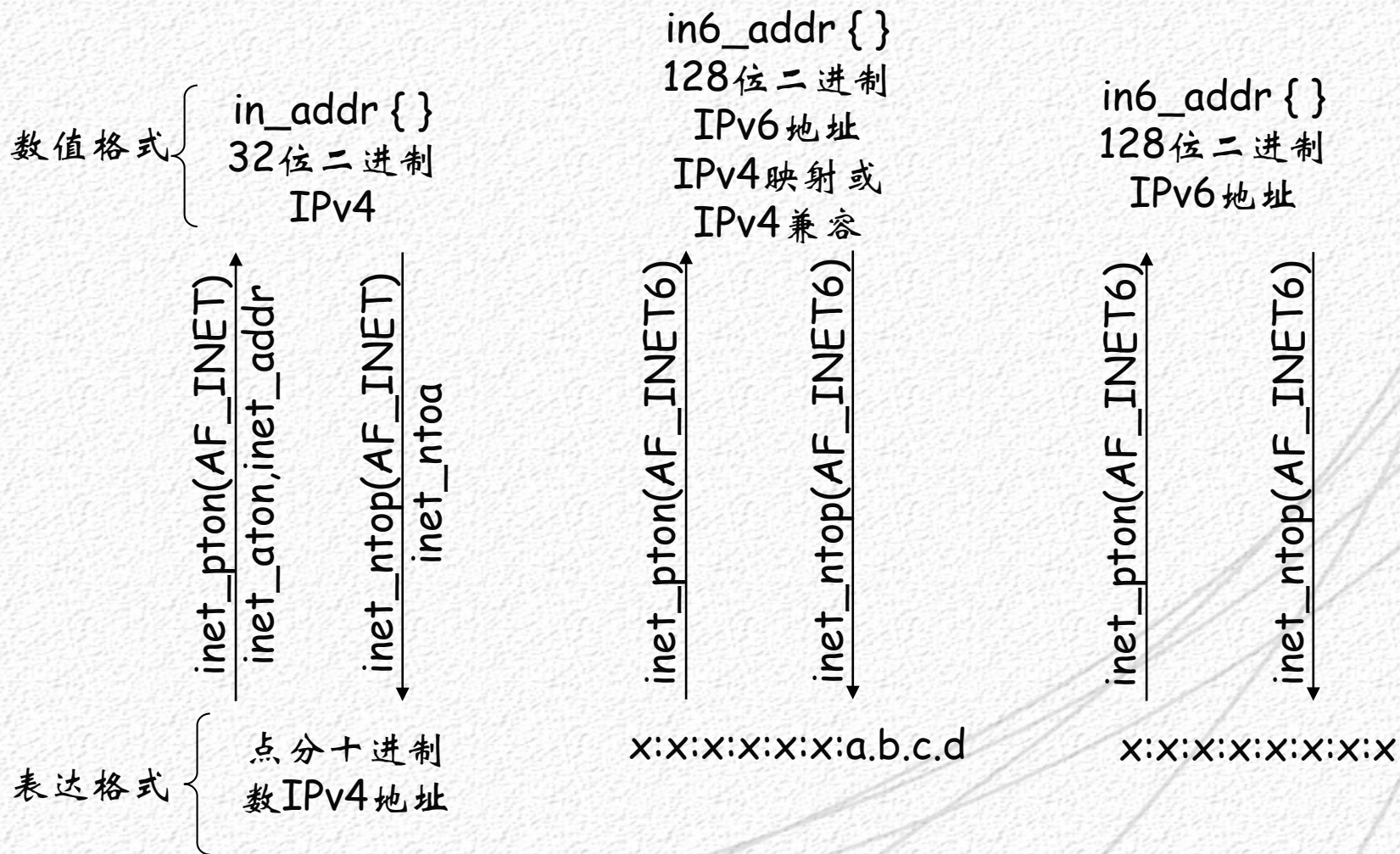
`const char *inet_ntop(int family, const void *src, char *dst, size_t cnt);`

返回：指向结果的指针——成功，NULL—出错

■ `family`参数可以是`AF_INET`,也可以是`AF_INET6`。

■ 如果长度参数`cnt`太小，无法容纳表达式格式结果，则返回一个空串。另外，目标指针`dst`调用前必须先由调用者分配空间。

地址转换函数小结





基本的套接字函数


```
#include <sys/socket.h>
```

```
int socket(int family, int type, int protocol)
```

返回：非负描述字—成功；-1—出错。

family:协议族；type:套接字类型；protocol：一般为0，除原始套接字外。

family

AF_INET

AF_INET6

AF_LOCAL

AF_ROUTE

AF_KEY

type

SOCK_STREAM

SOCK_DGRAM

SOCK_RAW

```
#include <sys/socket.h>
```

```
int bind(int sockfd, struct sockaddr *addr, socklen_t len)
```

返回：0—成功；-1—出错

- 该函数给套接字分配一个本地协议地址，注意：协议地址addr是通用地址。
- 一般而言，服务器调用此函数，而客户则很少调用它。
- 绑定地址时，可以指定地址和端口号，也可以指定其中之一，甚至一个也不指定。通配地址：INADDR_ANY

IP地址	端口	结果
通配地址	0	内核选择IP地址和端口号
通配地址	非0	内核选择IP地址，进程指定端口
本地IP	0	进程指定IP地址，内核选择端口
本地IP	非0	进程指定IP地址和端口号

■ 另外，需要注意以下几点：

- 参数**addr**中的相关字段在初始化时，必须是网络字节序；
- 如果由内核来选择**IP**地址和临时端口号，函数并不返回所选择的值。为了获得这些值，进程必须调用**getsockname**函数
- 函数**bind**返回的一个常见错误是：**EADDRINUSE**，我们可以通过设置套接口选项**SO_REUSEADDR**。

```
...  
struct sockaddr_in addr;  
int port = 1234;  
  
addr.sin_family = AF_INET;  
addr.sin_addr.s_addr = htonl(INADDR_ANY);  
addr.sin_port = htons(port);  
  
if (bind(fd, (struct sockaddr *)&addr, sizeof(addr)) == -1)  
{  
    /* 错误处理 */  
}
```



```
#include <sys/socket.h>
```

```
int listen(int sockfd, int backlog)
```

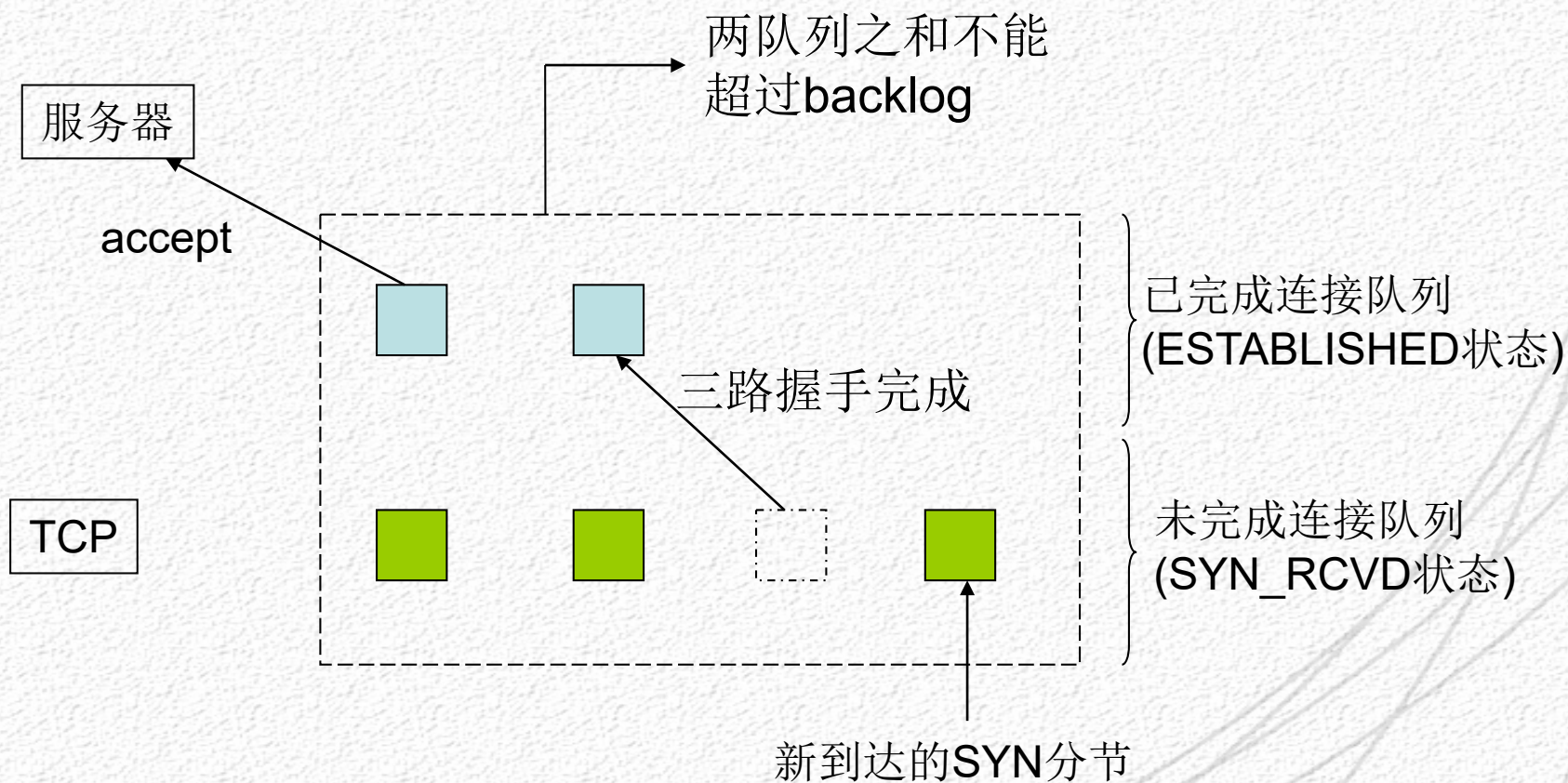
返回：0—成功；-1—出错；

■ 函数listen仅被服务器调用,它完成两件事情：

- 函数listen将未连接的套接字转化成被动套接字，指示内核应接受指向此套接字的连接请求；
- 函数的第二个参数规定了内核为此套接字排队的最大连接个数；

■ 对于给定的监听套接字，内核要维护两个队列

- 未完成连接队列
- 已完成连接队列
- 两个队列之和不超过backlog；



TCP为监听套接口维护的两个队列

■ 另外几点说明：

- 不同的实现对**backlog**有不同的解释，如源自**Berkeley**的实现将**backlog**增加一个模糊因子，把它乘以**1.5**，再作为两个队列之和；
- 不要把**backlog**定义为**0**，因为有些实现允许**1**个连接排队，而有些实现不允许连接排队；
- 当一个客户**SYN**到达时，若两个队列都是满的，**tcp**就忽略此分节，且不发送**RST**。这是因为，这种情况是暂时的，客户**tcp**将重发**SYN**，期望不久的将来就能在队列中找到空闲条目。如果发送**RST**，将会出现？

```
#include <sys/socket.h>
```

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

返回：0—成功；-1—出错；

■ 函数connect激发TCP的三路握手过程；仅在成功或出错返回；错误有以下几种情况：

- 如果客户没有收到**SYN**分节的响应（总共**75**秒，这之间需要可能需要重发若干次**SYN**），则返回**ETIMEDOUT**。
- 如果对客户的**SYN**的响应是**RST**，则表明该服务器主机在该端口上没有进程在等待。函数返回错误**ECONNREFUSED**；
- 如果客户发出的**SYN**在中间路由器上引发一个目的地不可达**ICMP**错误，则如第一种情况，连续发送**SYN**，直到规定时间，返回**EHOSTUNREACH**或**ENETUNREACH**。

- 客户在调用connect前不必非得调用bind函数，此时，内核会选择一个合适的IP地址和临时端口号；
- 如果函数connect失败，则套接字不可再用，必须关闭。不能再对此套接字再调用函数connect。

```
#include <sys/socket.h>
```

```
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

返回：非负描述字—OK； -1—出错；

- accept函数由TCP服务器调用；从已完成连接队列头返回下一个已完成连接；如果该队列空，则进程进入睡眠状态。
- 函数返回的套接字为**连接套接字**，应与监听套接字区分开来
- 该函数最多**返回三个值**：一个既可能是新套接字也可能是错误指示的整数，一个客户进程的协议地址（由cliaddr所指），以及该地址的大小（这后两个参数是值—结果参数）；也就是说，服务器可以通过参数cliaddr来得到请求连接并获得成功的客户的地址和端口号；



```
...  
struct sockaddr_in      servaddr, cliaddr;  
socklen_t   len;  
int         listenfd, connfd;  
  
...  
connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &len);  
if (connfd == -1)    {  
    /* 出错处理 */  
}  
printf(" connection from %s, port = %d\n",  
       inet_ntop(AF_INET, cliaddr.sin_addr, buff, sizeof(buff)),  
       ntohs(cliaddr.sin_port));  
...  

```

■ 客户端：

- 从命令行读入服务器的**IP**地址；并连接到服务器；

■ 服务器端：

- 接收客户的连接请求，并显示客户的**IP**地址和端口号；


```
#include <unistd.h>
```

```
int close(int sockfd);
```

返回:0—OK; -1—出错;

- **close**函数缺省功能是将套接字做上“已关闭”标记，并立即返回到进程。这个套接字不能再为该进程所用。
- 正常情况下，**close**将引发向TCP的四分节终止序列，但在终止前将发送已排队的数据；
- 如果套接字描述符访问计数在调用**close**后大于0（在多个进程共享同一个套接字的情况下），则不会引发TCP终止序列（即不会发送FIN分节）；

```
#include <sys/socket.h>
```

```
int shutdown(int sockfd, int howto);
```

返回:0—OK; -1—出错;

■ 该函数立即发送FIN分节（无论其访问计数是否大于0）。shutdown根据参数howto关闭指定方向的数据传输;

- **SHUT_RD**: 关闭连接的读这一半, 不再接收套接字中的数据且现留在接收缓冲区的数据作废;
- **SHUT_WR**: 关闭连接的写这一半, 当留在套接字发送缓冲区中的数据都被发送, 后跟tcp连接终止序列, 不管访问计数是否大于0; 此后将不能在执行对套接字的任何写操作;
- **SHUT_RDWR**: 连接的读、写都关闭, 这等效于调用shutdown两次, 一次调用是用SHUT_RD, 第二次用SHUT_WR。


```
#include <unistd.h>
```

```
int read(int fd, char *buf, int len);
```

返回:大于0—读写字节大小; -1—出错;

■ 调用函数read时, 有如下几种情况:

- 套接字接收缓冲区接收到数据, 返回接收到的字节数;
- **tcp**协议收到**FIN**数据, 返回**0**;
- **tcp**协议收到**RST**数据, 返回**-1**, 同时**errno**为**ECONNRESET**;
- 进程阻塞过程中接收到信号, 返回**-1**, 同时**errno**为**EINTR**。

```
#include <unistd.h>
```

```
int write(int fd, char *buf, int len);
```

返回:大于0—读写字节大小; -1—出错;

■ 调用函数write, 有如下几种情况:

- 套接字发送缓冲区有足够空间, 返回发送的字节数;
- **tcp**协议接收到**RST**数据, 返回-1, 同时**errno**为**ECONNRESET**; ;
- 进程阻塞过程中接收到信号, 返回-1, 同时**errno**为**EINTR**。


```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
ssize_t send (int s, const void *msg, size_t len, int flags);
```

返回：非0—发送成功的数据长度；-1—出错；

■ **flags** 是传输控制标志，其值定义如下：

- 0：常规操作，如同write()函数
- **MSG_OOB**，发送带外数据。
- **MSG_DONTROUTE**：忽略底层协议的路由设置，只能将数据发送给与发送机处在同一个网络中的机器上。

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
ssize_t recv(int s, void *buf, size_t len, int flags);
```

返回：大于0表示成功接收的数据长度；0: 对方已关闭，-1:出错。

■ **flags**是传输控制标志，其值定义如下：

- 0: 常规操作，如同**read()**函数；
- **MSG_PEEK**: 只查看数据而不读出数据，后续读操作仍然能读该数据；
- **MSG_OOB**: 忽略常规数据，而只读带外数据；
- **MSG_WAITALL**: **recv**函数只有在将接收缓冲区填满后才返回。

■ 实现TCP套接字基本步骤分为服务器端和客户端两部分：

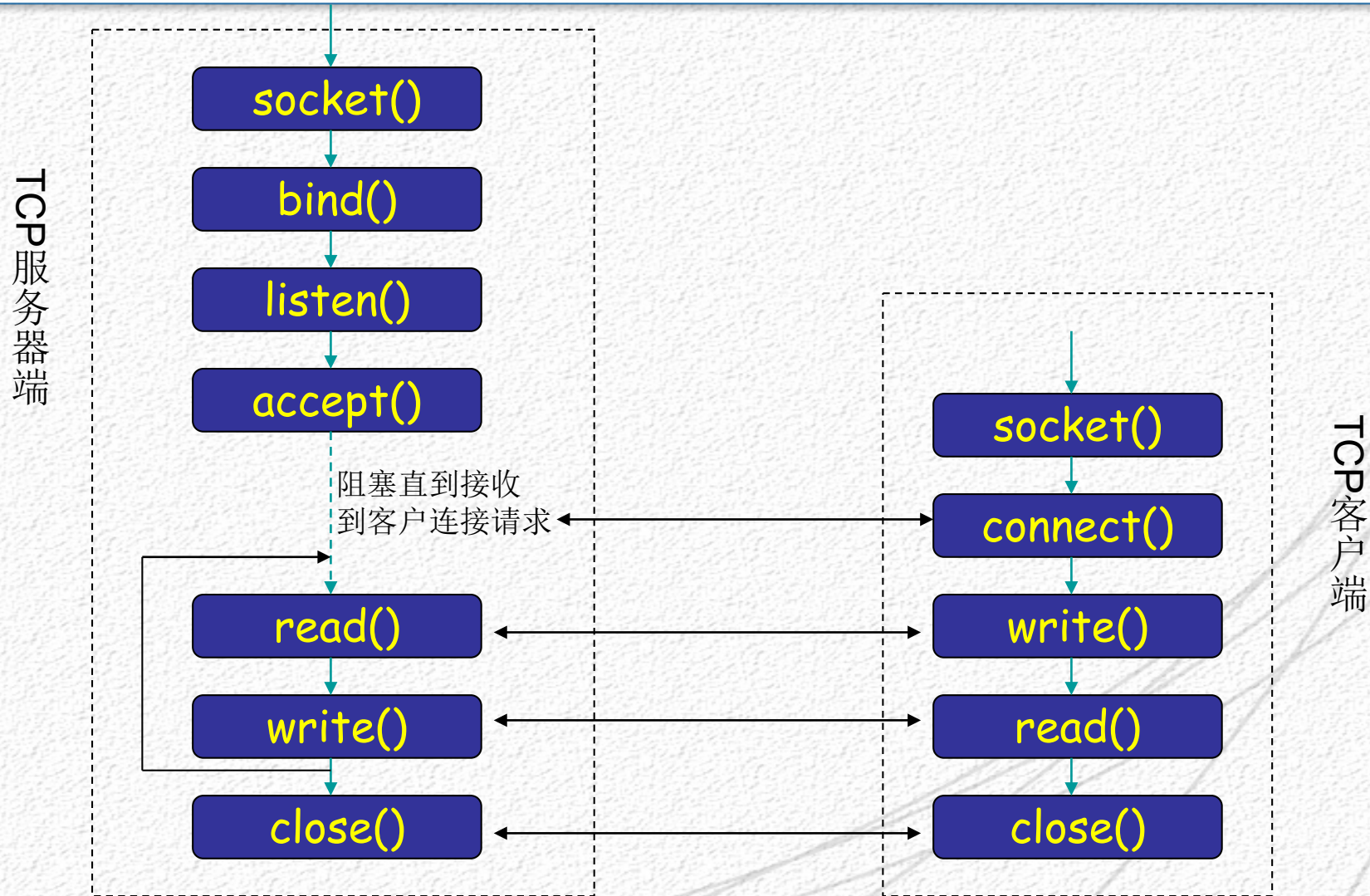
■ 服务器端

- ① 创建套接字；
- ② 绑定套接字；
- ③ 设置套接字为监听模式，进入被动接受连接状态；
- ④ 接受请求，建立连接
- ⑤ 读写数据
- ⑥ 终止连接

■ 客户端步骤

- ① 创建套接字
- ② 与远程服务器建立连接
- ③ 读/写数据;
- ④ 终止连接

TCP套接字编程 (cont.)



```
int main(void)
{
    int sockfd,connect_sock;
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1) {
        perror("create socket failed.");
        exit(-1);
    }
    /* bind sockfd to some address */
    /* listen */
    .....
    loop {
        if((connect_sock=accept(sockfd,NULL,NULL))==-1) {
            perror("Accept error."); exit(-1);
        }
        /* read and process request */
        close(connect_sock);
    }
    close(sockfd);
}
```



```
/* include some header files */  
int main(void)  
{  
    int sockfd;  
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))=-1)  
    {  
        perror("Create socket failed.");  
        exit(-1);  
    }  
    /* connect to server */  
    .....  
    /* send request and receive response */  
    .....  
    close(sockfd);  
}
```

■ 采用客户/服务器模式，完成下列功能：

- 客户根据用户提供的**IP**地址，连接相应的服务器；
- 服务器等待客户的连接，一旦连接成功，则显示客户的**IP**地址，并发欢迎信息给客户；
- 客户接收服务器发送的信息并显示；



```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define PORT 1234
#define BACKLOG 1
int main(void)
{
    int                listenfd, connectfd;
    struct sockadd_in  server, client;
    int                sin_size;

    if((listenfd=socket(AF_INET, SOCK_STREAM, 0))== -1)
    {
        perror("Create socket failed.");
        exit(-1);
    }
}
```

```
int opt = SO_REUSEADDR
setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

bzero(&server, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(PORT);
serv.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(listenfd, (struct sockaddr *)&server, sizeof(struct sockaddr))-1) {
    perror("Bind error.");
    exit(-1);
}

if (listen(listenfd, BACKLOG) == -1) {
    perror("listen error.");
    exit(-1);
}
```



```
sin_size = sizeof(struct sockaddr_in);
while(1) {
    if ((connectfd = accept(listenfd, (struct sockaddr *)&client, &sin_size)) == -1) {
        perror("accept error.");
        exit(-1);
    }
    printf("You get a connection from %s\n", inet_ntoa(client.sin_addr));
    send(connectfd, "Welcome to my server.\n", 22, 0);
    close(connectfd);
} /* while */

close(listenfd);
}
```

```
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define PORT          1234
#define MAXDATASIZE  100
int main(int argc, char *argv[])
{
    int fd, numbytes;
    char      buf[MAXDATASIZE];
    struct hostent *    he;
    struct sockaddr_in server;
    if (argc != 2)
    {
        printf("Usage: %s <IP address>\n", argv[0]);
        exit(-1);
    }
}
```



```
if ((he = gethostbyname(argv[1])) == NULL) {  
    perror("gethostbyname error.");  
    exit(1);  
}  
  
if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {  
    perror("Create socket failed.");  
    exit(1);  
}  
  
bzero(&server, sizeof(server));  
server.sin_family = AF_INET;  
server.sin_port = htons(PORT);  
server.sin_addr = *((struct in_addr *) he->h_addr);
```

```
if (connect(fd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1) {  
    perror("connect failed.");  
    exit(1);  
}  
  
if( ((numbytes = recv(fd, buf, MAXDATASIZE, 0)) == -1) {  
    perror("recv error.");  
    exit(1);  
}  
  
buf[numbytes] = '\0';  
printf("Server Message: %s\n",buf);  
  
close(fd);  
}
```



```
202.115.28.198 - SecureCRT
File Edit View Options Transfer Script Window Help
[lyren@198 lyren]$ ./client1 202.115.28.198
connect failed.: Connection refused
[lyren@198 lyren]$ ./client1 127.0.0.1
Server Message: Welcome to my server.

[lyren@198 lyren]$ ./client1 202.115.28.198
Server Message: Welcome to my server.

[lyren@198 lyren]$
```

```
202.115.28.198 - SecureCRT
File Edit View Options Transfer Script Window Help
[lyren@198 lyren]$ ./server1
You get a connection from 127.0.0.1
You get a connection from 202.115.28.198

Ready ssh1: 3DES 4, 1 9 Rows, 100 Cols Linux NUM
```

- 当一个客户向接收了RST的套接口进行写操作时，内核将给该进程发一个SIGPIPE信号。该信号缺省动作是终止进程。同样，往没有建立连接的tcp套接口发送数据，也会出现上述问题。
- 我们可以做如下试验：
 - 服务器accept后立即返回；
 - 客户连接成功后，发送两次数据，中间睡眠一小段时间
- 或者，创建tcp套接字，但并不执行connect操作就发送报文，也会产生SIGPIPE信号。

■ 实现UDP套接字基本步骤分为服务器端和客户端两部分：

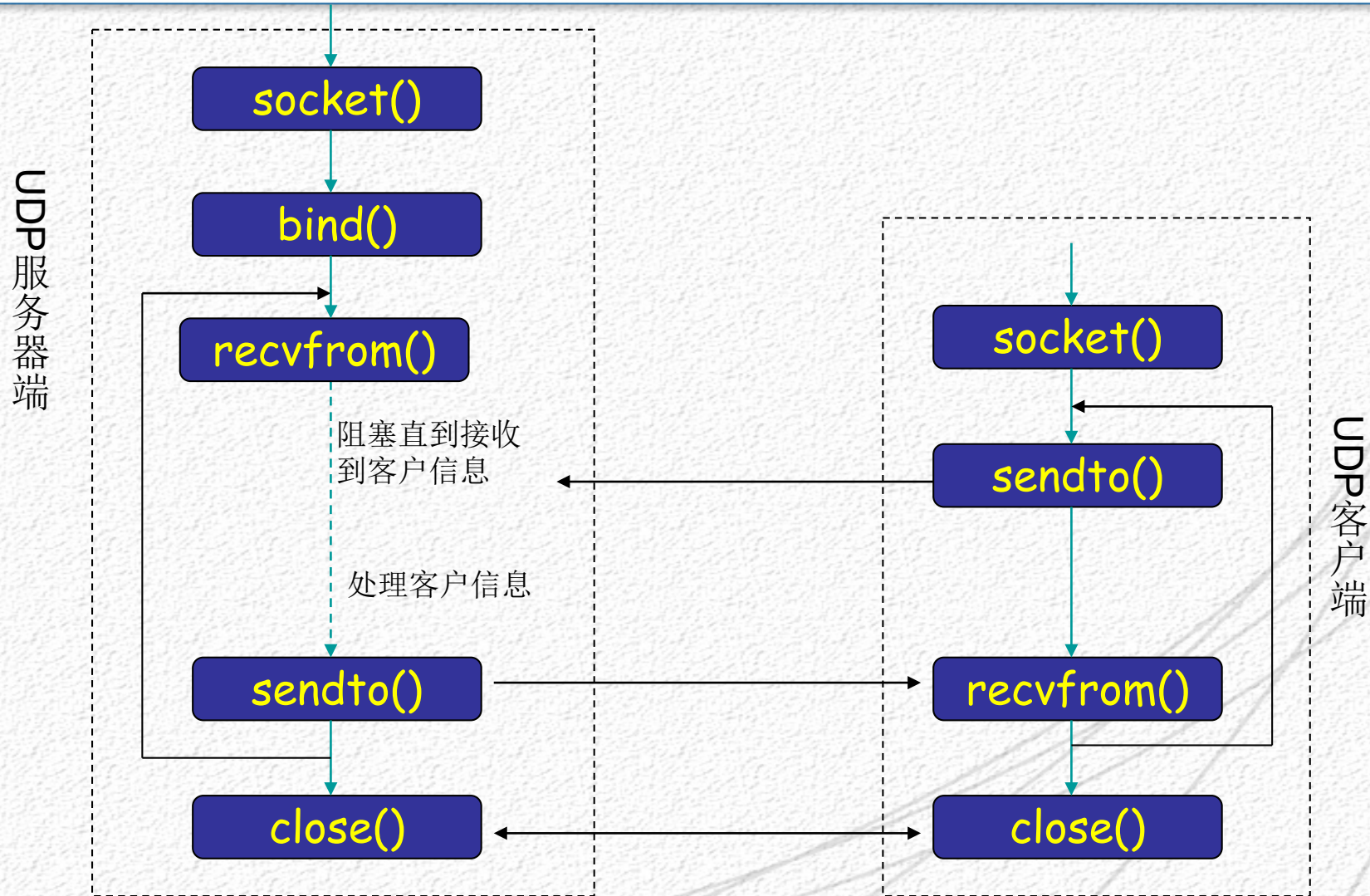
■ 服务器端

- ① 建立**UDP**套接字；
- ② 绑定套接字到特定地址；
- ③ 等待并接收客户端信息；
- ④ 处理客户端请求；
- ⑤ 发送信息回客户端；
- ⑥ 关闭套接字；

■ 客户端步骤

- ① 建立**UDP**套接字;
- ② 发送信息给服务器;
- ③ 接收来自服务器的信息;
- ④ 关闭套接字

UDP套接字编程 (Cont.)



```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
ssize_t sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, int  
tolen);
```

返回：大于0—成功发送数据长度；-1—出错；

- UDP套接字使用无连接协议，因此必须使用sendto函数，指明目的地址；
- flags是传输控制标志，其值定义如下：
 - 0：常规操作，如同write()函数；
 - MSG_OOB：发送带外数据；
 - MSG_DONTROUTE：忽略底层路由协议，直接发送。

UDP数据传输函数—recvfrom

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, int *fromlen);
```

返回：大于0—成功接收数据长度；-1—出错；

- UDP套接字使用无连接协议，因此必须使用recvfrom函数，指明源地址；
- flags是传输控制标志，其值定义如下：
 - 0：常规操作，如同read()函数；
 - MSG_PEEK：只察看数据而不读出数据；
 - MSG_OOB：忽略常规数据，而只读取带外数据；
- from 和 fromlen 是“值—结果”参数。

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main(void)
{
    int sockfd;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Create socket failed.");
        exit(1);
    }
    /* Bind socket to address */
    .....
    loop {
        /* receive and process data from client */
        .....
        /* send results to client */
    }
    close(sockfd);
}
```



```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main(void)
{
    int sockfd;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Create socket failed.");
        exit(1);
    }
    /* send data to the server */
    .....
    /* receive data from the server */
    .....
    close(sockfd);
}
```

- 本例程分为服务器和客户两部分，主要完成如下功能：
 - 服务器循环接收客户发来的消息，并显示客户IP地址和相应消息；
 - 如果服务器收到”quit“，则退出循环，并关闭套接字；
 - 客户向服务器发送消息，并接收服务器响应，显示该消息，并关闭套接字。



```
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define PORT 1234
#define MAXDATASIZE 100
int main(void)
{
    int                sockfd;
    struct sockaddr_in server, client;
    int                sin_size, num;
    char                msg[MAXDATASIZE];
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("Create socket failed.");
        exit(1);
    }
}
```

```
bzero(&server, sizeof(server));  
server.sin_family = AF_INET;  
server.sin_port = htons(PORT);  
server.sin_addr.s_addr = htonl(INADDR_ANY);  
if (bind(sockfd, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1) {  
    perror("Bind error.");  
    exit(1);  
}
```

```
sin_size = sizeof(struct sockaddr_in);  
while(1) {  
    num = recvfrom(sockfd, msg, MAXDATASIZE, 0, (struct sockaddr  
    *)&client, &sin_size);  
    if (num < 0) {  
        perror("recvfrom error.");  
    }
```



```
        exit(1);
    }
    msg[num] = '\0';
    printf("You got a message (%s) from %s\n", msg,
        inet_ntoa(client.sin_addr));
    sendto(sockfd, "Welcom.", 8, 0, (struct sockaddr *)&client, sin_size);
    if (!strcmp(msg, "quit")) break;
}
close(sockfd);
}
```



```
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define PORT          1234
#define MAXDATASIZE  100

int main(int argc, char *argv[])
{
    int fd, numbytes;
    char      buf[MAXDATASIZE];
    struct hostent *    he;
    struct sockaddr_in server, reply;
    if (argc != 3)
    {
        printf("Usage: %s <IP address> <Message>\n", argv[0]);
        exit(-1);
    }
}
```



```
if ((he = gethostbyname(argv[1])) == NULL) {  
    perror("gethostbyname error.");  
    exit(1);  
}  
if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {  
    perror("Create socket failed.");  
    exit(1);  
}  
bzero(&server, sizeof(server));  
server.sin_family = AF_INET;  
server.sin_port = htons(PORT);  
server.sin_addr = *((struct in_addr *) he->h_addr);  
sendto(fd, argv[2], strlen(argv[2]), 0, (struct sockaddr *)&server, sizeof(struct  
sockaddr));  
while(1) {  
    int len;  
    numbytes = recvfrom(fd, buf, MAXDATASIZE, 0, (struct sockaddr  
*)reply, &len);
```

```
    if (numbytes == -1) {  
        perror("recvfrom error.");  
        exit(1);  
    }  
    if (len != sizeof(struct sockaddr) || memcmp((const void *)&server,  
void *)&reply, len) != 0) {                                     (const  
        printf("receive message from other server.\n");  
        continue;  
    }  
    buf[numbytes] = '\0';  
    printf("Server Message: %s\n",buf);  
    break;  
} /* while(1) */  
close(fd);  
}
```



```
202.115.28.198 - SecureCRT
File Edit View Options Transfer Script Window Help
[lyren@198 lyren]$ ./server2
You got a message (iam198) from 202.115.28.198
You got a message (iam143) from 202.115.14.143
```

```
202.115.28.198 - SecureCRT
File Edit View Options Transfer Script Window Help
[lyren@198 lyren]$ ./client2 202.115.28.198 iam198
Server Message: Welcom.
[lyren@198 lyren]$
[lyren@198 lyren]$
```

```
202.115.14.143 - SecureCRT
File Edit View Options Transfer Script Window Help
[lyren@tomcat lyren]$ ./client2 202.115.28.198 iam143
Server Message: Welcom.
[lyren@tomcat lyren]$
[lyren@tomcat lyren]$
```

■ 客户端：

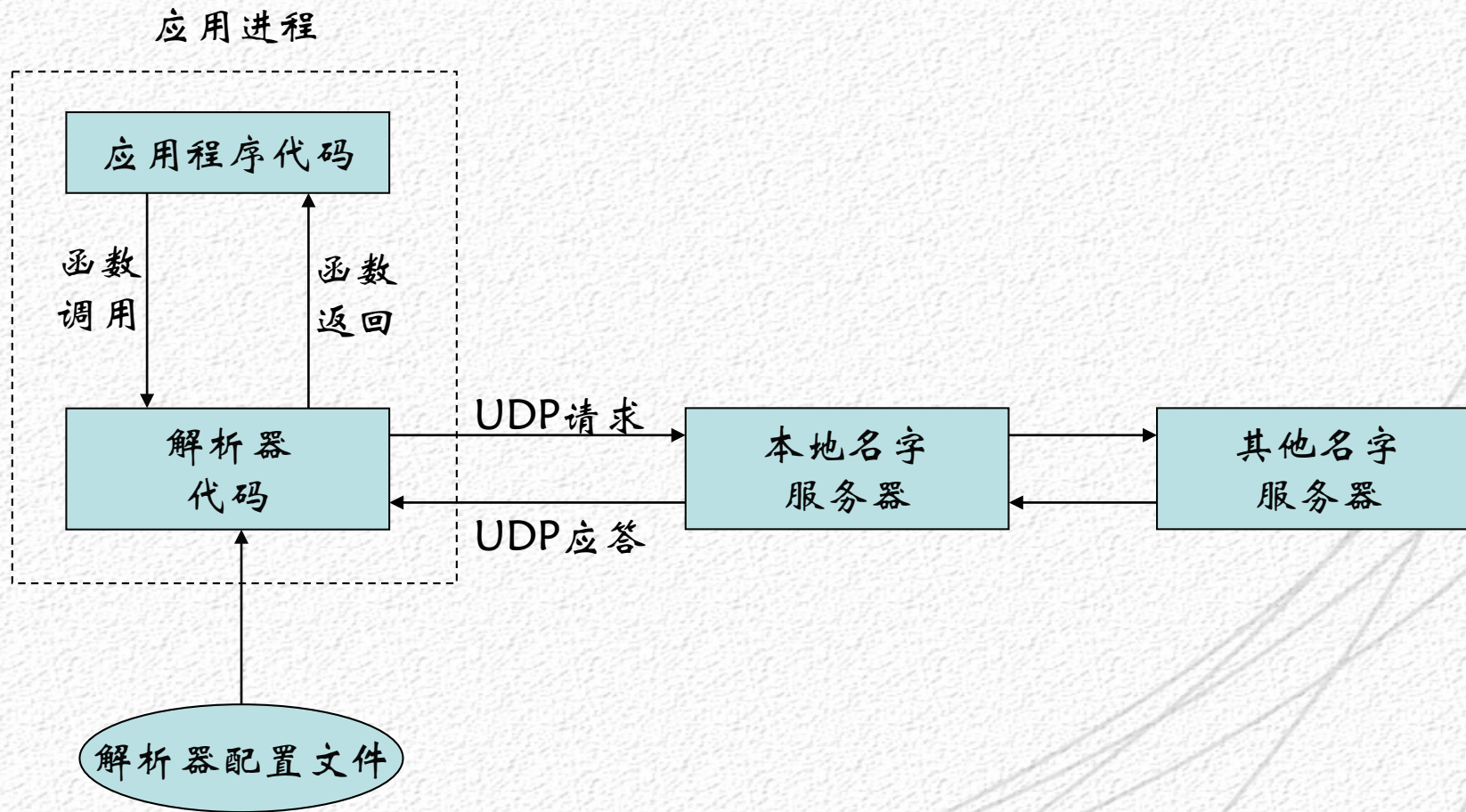
- 从命令行读入服务器的**IP**地址；并连接到服务器；
- 循环从命令行读入一行字符串，并传递给服务器，由服务器对字符串反转，并将结果返回客户程序；
- 客户程序显示反转后的字符串；

■ 服务器端：

- 接收客户的连接请求，并显示客户的**IP**地址和端口号；
- 接收客户传来的字符串，反转后传递给客户；

■ DNS中的条目称为资源记录，常用的有以下几类：

- **A**：记录将主机名映射为**32位**的**IPv4**地址；
- **AAAA**：记录将主机名映射成**128**的**IPv6**地址；
- **MX**：指定一主机作为邮件交换器；
- **CNAME**：指定服务器的规范名字；




```
#include <netdb.h>
```

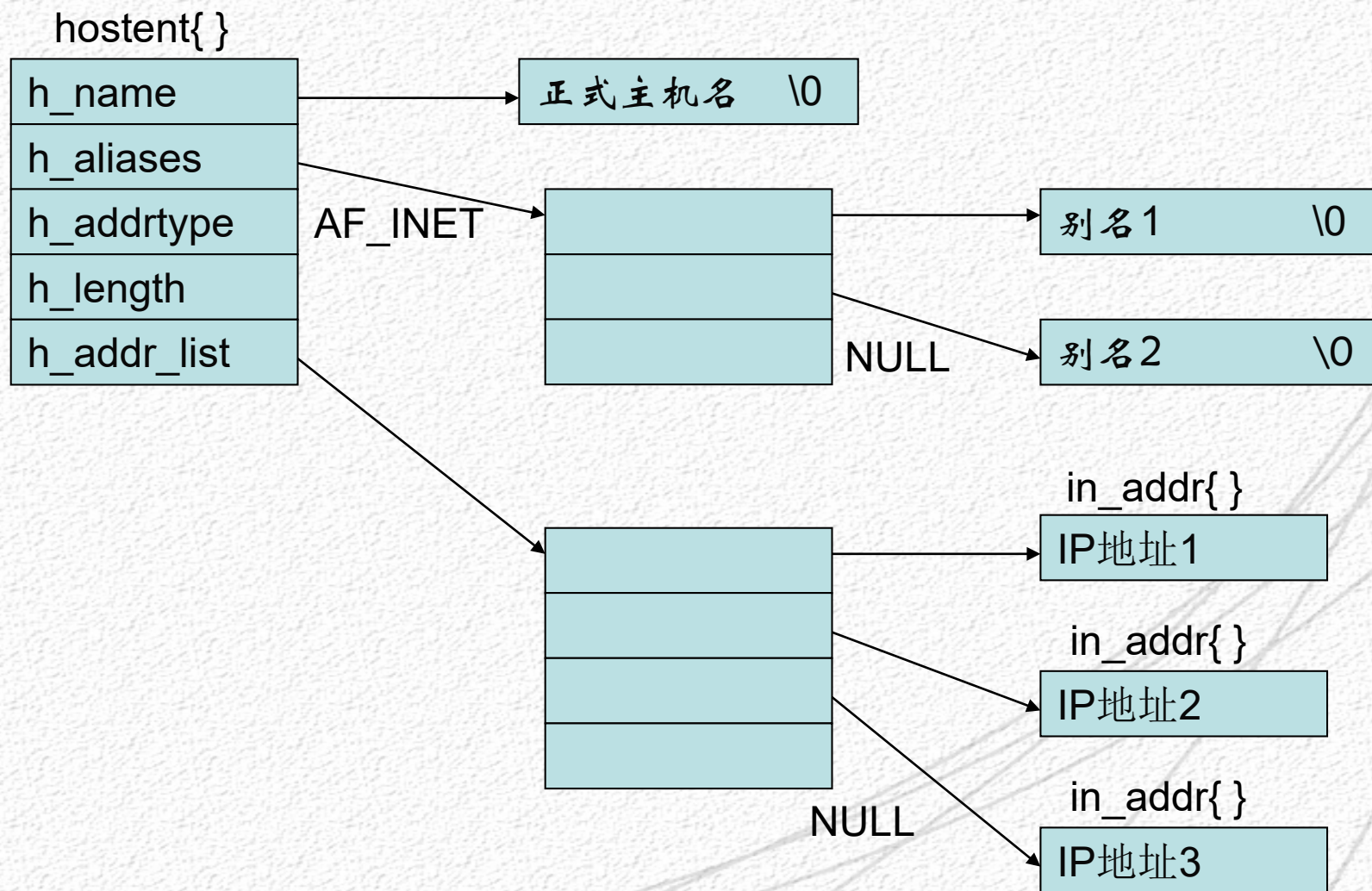
```
struct hostent *gethostbyname(const char *hostname)
```

返回：非空指针—成功；空指针—出错，同时设置h_error

- 该函数既可解析IPv4地址，也可解析IPv6地址；
- 该函数既可接收域名，也可接收点分十进制参数
- 当hostname为点分十进制时，函数并不执行网络查询，而是直接将其拷贝到结果字段中。
- 此函数返回的非空指针指向下面的hostent结构

```
struct hostent {  
    char *h_name;           /* official (canonical) name of host */  
    char **h_aliases;       /* pointer to array of pointers to alias names */  
    int  h_addrtype;        /* host address type: AF_INET or AF_INET6 */  
    int  h_length;          /* length of address :4 or 16 */  
    char **h_addr_list;     /* ptr to array of ptrs with IPv4 or IPv6 address */  
};  
  
#define h_addr h_addr_list[0] /* first address in list */
```


gethostbyname 返回的信息



```
#include <netdb.h>
```

```
struct hostent *gethostbyaddr(const char *addr, size_t len, int family);
```

返回：非空指针—成功；空指针—出错，同时设置h_error

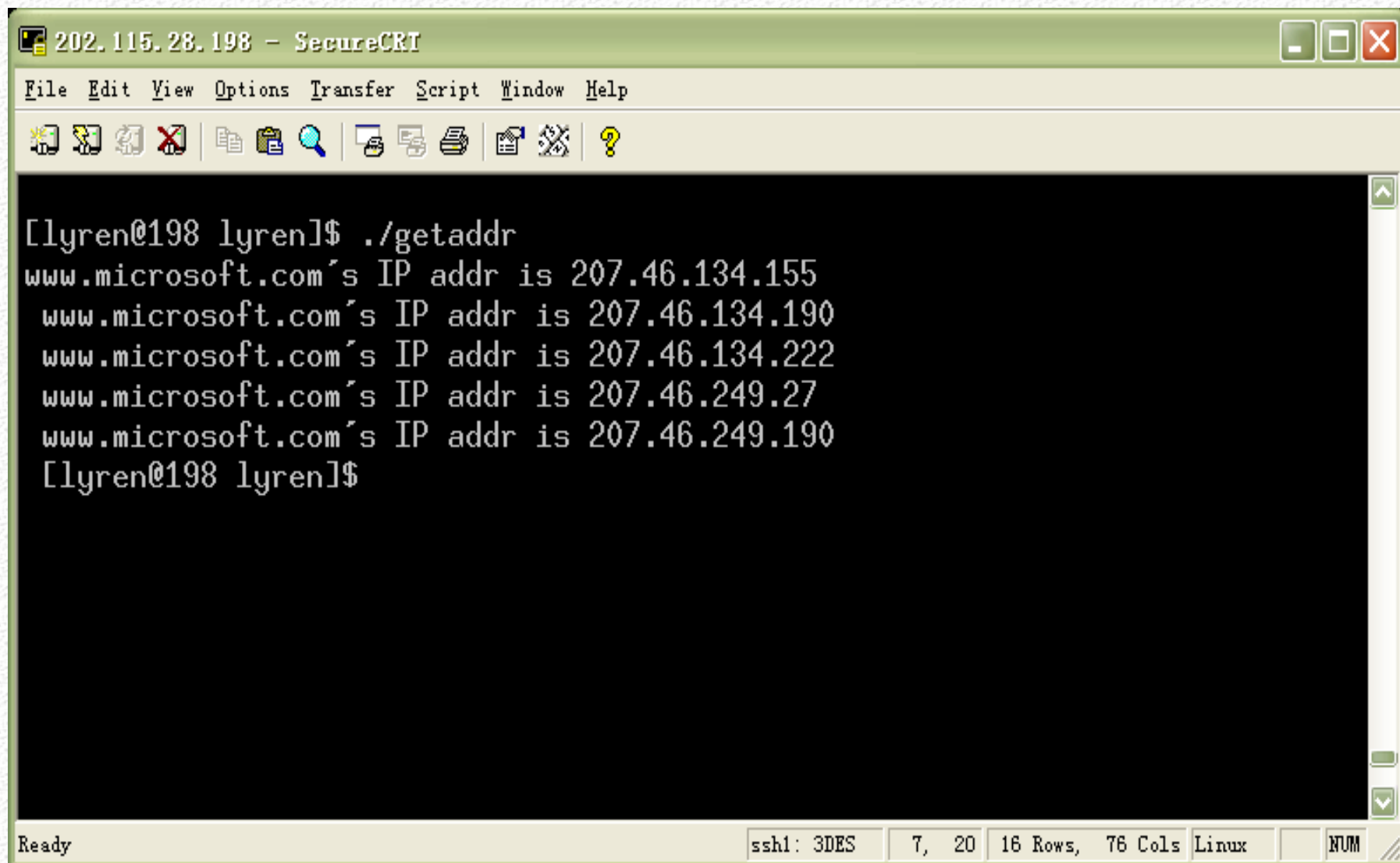
- 该函数取一个二进制的IP地址，并试图找到相应的主机名；
- 参数addr不是char *类型，而实际上是一个真正指向IPv4或IPv6地址结构in_addr或in6_addr，因此当输入是点分十进制串时，需先调用inet_aton或inet_pton函数


```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main(int argc, char *argv[])
{
    struct sockaddr_in    addr;
    struct hostent        *he;
    char                  **alias;
    if(argc<2) {
        perror("Usage :%s name | IP",argv[0]);
        exit(1);
    }
}
```

```
argv++;  
for(; *argv != NULL; argv++) {  
    if (inet_aton( *argv, &addr.sin_addr) != 0) {  
        he = gethostbyaddr((char *)&addr.sin_addr, 4, AF_INET);  
        printf("address information of IP %s:\n", *argv);  
    } else {  
        he = gethostbyname(*argv);  
        printf("address information of host %s:\n", *argv);  
    }  
    if (he == NULL) {  
        fprintf(stderr, "no address information of %s\n", *argv);  
        continue;  
    }  
}
```



```
printf("Official host name: %s\n", he->h_name);  
printf("name aliases:");  
for(alias = he->h_aliases; *alias != NULL; alias ++)  
    printf("%s, ", *alias);  
printf("\nIP addresses: ");  
for (alias = he->h_addr_list; *alias != NULL; alias++)  
    printf("%s", inet_ntoa( *(struct in_addr *)(* alias)));\br/>}
```



202.115.28.198 - SecureCRT

File Edit View Options Transfer Script Window Help

[lyren@198 lyren]\$./getaddr
www.microsoft.com's IP addr is 207.46.134.155
www.microsoft.com's IP addr is 207.46.134.190
www.microsoft.com's IP addr is 207.46.134.222
www.microsoft.com's IP addr is 207.46.249.27
www.microsoft.com's IP addr is 207.46.249.190
[lyren@198 lyren]\$

Ready ssh1: 3DES 7, 20 16 Rows, 76 Cols Linux NUM


```
#include <sys/utsname.h>
```

```
int uname(struct utsname *name);
```

返回：非负值—成功；-1—出错。

- 该函数返回本机的名字，它不是解析器库中的一部分，但它经常与gethostbyname函数一起用来确定本地主机IP地址；

- 函数的参数是“值—结果”参数，其类型如下：

```
#define UTS_NAMESIZE 16
```

```
#define UTS_NODESIZE 256
```

```
struct utsname {
```

```
    char sysname[UTS_NAMESIZE];           /* name of this operating system */
```

```
    char nodename[UTS_NODESIZE];          /* name of this node */
```

```
    char release[UTS_NAMESIZE];           /* O.S release level */
```

```
    char version[UTS_NAMESIZE];           /* O.S version level */
```

```
    char machine[UTS_NAMESIZE];           /* hardware type */
```

```
}
```

```
#include <netdb.h>
#include <sys/utsname.h>
#include <netinet/in.h>
#include <stdio.h>

int main(void)
{
    struct hostent *he;
    struct utsname myname;
    struct in_addr addr;
    char temp[100];

    if(uname(&myname) < 0)
        return(0);
```




```
if (( he = gethostbyname(myname.nodename)) == NULL)
{
    perror("error.");
    exit(1);
}
printf("OS name is %s\n",myname.sysname);
printf("node name is %s\n",myname.nodename);
printf("OS version level is %s\n",myname.version);
printf("hardware type is %s\n",myname.machine);

memcpy(&addr, he->h_addr, sizeof(addr));
printf("host IP is %s\n", inet_ntoa(addr));
return(0);
}
```



```
202.115.28.198 - SecureCRT
File Edit View Options Transfer Script Window Help
[lyren@198 lyren]$ ./gethostip
OS name is Linux
node name is 198
OS version level is #1 Thu Sep 6 17:27:27 EDT 2001
hardware type is i686
host IP is 0.0.0.198
[lyren@198 lyren]$
```

Ready ssh1: 3DES 7, 20 22 Rows, 76 Cols Linux NUM


```
#include <unistd.h>
```

```
int gethostname(char *name, size_t namelen);
```

返回：0—成功；-1—出错

- 该函数也返回当前主机的名字；
- name指向主机名存储位置的指针；

```
#include <netdb.h>
```

```
struct servent *getservbyname(const char *servname, const char *protoname);
```

返回：非空指针—成功；空指针—出错。

- 该函数根据服务器提供的服务名字获取服务器的有关信息；
- 参数servname指向服务的名字，如” domain”, ”ftp”, ”www”；
- 函数返回的结构如下：

```
struct servent {  
    char *s_name;           /* official service name */  
    char **s_aliases;       /* alias list */  
    int s_port;             /* port number, net-byte order */  
    char *s_proto;          /* protocol to use */  
}
```

如： struct servent *ptr;

```
sptr = getservbyname(“FTP”, “TCP”);
```



```
#include <netdb.h>
```

```
struct servent *getservbyport(int port, const char *protoname);
```

返回：非空指针—成功；空指针—出错。

- 该函数根据端口号和可选的协议查找相应的服务；
- 参数port必须为网络字节序；

如：

```
struct servent *sptr;
```

```
sptr = getservbyport(htons(53), "UDP");
```

```
/*DNS using UDP */
```

```
sptr = getservbyport(htons(21), "TCP");
```

```
/* FTP using TCP */
```

```
sptr = getservbyport(htons(21), NULL);
```

```
/* FTP using TCP */
```

```
sptr = getservbyport(htons(21), "UDP");
```

```
/* this call will fail */
```

```
#include <sys/socket.h>
```

```
int getsockname(int sockfd, struct sockaddr *localaddr, socklen_t *len);
```

```
int getpeername(int sockfd, struct sockaddr *peeraddr, socklen_t *len);
```

均返回：0—成功；-1—出错

- **getsockname**函数返回套接字的本地地址；而后者返回套接字对应的远程地址；
- 一般在以下三中情况下调用这两个函数：
 - 客户机在**connect**后，用**getsockname**获取系统选择的ip和端口号；
 - 以**INADDR_ANY**调用**bind**的服务器，在接收到连接后（即**accept**后），调用**getsockname**获取系统选择的IP地址；
 - 服务器获得连接后，并**exec**真正的程序后。只能用**getpeername**获取客户套接字地址；