Socket编程

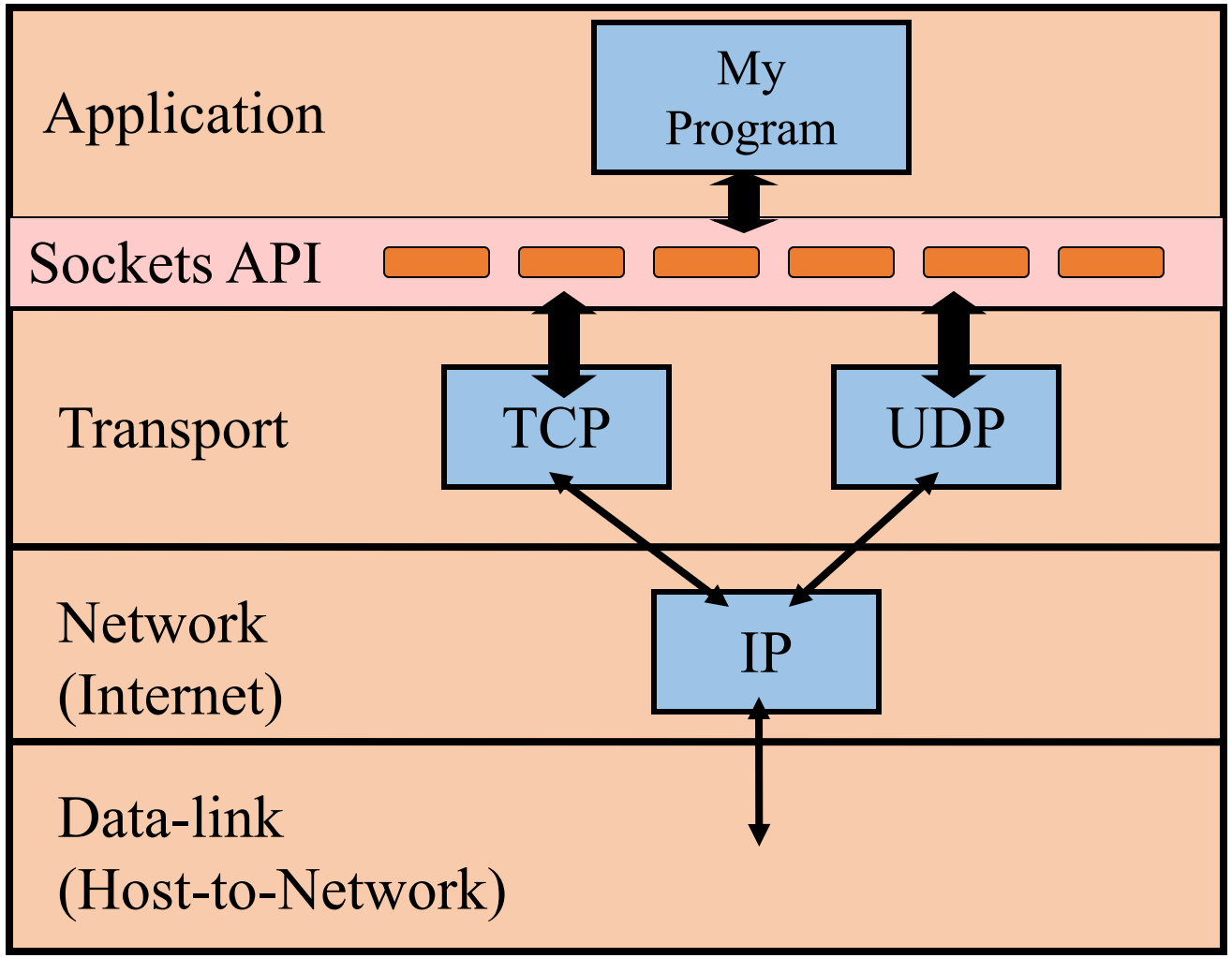
Socket编程

- Socket简介
- Windows Socket
  - Winsock1.1基本API
  - Winsock2.0

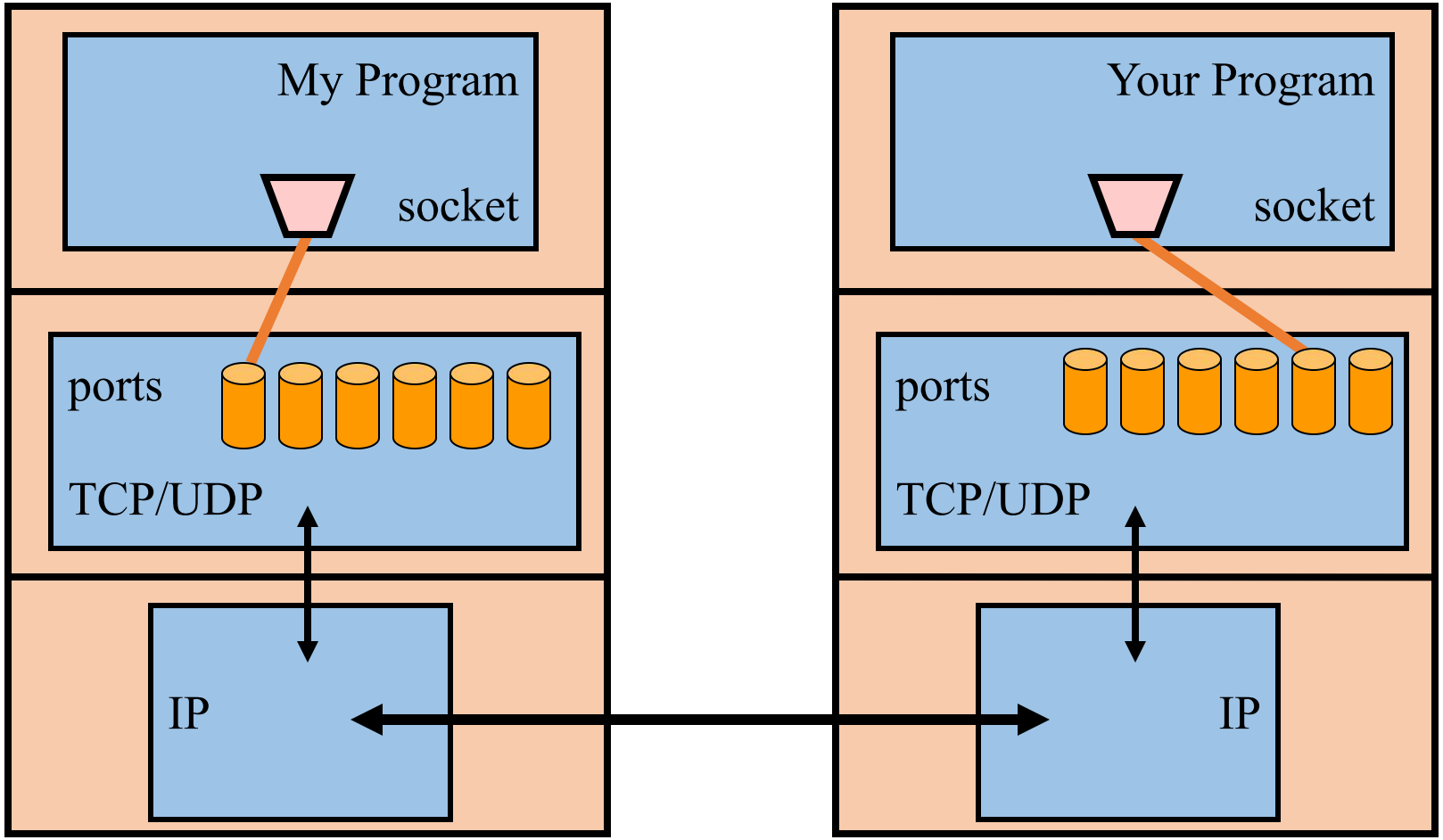
什么是socket?

- 应用程序与网络之间的接口
  - 应用程序创建socket
  - socket *类型* 决定了通信的类型
    - 可靠的 vs. 尽最大努力的
    - 面向连接的 vs. 无连接的
- 一旦socket配置完成，应用程序就可以
  - 把数据传给socket，从而进行网络传输
  - 从socket接收数据(其他主机通过网络发送过来的)
- Socket在计算机中提供了一个通信接口，可以通过这个接口与任何一个具有Socket接口的计算机通信。应用程序在网络上传输，接收的信息都通过这个Socket接口来实现。

Socket在协议栈中的位置



Socket到Socket的通信



Windows Socket(1)

- Windows Socket 是从 Berkeley Socket扩展而来的，其在继承Berkeley Socket的基础上，又进行了新的扩充。这些扩充主要是提供了一些异步函数，并增加了符合Windows消息驱动特性的网络事件异步选择机制
- Windows socket的版本
  - Winsock1.1
  - Winsock2.0

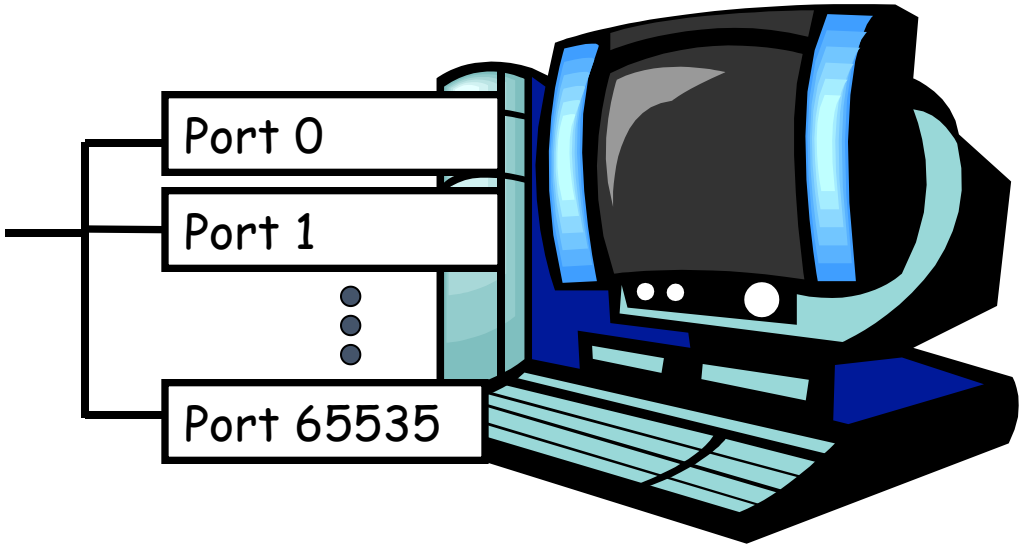
Windows Socket(2)

- Winsock的实现方式
  - Winsock: C-Based API, 与Unix C API类似
  - CAsyncSocket: 对Winsock API的简单C++封装
  - CSocket: 对Socket的高层抽象 (自动处理字节顺序转换等)
    - 对象串行化
      - CSocketFile
      - CArchive

Windows Socket编程原理

端口

- 每个主机有65,536个端口
- 一些端口被预留，用于特定的应用程序
  - 20,21: FTP
  - 23: Telnet
  - 80: HTTP
  - 参考RFC 1700 (大约1024个端口被预留)



- **Socket提供了一个通过端口在网络上收发数据的接口**

IP地址、端口和Socket

- 类似公寓和邮箱
  - 你是应用程序
  - 你的公寓地址是IP地址
  - 你的邮箱是端口
  - 邮局是网络
  - Socket是使你能够使用邮箱的钥匙(假设发出去的信是由你放入邮箱的)

- Q: 如何为Socket选择要连接的端口?

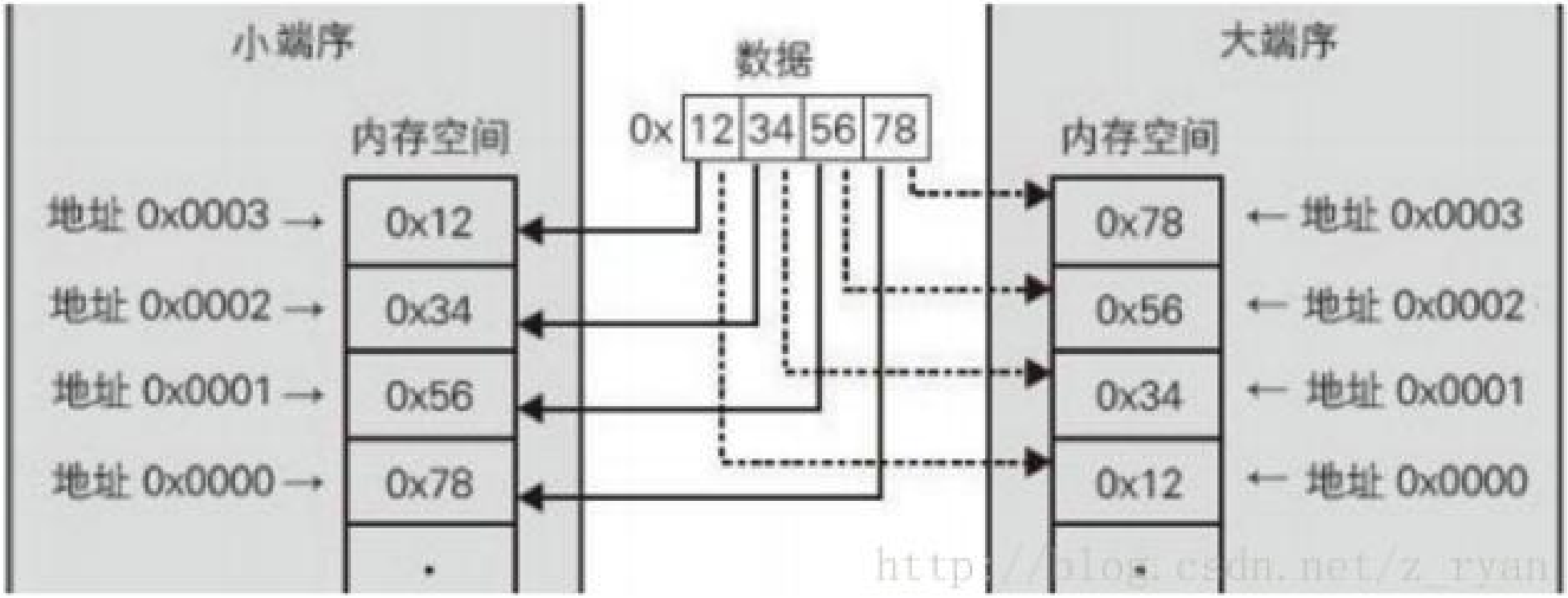
Internet地址数据结构

```
#include <winsock.h>

struct sockaddr_in
{
    short sin_family;//AF_INET
    unsigned short sin_port;    //16位端口号，网络字节顺序
    struct in_addr sin_addr;    //32位IP地址，网络字节顺序
    char sin_zero[8];//保留
};
```

- sin_family = AF_INET 选择Internet地址族

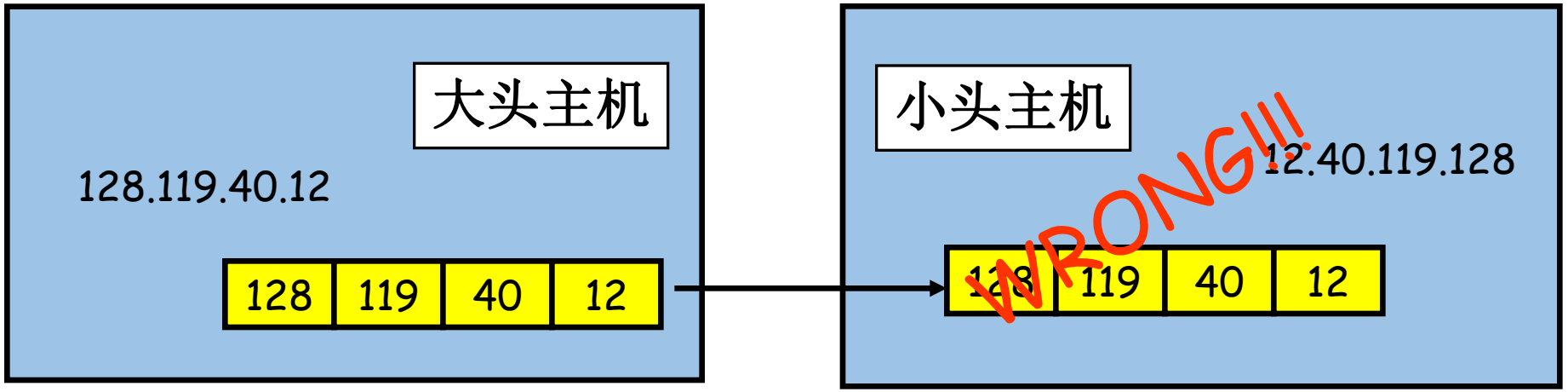
字节顺序

- 两种顺序
  - 大头字节序：最高有效位存于最低内存地址处，最低有效位存于最高内存处
  - 小头字节序：最高有效位存于最高内存地址，最低有效位存于最低内存处



字节顺序

- 问题：
  - 不同的机器使用不同的字节顺序
    - 小头：低字节优先 i386，alpha，...
    - 大头：高字节优先 Sun Solaris，PowerPC，...
  - 使用不同字节顺序的主机如何通过网络通信



解决方法:网络字节顺序

- 定义:
  - 主机字节顺序: 主机使用的字节顺序 (大头或小头)
  - 网络字节顺序：网络使用的字节顺序–大头
- 通过网络传输的任何信息都应该转化成网络字节顺序，接收后再转回主机字节顺序
- Q: Socket是否自动完成字节顺序转换？
- Q: 大头主机不需要转换字节顺序，而小头主机需要，如何避免写两个版本的代码？

字节顺序转换函数

- 在主机字节顺序和网络字节顺序之间转换
  - ‘h’ = 主机字节顺序
  - ‘n’ = 网络字节顺序
  - ‘l’ = long (4 bytes), 转换IP地址
  - ‘s’ = short (2 bytes), 转换端口号

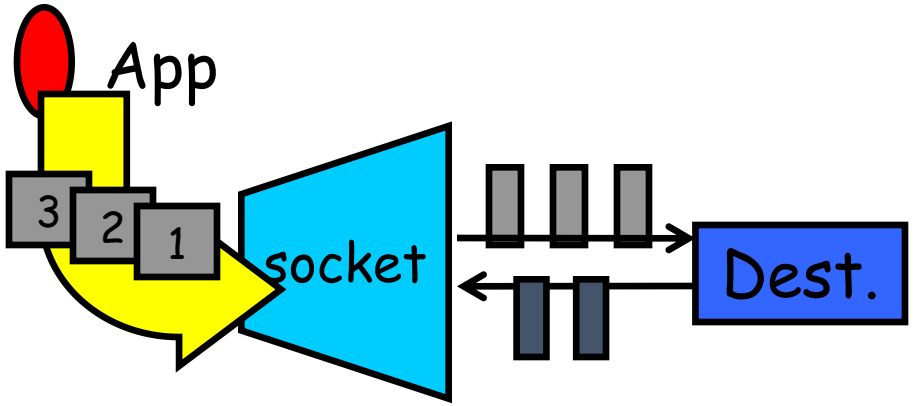
```
#include <winsock.h>
 u_long PASCAL FAR htonl( u_long hostlong);
u_short PASCAL FAR htons( u_short hostshort);
u_long PASCAL FAR ntohl( u_long netlong);
u_short PASCAL FAR ntohs( u_short netshort);
```

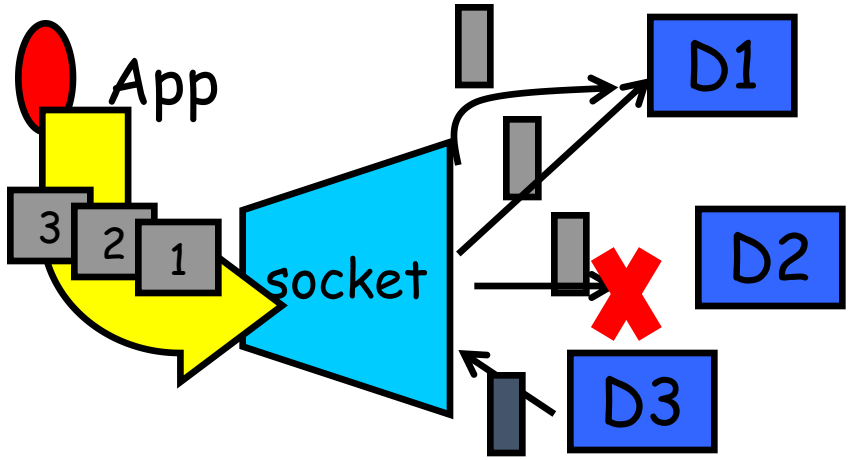
- 在大头主机上，上述函数不作任何转换
- 在小头主机上，上述函数把字节顺序逆序

Socket的两种基本类型

- SOCK_STREAM
  - TCP
  - 可靠传输
  - 保证顺序
  - 面向连接
  - 双向

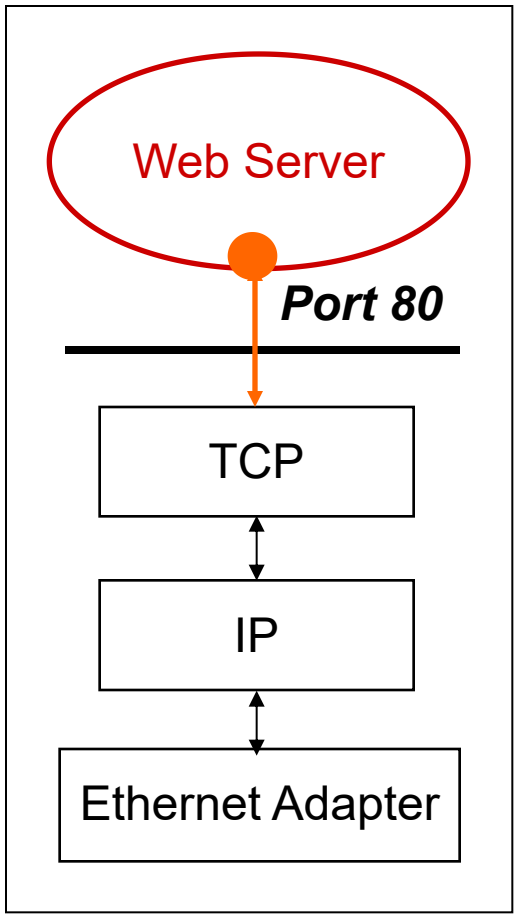
- SOCK_DGRAM
  - UDP
  - 不可靠传输
  - 无顺序保证
  - 无“连接”概念 – 应用程序为每个包指定目的地
  - 可以发送或接收





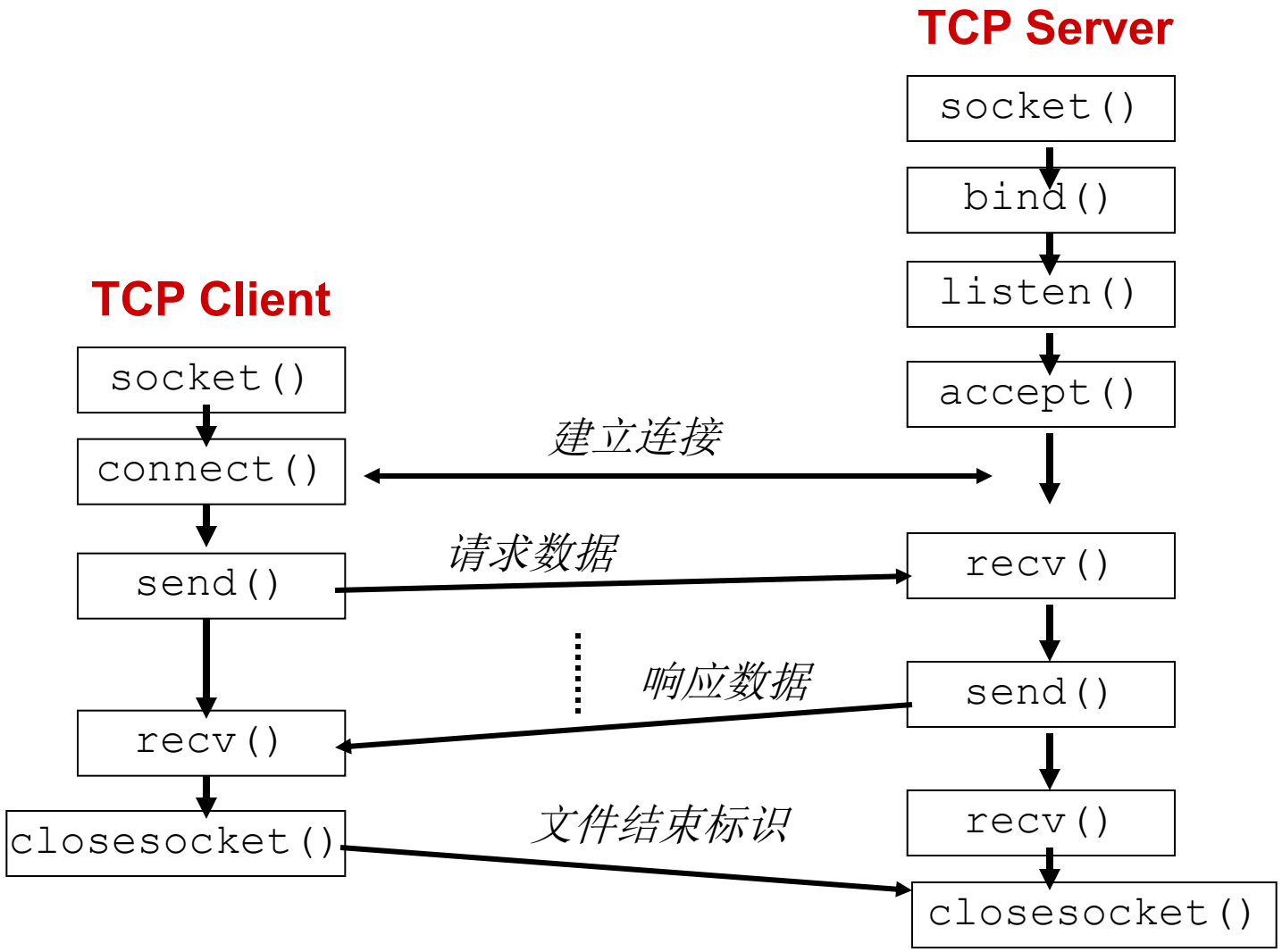
Q: 为什么需要 SOCK_DGRAM类型?

TCP Server



- 举例: web server
- *web client怎么去连接web server？*
- *为了接收来自web client的连接请求，web server应做什么准备?*

TCP Client-Server交互流程



Winsock的初始化

- 每个Winsock应用都必须加载Winsock DLL的相应版本

```
#include <winsock.h>
int PASCAL FAR WSAStartup ( WORD
wVersionRequested, LPWSADATA lpWSAData );
```

wVersionRequested：Windows Socket API提供的调用方可使用的最高版本号.高位字节指出副版本(修正)号,低位字节指明主版本号.

lpWSAData：指向WSADATA数据结构的指针,用来接收Windows Socket实现的细节.

```
WSADATA  wsd;
if(WSAStartup(MAKEWORD(1,1),&wsd)!=0)
{
      return FALSE;
}
```

Socket I/O: socket()

- Web使用TCP, web server需要创建一个SOCK_STREAM套接字

```
#include <winsock.h>
SOCKET PASCAL FAR socket( int af, int type, int
protocol);
```

af：一个地址格式描述。

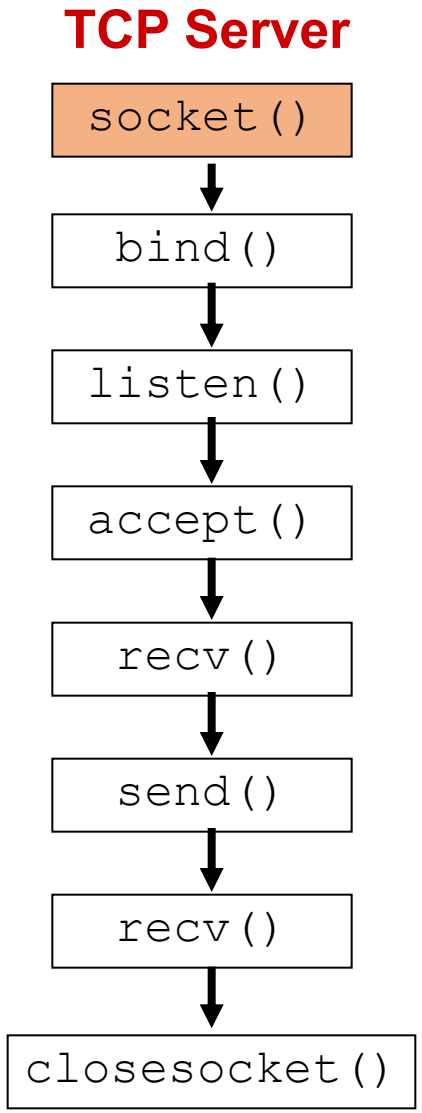
type：新套接字的类型描述。

protocol：套接字所用的协议。如调用者不想指定，可用0。

```
SOCKET m_hSocket;
m_hSocket=socket(AF_INET,SOCK_STREAM,0);
```

AF_INET 把socket与Internet协议族相关联

SOCK_STREAM 选择 TCP协议



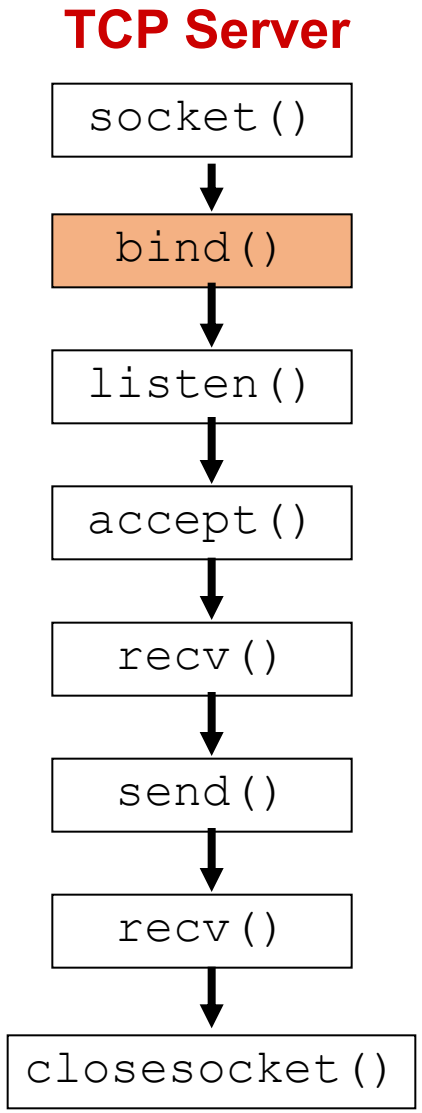
Socket I/O: bind()

- 把*socket* 绑定到一个特定端口

```
int PASCAL FAR bind(SOCKET s, const struct
sockaddr FAR * name, int namelen);
```

s：标识一未捆绑套接字的描述字。
name：赋予套接字的地址。
返回值：没有错误，bind()返回0，否则返回SOCKET_ERROR

```
sockaddr_in m_addr;
m_addr.sin_family = AF_INET;
m_addr.sin_addr.S_un.S_addr =htonl(INADDR_ANY);
m_addr.sin_port = htons(80);
      int ret = 0;int error = 0;
ret = bind(m_hSocket, (LPSOCKADDR)&m_addr,
sizeof(m_addr));
if(ret == SOCKET_ERROR)
{     AfxMessageBox("Binding Error"); //绑定错误
      return FALSE;
}
```



Socket I/O: listen()

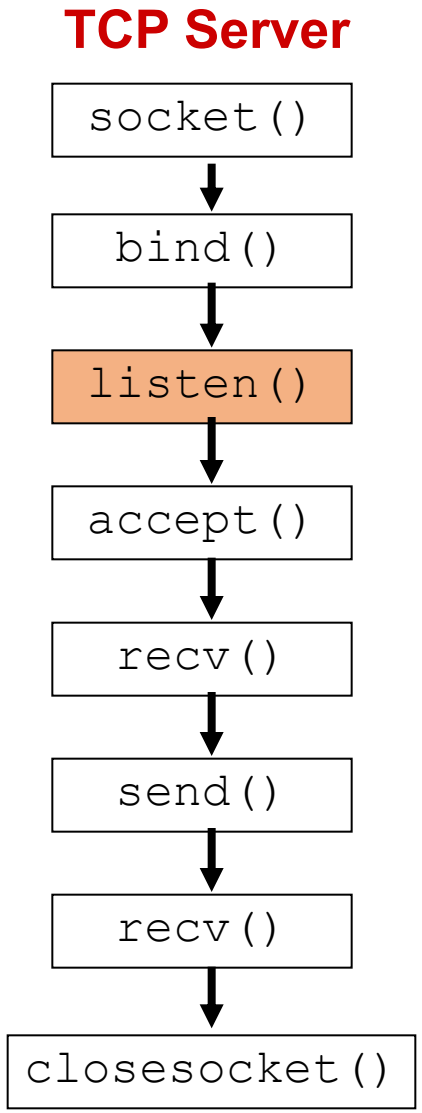
- *listen* 表示 server准备接收连接请求

```
#include <winsock.h>
int PASCAL FAR listen( SOCKET s, int
backlog);
```

S：用于标识一个已捆绑未连接套接字的描述字。

backlog：等待连接队列的最大长度。

```
ret = listen(m_hSocket, 5);
          //第二个参数表示最多支持的客户连接数
if(ret == SOCKET_ERROR)
{    //listen失败
     AfxMessageBox("Listen Error");
     return FALSE;
}
```



Socket I/O: accept()

- *accept* 等待一个连接请求

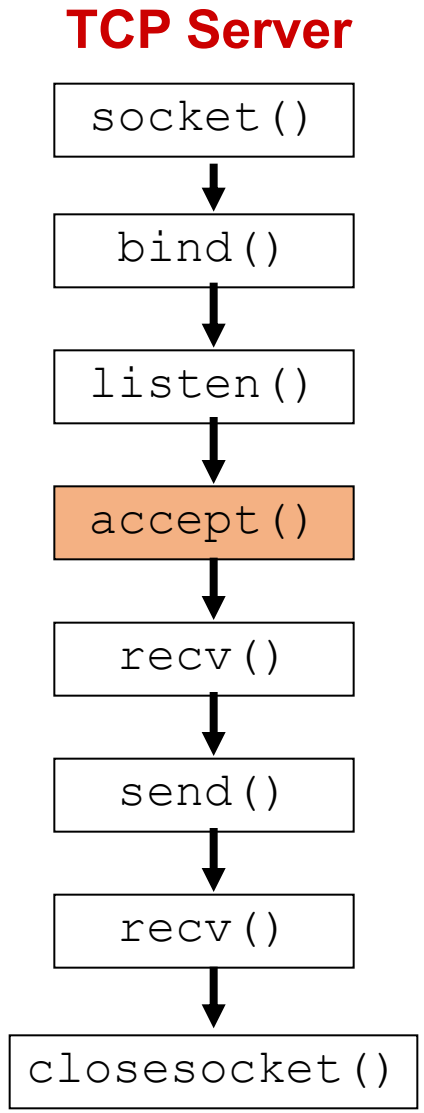
```
#include <winsock.h>
SOCKET PASCAL FAR accept( SOCKET s, struct
sockaddr FAR* addr, int FAR* addrlen);
```

s：套接字描述字，该套接字在**listen()**后监听连接。

addr：（可选）指针，指向一缓冲区，接收连接实体的地址。**Addr**参数的实际格式由套接字创建时所产生的地址族确定。

addrlen：（可选）指针，指向存有**addr**地址长度的整形数。

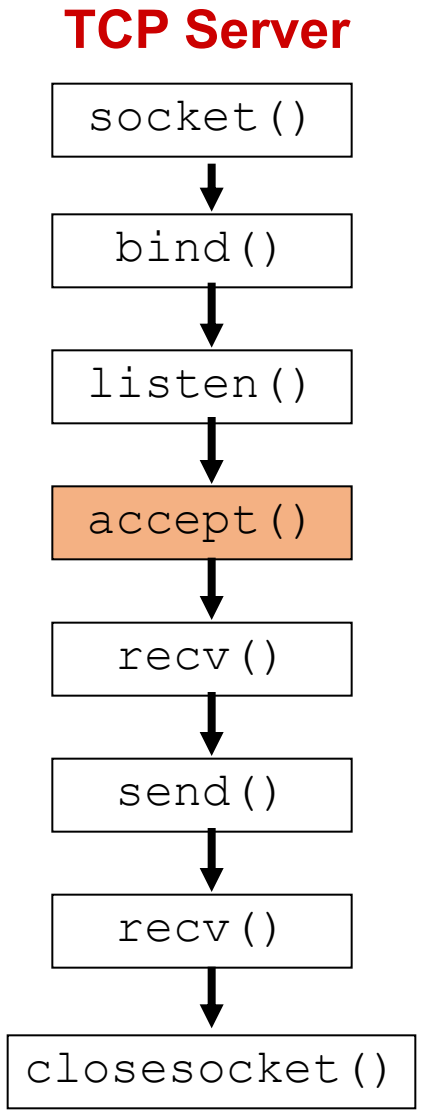
返回值：返回一个新的**socket**，其属性与**s**相同。若返回值**<0**，则表明发生了错误。



Socket I/O: accept()

```
#include <winsock.h>
SOCKET socket;
struct sockaddr cli;
int cli_len = sizeof(cli);

socket= accept(m_hSocket, &cli, &cli_len);
if(socket < 0) {
      AfxMessageBox("Listen Error");
      return FALSE;
}
```

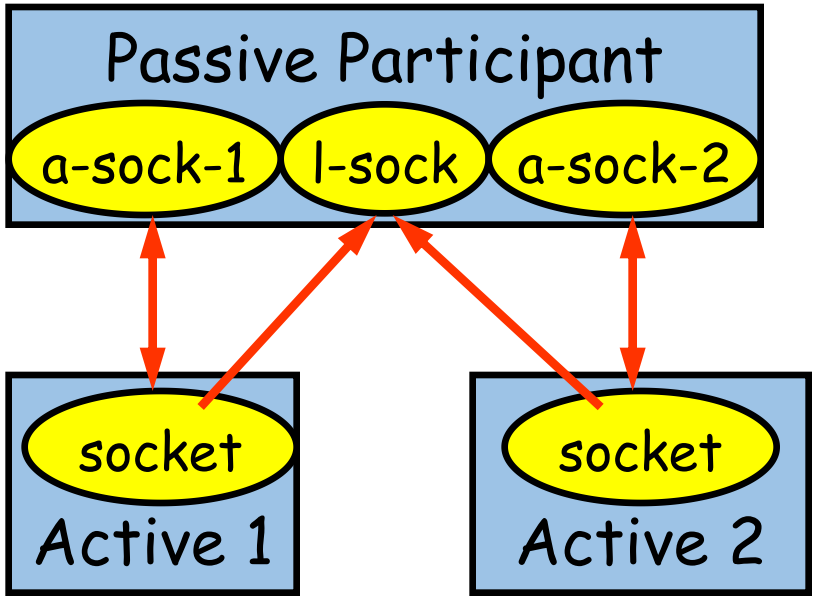
- Server如何得到Client的基本信息?
  - cli.sin_addr.s_addr 包含了client的IP地址
  - cli.sin_port 包含了Client的端口号
- *accept* 为什么要返回一个新的socket?



建立连接

- 被动参与者
  - step 1: listen (监听连接请求)
  - step 3: accept (接受连接请求)
  - step 4: 数据传输
- 被接受的连接工作在新的Socket上
- 旧Socket继续监听其它的主动参与者

- 主动参与者
  - step 2: 请求并建立连接
  - step 4: 数据传输



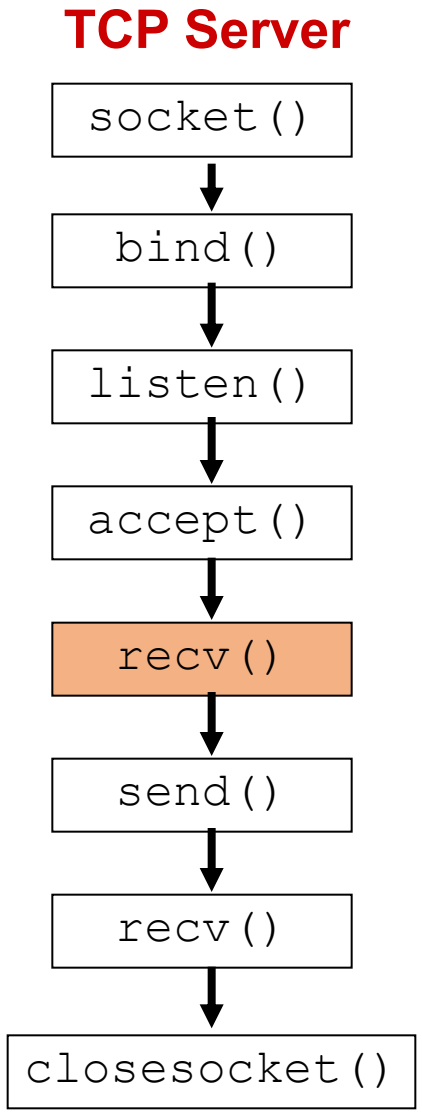
Socket I/O: recv()

- *recv* 可以从一个socket接收数据。

```
#include <winsock.h>
int PASCAL FAR recv( SOCKET s, char FAR* buf,
int len, int flags);
```

s：一个标识已连接套接字的描述字。
buf：用于接收数据的缓冲区。
len：缓冲区长度。
flags：指定调用方式。
返回值：若无错误发生，**recv()**返回读入的字节数。如果连接已中止，返回0。否则返回**SOCKET_ERROR**错误。

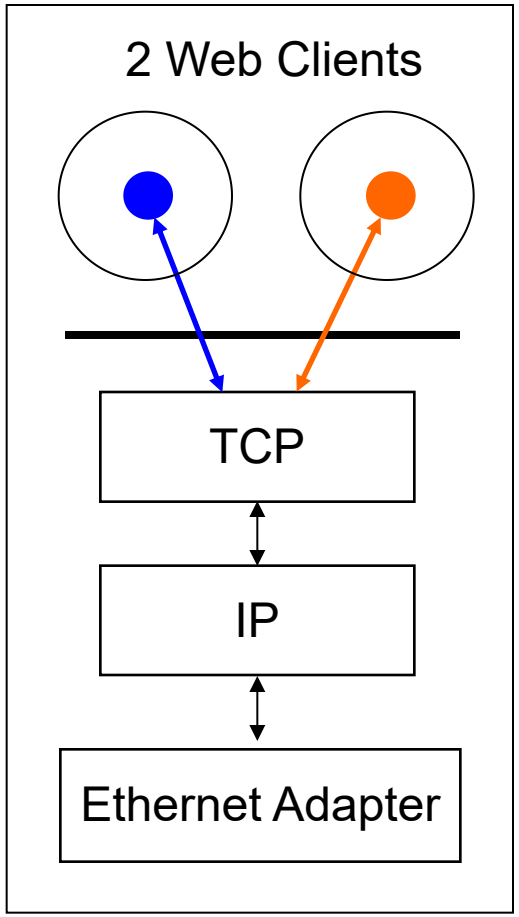
```
#include <winsock.h>
char s[1024];
int len;
len=recv(socket,s,1024,0);
```



TCP Client

- 举例: web client

- *web client* 如何连接到一个*web server*?



处理IP地址

- IP地址通常表示为字符串 （“128.192.35.50”),但程序把IP地址作为整数来处理。

将字符串地址转化成数值地址:

```
#include <winsock.h>
unsigned long PASCAL FAR inet_addr( const struct FAR* cp);
```

cp：一个以Internet标准“.”间隔的字符串。

返回值：若无错误发生，返回一个无符号长整型数。如果传入的字符串不是一个合法的Internet地址，那么返回INADDR_NONE。

```
struct sockaddr_in srv;
srv.sin_addr.s_addr = inet_addr(“128.192.35.50”);
```

将数值地址转化成字符串地址:

```
#include <winsock.h>
char FAR* PASCAL FAR inet_ntoa( struct in_addr in);
```

in：一个表示Internet主机地址的结构。

把域名转换成地址

- gethostname 获得本地主机名
- gethostbyname 为DNS提供接口，从主机名得到对应的主机地址。
- gethostbyaddr – 由网络地址得到对应的主机信息。

```
struct hostent {
   char FAR*  h_name;
    char FAR FAR** h_aliases;
   short  h_addrtype;
   short  h_length;
   char FAR FAR** h_addr_list;
};
```

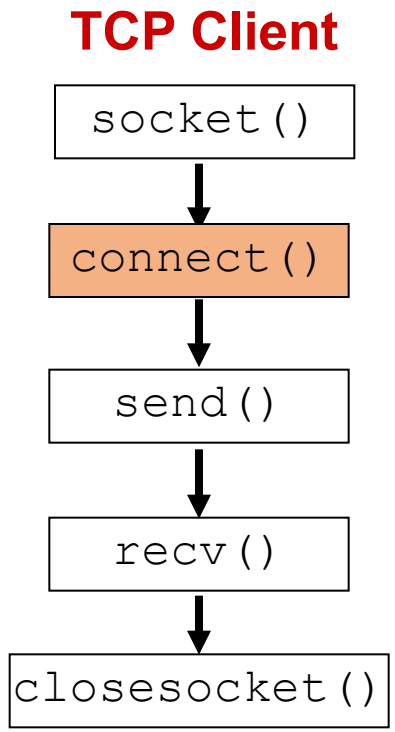
```
struct hostent *hp;
struct sockaddr_in peeraddr;
char *name = "www.cs.uga.edu";
peeraddr.sin_family = AF_INET;
hp = gethostbyname(name)
peeraddr.sin_addr.s_addr = ((struct in_addr*)(hp->h_addr_list[0]))->s_addr;
```

Socket I/O: connect()

- *connect* client连接到server

```
#include <winsock.h>
int PASCAL FAR connect( SOCKET s, const struct
sockaddr FAR* name, int namelen);
   s：标识一个未连接套接字的描述字。
   name：欲进行连接的端口名。
   namelen：名字长度。
   返回值：若无错误发生，则connect()返回0。否则的话，返回
SOCKET_ERROR错误。
```

```
int ret = 0;
ret = connect(m_hSocket, (LPSOCKADDR)&m_addr,
sizeof(m_addr));
if(ret == SOCKET_ERROR)
{     AfxMessageBox("连接失败");
      return FALSE;
}
```

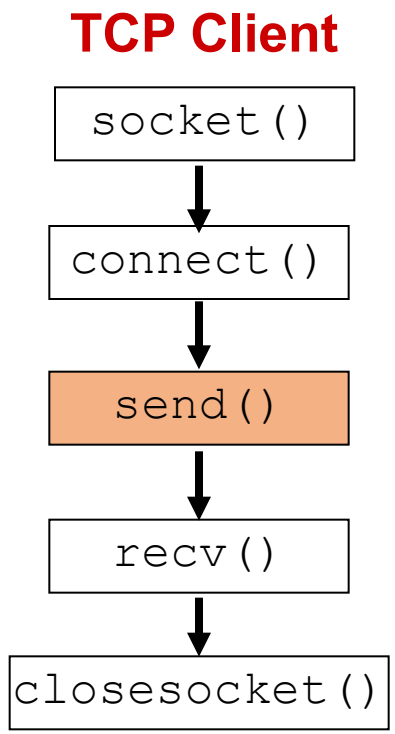


Socket I/O: send()

- *send* 向一个已连接的socket发送数据

```
#include <winsock.h>
int PASCAL FAR send( SOCKET s, const char FAR* buf,
int len, int flags);
   s: 一个用于标识已连接套接字的描述字。
   buf: 包含待发送数据的缓冲区。
   len: 缓冲区中数据的长度。
   flags: 调用执行方式。
   返回值: 若无错误发生，返回所发送数据的总数，否则返回
SOCKET_ERROR错误。
```

```
if(send(m_hSocket,buf,strlen(buf),0)==SOCKET_ERROR)
{
        AfxMessageBox("Send data error");
}
```



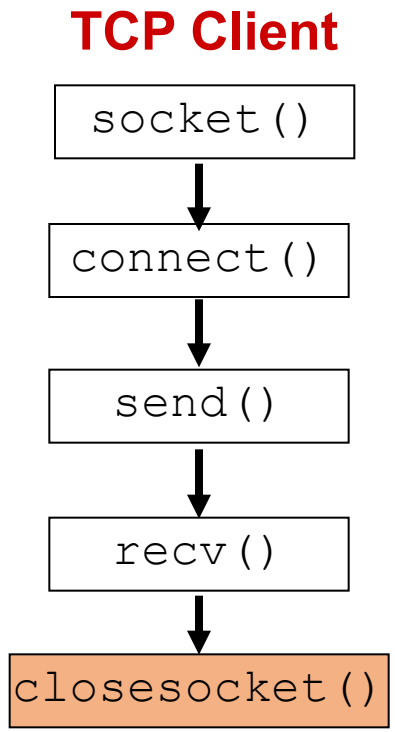
Socket I/O: closesocket()

- *closesocket* 关闭一个socket

```
#include <winsock.h>
int PASCAL FAR closesocket( SOCKET s);
```

s：一个套接字的描述字。

返回值：如无错误发生，则返回0，否则返回SOCKET_ERROR错误。



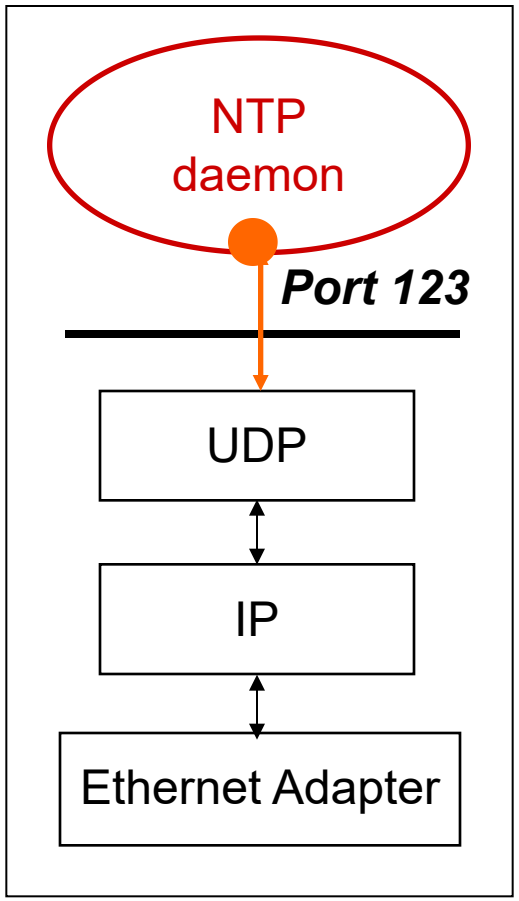
tcp_server.c

```
#include <winsock.h>
#define MY_PORT 3434

int main() {
    SOCKET listen_sock, new_sock;
    struct sockaddr_in my_addr;
    int dummy;
    char *buffer ="How old are you?\n";
    WSADATA wsaData;

    WSAStartup(MAKEWORD(1,1),&wsaData);

    listen_sock = socket(AF_INET, SOCK_STREAM, 0);

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MY_PORT);
    my_addr.sin_addr.s_addr = INADDR_ANY;

    bind(listen_sock, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
    listen(listen_sock, 5);
    new_sock = accept(listen_sock, NULL, &dummy);
    send(new_sock, buffer, strlen(buffer), 0);

    closesocket(new_sock);
    closesocket(listen_sock);
    WSACleanup();

    return 0;
}
```

tcp_client.c

```
#include <winsock.h>
#include <stdio.h>

#define MY_PORT 3434
int main() {
    SOCKET conn_sock;
    struct sockaddr_in remote_addr;
    int bytes_recvd;
    char buffer[100];
    WSADATA wsaData;

    WSAStartup(MAKEWORD(1,1),&wsaData);

    conn_sock = socket(AF_INET, SOCK_STREAM, 0);

    remote_addr.sin_family = AF_INET;
    remote_addr.sin_port = htons(MY_PORT);
    remote_addr.sin_addr.s_addr = inet_addr("137.189.90.38");

    connect(conn_sock, (struct sockaddr *)&remote_addr, sizeof(struct sockaddr));

    bytes_recvd = recv(conn_sock, buffer, sizeof(buffer), 0);

    printf("Received (%d bytes): \"%s\"\n", bytes_recvd, buffer);

    closesocket(conn_sock);
    WSACleanup();

    return 0;
}
```

UDP Server



- 举例: NTP daemon
- 为了接收来自*UDP client*的*服务请求，UDP server*应做什么准备?

Socket I/O: socket()

- UDP server必须创建一个datagram socket

```
SOCKET m_hSocket;
m_hSocket=socket(AF_INET,SOCK_DGRAM,0);
```

AF_INET 把socket与Internet协议族相关联

SOCK_DGRAM 选择UDP协议

Socket I/O: bind()

- 把*socket* 绑定到一个特定端口

```
sockaddr_in m_addr;
m_addr.sin_family = AF_INET;
m_addr.sin_addr.S_un.S_addr =htonl(INADDR_ANY);
m_addr.sin_port = htons(123);
      int ret = 0;int error = 0;
ret = bind(m_hSocket, (LPSOCKADDR)&m_addr, sizeof(m_addr));
if(ret == SOCKET_ERROR)
{
      AfxMessageBox("Binding Error"); //绑定错误
      return FALSE;
}
```

Socket I/O: recvfrom()

- *recvfrom* 接收一个数据报并保存源地址

```
#include <winsock.h>
int PASCAL FAR recvfrom( SOCKET s, char FAR* buf, int len, int
flags,struct sockaddr FAR* from, int FAR* fromlen);
```

s：标识一个套接字的描述字。

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。

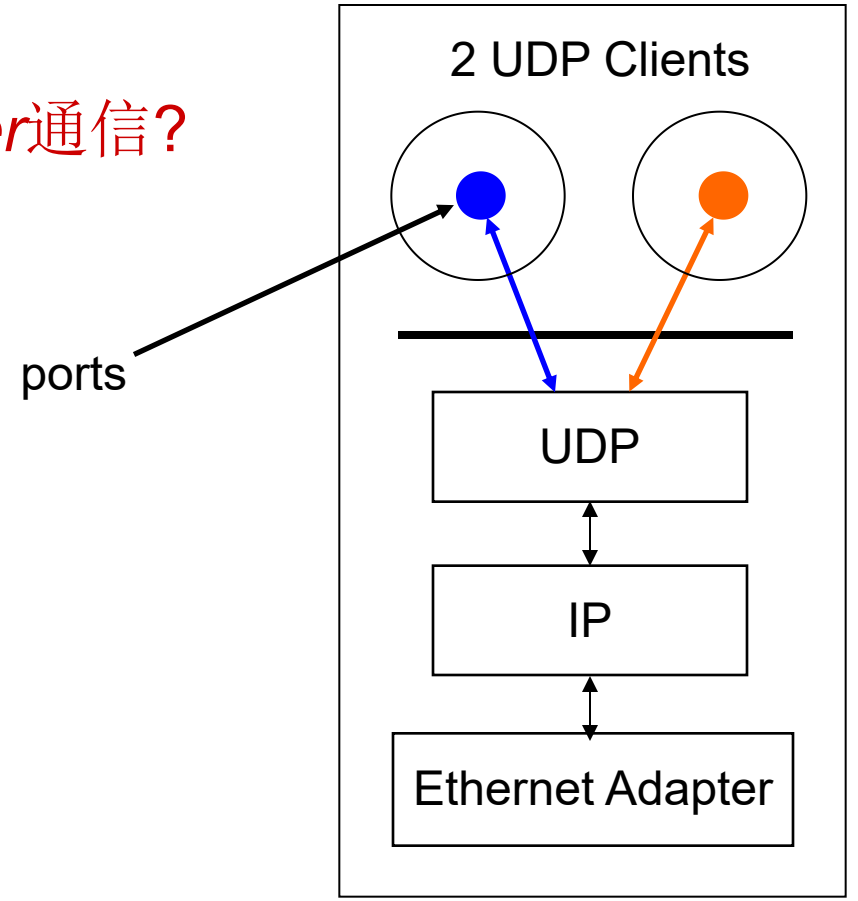
返回值：若无错误发生，返回读入的字节数。如果连接已中止，返回0。否则返回**SOCKET_ERROR**错误。

Socket I/O: recvfrom()

```
SOCKET srvsock;
sockaddr  cli;
char buf[512];
int cli_len = sizeof(cli);
int nbytes;
……
nbytes = recvfrom(srvsock, buf, sizeof(buf), 0 ,(struct sockaddr*) &cli,
&cli_len);
if(nbytes < 0) {
      AfxMessageBox(“接收数据失败”);
      return FALSE;
}
```

UDP Client

- *UDP client* 如何与*UDP server*通信?

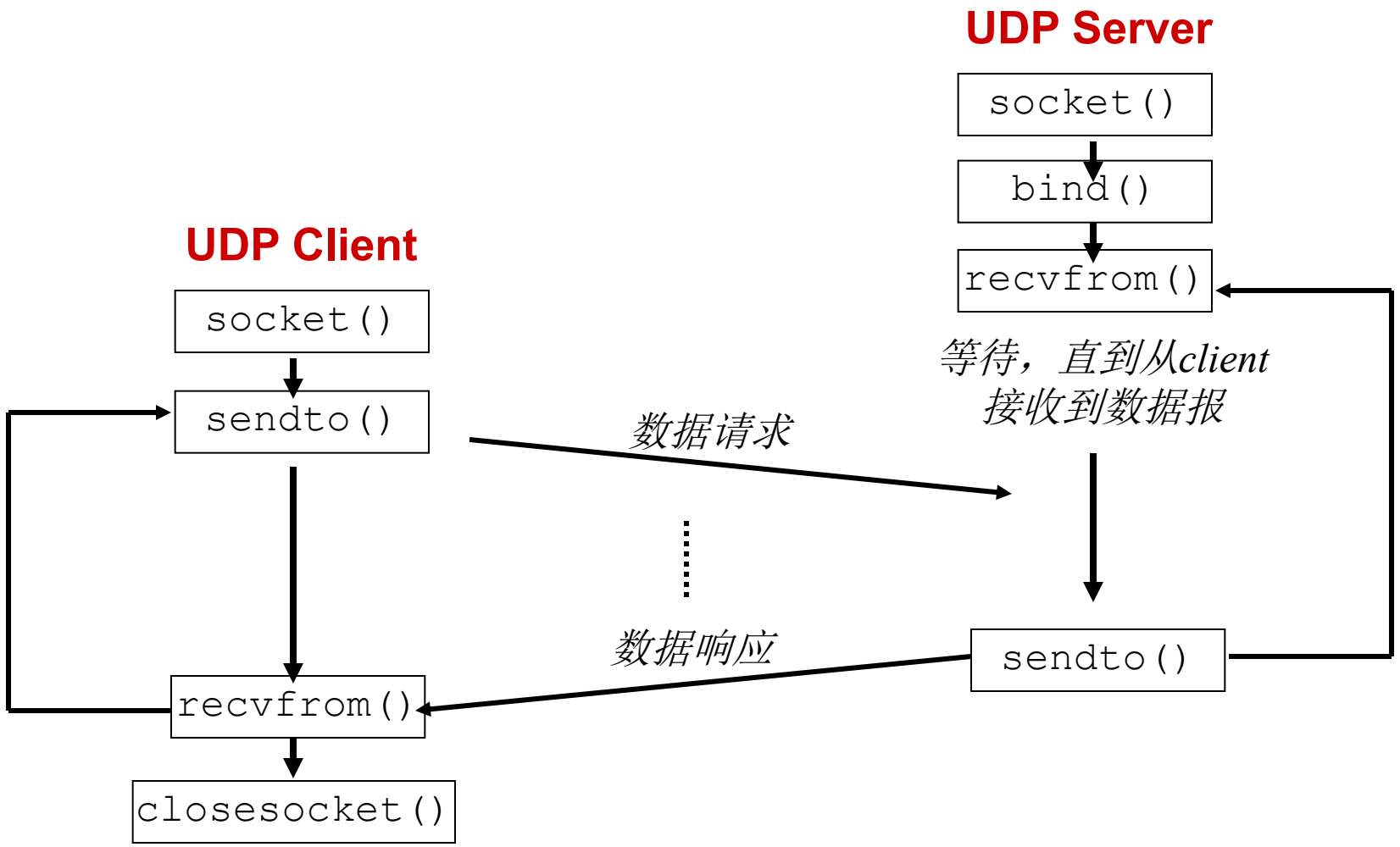


Socket I/O: sendto()

- UDP client 不与特定端口号绑定
  - 第一次调用*sendto*时，动态为其分配一个端口号

```
SOCKET clisock;
char buf[512];
int nbytes;
……
sockaddr_in m_addr;
m_addr.sin_family = AF_INET;
m_addr.sin_addr.S_un.S_addr =inet_addr("128.192.35.50");
m_addr.sin_port = htons(123);
nbytes = sendto(clisock, buf, sizeof(buf),0,
     (struct sockaddr*) &m_addr, sizeof(m_addr));
if(nbytes < 0) {
      AfxMessageBox("发送数据失败");
      return FALSE;
}
```

UDP Client-Server交互流程



udp_server.c

```
#include <winsock.h>
#define MY_PORT 3434
int main() {
    SOCKET udp_sock;
    struct sockaddr_in remote_addr, local_addr;
    char *buffer = "How do you do?";
    WSADATA wsaData;
    WSAStartup(MAKEWORD(1,1),&wsaData);
    udp_sock = socket(AF_INET, SOCK_DGRAM, 0);

    local_addr.sin_family = AF_INET;
    local_addr.sin_port = htons(MY_PORT);
    local_addr.sin_addr.s_addr = INADDR_ANY;

    bind(udp_sock, (sockaddr *)&local_addr, sizeof(struct sockaddr_in));

    remote_addr.sin_family = AF_INET;
    remote_addr.sin_port = htons(MY_PORT);
    remote_addr.sin_addr.s_addr = inet_addr("137.189.90.38");

    for (int i = 0; i < 10; i++)
        sendto(udp_sock, buffer, strlen(buffer), 0, (struct sockaddr *)&remote_addr, sizeof(struct sockaddr_in));

    closesocket(udp_sock);
    WSACleanup();
    return 0;
}
```

udp_client.c

```
#include <winsock.h>
#include <stdio.h>
#define MY_PORT 3434
int main() {
    SOCKET udp_sock;
    struct sockaddr_in remote_addr, local_addr;
    int bytes_recvd, dummy;
    char buffer[100];
    WSADATA wsaData;

    WSAStartup(MAKEWORD(1,1),&wsaData);
    udp_sock = socket(AF_INET, SOCK_DGRAM, 0);

    local_addr.sin_family = AF_INET;
    local_addr.sin_port = htons(MY_PORT);
    local_addr.sin_addr.s_addr = INADDR_ANY;

    bind(udp_sock, (sockaddr *)&local_addr, sizeof(struct sockaddr_in));
    bytes_recvd = recvfrom(udp_sock, buffer, sizeof(buffer), 0, NULL, &dummy);
    buffer[bytes_recvd] = '\0';

    printf("Received (%d bytes): \"%s\"\n", bytes_recvd, buffer);

    closesocket(udp_sock);
    WSACleanup();

    return 0;
}
```

Winsock2.0

- Winsock2.0是对winsock1.1的升级
  - 支持多种传输协议
    - TCP/IP、IPX/SPX、ATM、NETBIOS、AppleTalk、红外线
  - 重叠I/O和事件对象
  - 服务质量
  - 套接字组
  - 共享套接字
  - 协议无关的多播通信

Winsock2.0提供的新函数(1)

- WSAAccept() accept()函数的扩展版本，它支持条件接收和套接字分组。
- WSACloseEvent() 释放一个事件对象。
- WSAConnect() connect()函数的扩展版本，它支持连接数据交换和QOS规范。
- WSACreateEvent() 创建一个事件对象。
- WSADuplicateSocket() 为一个共享套接字创建一个新的套接字描述字。
- WSAEnumNetworkEvents() 检查是否有网络事件发生。
- WSAEnumProtocols() 得到每个可以使用的协议的信息。
- WSAEventSelect() 把网络事件和一个事件对象连接。

Winsock2.0提供的新函数(2)

- WSAGetOverlappedResu()　得到重叠操作的完成状态。
- WSAGetQOSByName()　对于一个传输协议服务名字提供相应的QOS参数。
- WSAHtonl()　htonl()函数的扩展版本。
- WSAHtons()　htons()函数的扩展版本。
- WSAIoctl()　ioctlsocket()函数的允许重叠操作的版本。
- WSAJoinLeaf()　在多点对话中加入一个叶节点。
- WSANtohl()　ntohl()函数的扩展版本。
- WSANtohs()　ntohs()函数的扩展版本。

Winsock2.0提供的新函数(3)

- WSARecv() recv()函数的扩展版本。它支持分散/聚集I/O和重叠套接字操作。
- WSARecvDisconnect() 终止套接字的接收操作。如果套接字是基于连接的，得到拆除数据。
- WSARecvFrom() recvfrom()函数的扩展版本。它支持分散/聚集I/O和重叠套接字操作。
- WSAResetEvent() 重新初始化一个事件对象。
- WSASend() send()函数的扩展版本。它支持分散/聚集I/O和重叠套接字操作。
- WSASendDisconnect() 启动一系列拆除套接字连接的操作，并且可以选择发送拆除数据。
- WSASendTo() sendto()函数的扩展版本。它支持分散/聚集I/O和重叠套接字操作。

Winsock2.0提供的新函数(4)

- WSASetEvent()　　设置一个事件对象。
- WSASocket()
  - socket()函数的扩展版本。它以一个PROTOCOL_INFO结构作为输入参数，并且允许创建重叠套接字。它还允许创建套接字组。
- WSAWaitForMultipleEvents()　　阻塞多个事件对象。