计算机网络

14. CLIENT-SERVER AND SOCKET API

厦门大学软件学院

黄炜 助理教授

PART IV Network Applications

Ch 26 Client-Server Interaction

客户-服务器交互

Comparison: OSI vs. TCP/IP

- **Every application program must include all tasks assigned to the session, presentation, and application layers of the OSI model.**

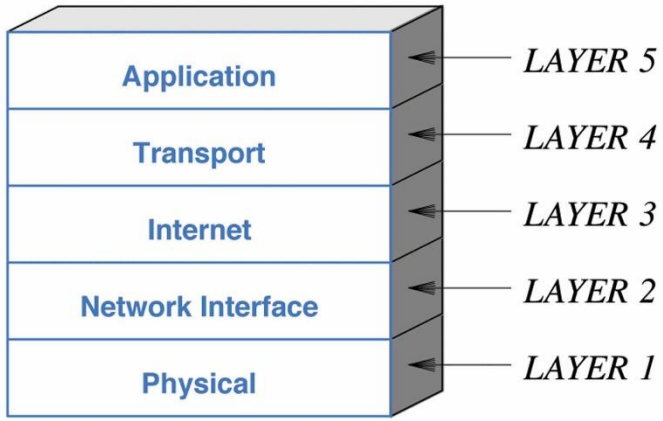


Figure 1.1 The layering model used with the Internet protocols (TCP/IP).

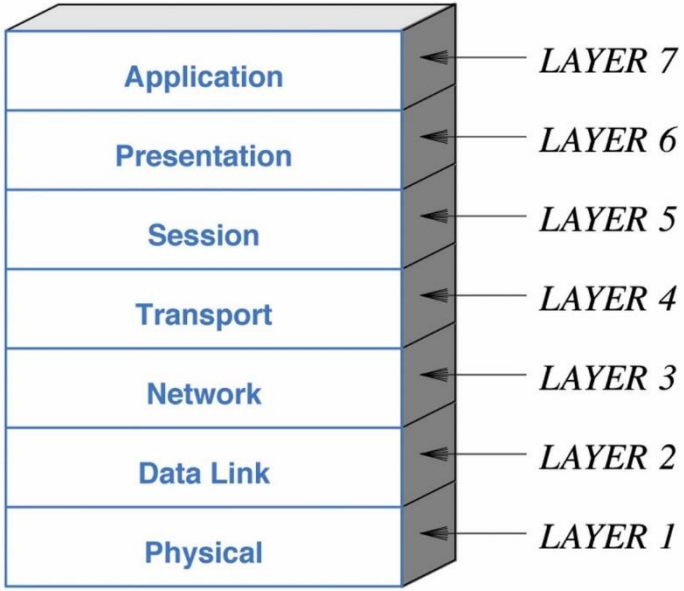


Figure 1.4 The OSI seven-layer model standardized by ISO.

Functionality An Internet Provides

- 虽然互联网系统提供基本的通信服务，协议软件不能开始联系，或接受来自远程计算机的联系。
- 应用程序参与通信，一个启动通信和另一个接受它。
- 没有办法通知应用程序通信已到达的协议软件，并没有办法同意接受任意输入消息的应用程序。
- 应用程序通知本地协议软件，该软件是一种特定类型的消息，然后应用程序等待。

Client-Server Paradigm

- 当传入消息与应用程序指定的时候相匹配时，协议软件将该消息传递给应用程序。
- 一个应用必须积极启动互动，而另一个应用被动等待。
  - **app. that actively initiates contact is called a client,**
  - **app. that passively waits for contact is called a server.**

Client

- **A client is a program running on the local machine requesting service from a server.**
- **A client program is finite:**
  - 主动打开：客户端使用远程主机的地址和特定服务器程序运行在该机上的著名端口地址的地址通道打开通信通道。
  - 主动结束：虽然请求的响应部分可以重复几次，整个过程是有限的，并最终得出一个结束。在那一刻，客户端关闭通信通道。

Server

- **A server is a program running on the remote machine providing service to the clients.**
- **A server program is an infinite program.**
  - 当它启动时，它会打开一个被动打开的大门，向客户提供来自客户端的请求，但除非被要求，服务不会启动。
  - 当一个请求到达时，它会对请求作出响应，或者是迭代的，或者是同时进行的。

26.6 Characteristics of C/S

- **Client software**
  - 任意的应用程序，当需要远程访问时，就成为一个临时的客户端，但也可以在本地执行其他的计算程序。
  - 被调用直接由用户，并只执行一次会话。
  - 本地用户的个人计算机上运行。
  - 主动与服务器接触。
  - 可以在需要的时候访问多个服务，但是在一个时间中，主动联系一个远程服务器。
  - 不需要特殊的硬件或复杂的操作系统。

26.6 Characteristics of C/S

- **Server software**
  - 一个特殊用途，专用于提供一个服务的专用程序，同时可以同时处理多个远程客户端。
  - 系统启动时自动调用，并通过多个会话连续执行。
  - 在共享计算机上运行（不在用户的个人计算机上）。
  - 被动地等待从任意远程客户的联系
  - 接受来自任意客户的联系，但提供单一服务。
  - 需要强大的硬件和复杂的操作系统。

Transport Protocols

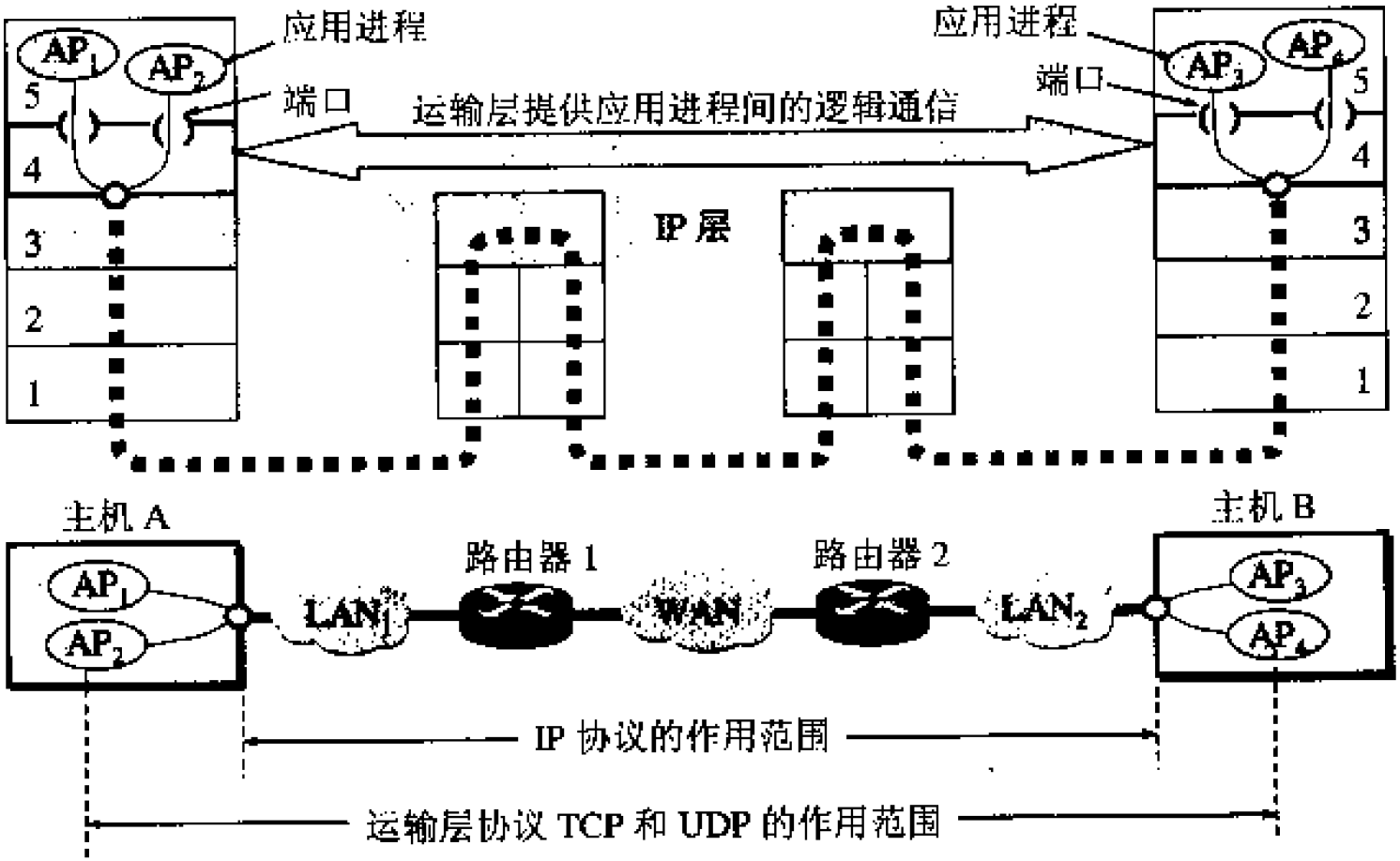


图 7-2　运输层为相互通信的应用进程提供了逻辑通信

Multiple services on one computer

- **TCP / UDP multiplexing and demultiplexing**
  - 一个够强大的计算机系统可同时运行多个客户端和服务器。
  - 计算机只需要单一的物理连接到互联网。
  - TCP程序通过TCP报头中由源主机指出的端口地址，了解到发送主机希望目标主机的什么应用层程序接收数据报。
  - 相应地，UDP类似。

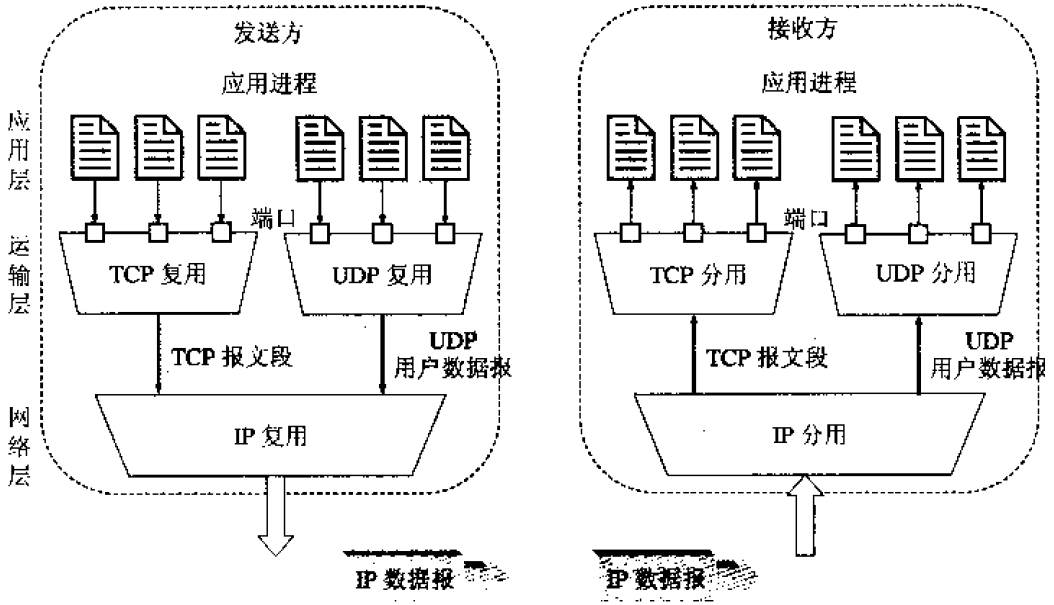


图 7-7　端口在进程之间的通信中所起的作用

Multiple services on one computer

- 为了在通信时不致发生混乱，就必须把端口号和主机的IP地址结合在一起使用。
- TCP连接由它的两个端点来标识
  - 每个端点由IP地址和端口号决定的，称为Socket（插口）。
- UDP无连接但也需要Socket
  - 虽然在相互通信的两个进程之间没有一条虚连接，但发送端UDP一定有一个发送端口而在接收端UDP也一定有一个接收端口。

并发（Concurrency）

- 允许多个应用程序在同一时间执行的计算机系统被称为支持并发。
- 一个拥有多个线程的程序称为并发程序。
- 并发是客户端-服务器交互模式的基础。
- 因为并发服务器同时为多个客户提供服务，而不需要每个客户机等待上一个客户端来完成。

UDP报文队列

- UDP与应用层之间的端口都是用报文队列来实现。
- 操作系统为该进程创建：入队列和出队列

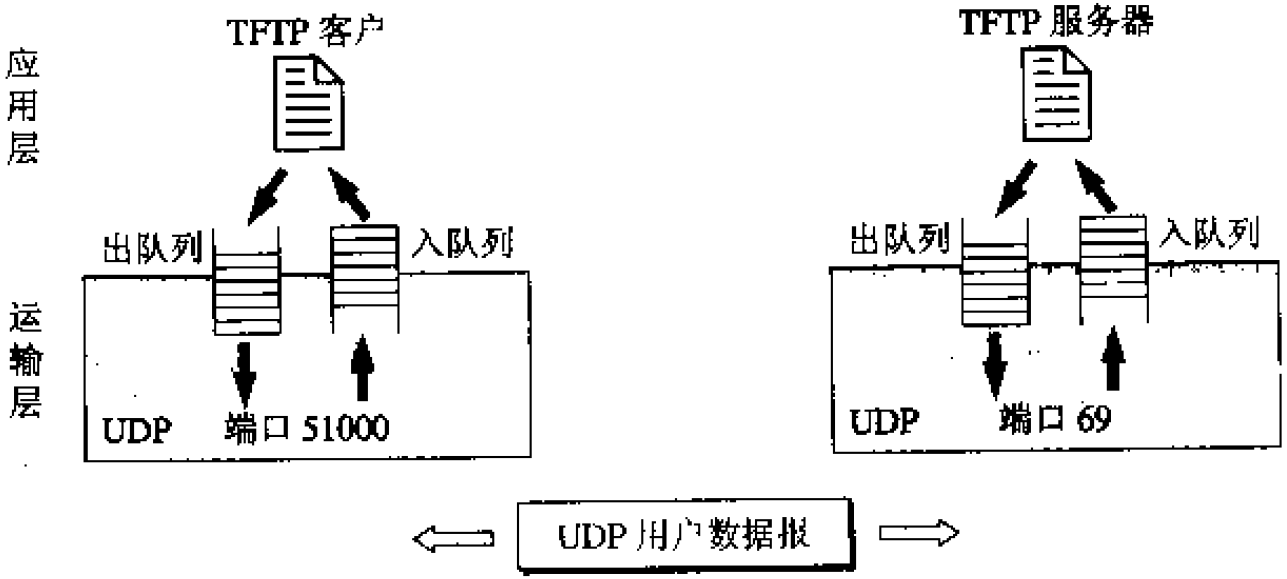
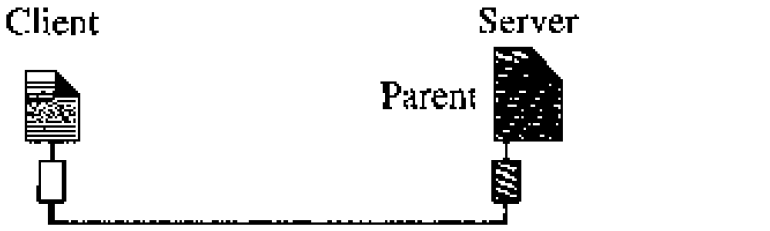


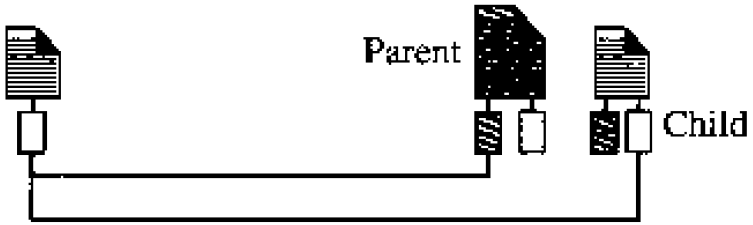
图 7-9　UDP 中的队列

Relationship between CS

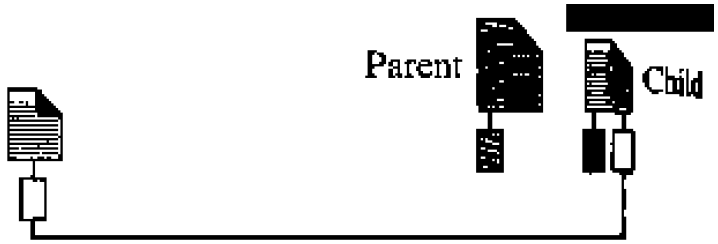


a. After connect, before accept

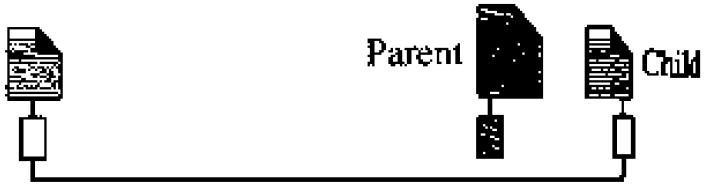
b. After accept



c. After fork



d. After parent closes ephemeral port



e. After child closes well-known port

PART IV Network Applications

Ch 27 The Socket Interface

Socket 接口

Socket

把应用搬上网络

Application Program Interface(API)

- **A programmer can create Internet application software without understanding the underlying network technology or communication protocols.**

单机进程通信

- 进程通信的概念最初来源于单机系统。
- 由于每个进程都在自己的地址范围内运行，为保证两个相互通信的进程之间既互不干扰又协调一致工作，操作系统为进程通信提供了相应设施：
  - 如UNIX BSD中的管道（pipe）、命名管道（named pipe）和软中断信号（signal），UNIX system V的消息（message）、共享存储区（shared memory）和信号量（semaphore)等，但都仅限于用在本机进程之间通信。

网间进程通信

- 网间进程通信要解决不同主机进程间的相互通信问题。
  - 可以把同机进程通信看作是其中的特例。
  - 首先要解决的是网间进程标识问题。同一主机上，不同进程可用进程号（process ID）唯一标识。但在网络环境下，各主机独立分配的进程号不能唯一标识该进程。
  - 其次，操作系统支持的网络协议众多，不同协议的工作方式不同，地址格式也不同。因此，网间进程通信还要解决多重协议的识别问题。

协议端口

- 网络中可以被命名和寻址的通信端口，是操作系统可分配的一种资源。
- 传输层与网络层在功能上的最大区别是传输层提供进程通信能力。
- 网络通信的最终地址就不仅仅是主机地址了，还包括可以描述进程的某种标识符。
  - TCP/IP协议提出了协议端口（protocol port，简称端口）的概念，用于标识通信的进程。
  - 端口是一种抽象的软件结构（包括一些数据结构和I/O缓冲

协议端口

- 在TCP/IP协议的实现中，端口间的操作类似于一般的I/O操作，进程获取一个端口，相当于获取本地唯一的I/O文件，可以用一般的读写原语访问。
- 每个端口有端口号（port number），区别不同端口。
  - TCP和UDP是完全独立的两个软件模块，因此各自的端口号也相互独立
    - TCP有一个255号端口，UDP也有一个255号端口，二者并不冲突。
  - 常用端口号要记住

Network Programming & Socket

- The interface an app uses to specify communication is known as an Application Program Interface (API)
  - Details of an API depend on the OS
- Socket API (Sockets for short) has emerged as the de facto standard for software that communicates over the Internet
- The socket API is available for many OS
  - Windows systems as well as various UNIX systems

Sockets, Descriptors, Network I/O

- **Part of the UNIX OS, integrated with I/O**
  - **An app. creates a socket to use for Internet**
  - **OS returns an integer descriptor**
    - 计算机常用一个无符号整数表示指针、句柄等
  - **The app. passes the descriptor as an argument**
    - **when it calls functions to perform an operation on the socket**
  - **In many OS, socket descriptors are integrated with other I/O descriptors**
  - **An application can use the read and write operations for socket I/O or I/O to a file**

Parameters and the Socket API

- Socket programming differs from conventional I/O
- An app. must specify many details, such as
  - the address of a remote computer
  - the protocol port number
  - whether act as a client or as a server
- An app. creates a socket, and then invokes functions for details
  - Advantage: have three or fewer parameters
  - Disadv.: programmer must call multiple functions

主要的Socket API函数名

Name	Used By	Meaning
accept	server	Accept an incoming connection
bind	server	Specify IP address and protocol port
close	either	Terminate communication
connect	client	Connect to a remote application
getpeername	server	Obtain client's IP address
getsockopt	server	Obtain current options for a socket
listen	server	Prepare socket for use by a server
recv	either	Receive incoming data or message
recvmsg	either	Receive data (message paradigm)
recvfrom	either	Receive a message and sender's addr.
send (write)	either	Send outgoing data or message
sendmsg	either	Send an outgoing message
sendto	either	Send a message (variant of sendmsg)
setsockopt	either	Change socket options
shutdown	either	Terminate a connection
socket	either	Create a socket for use by above

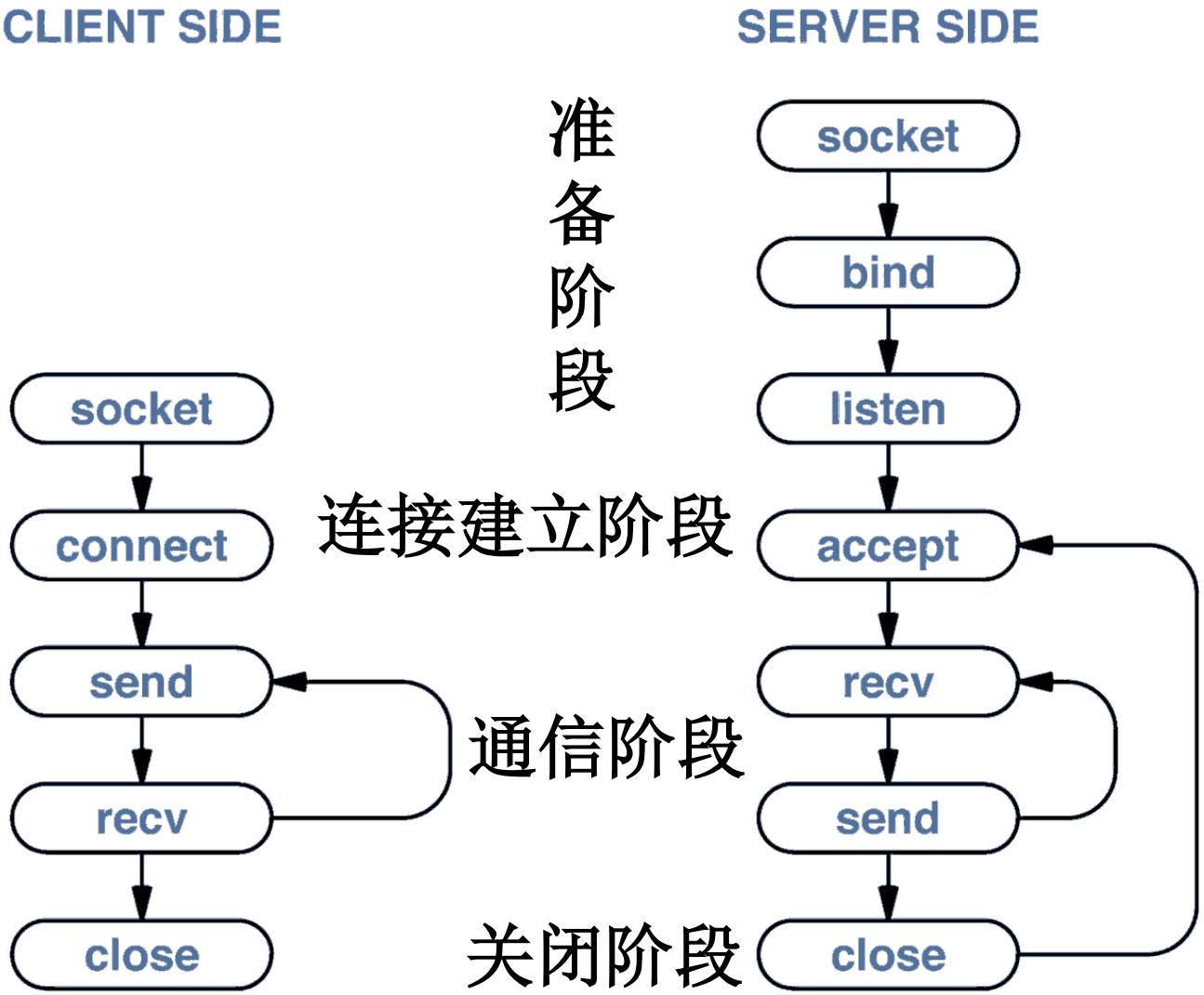
Server和Client有什么区别？

区别大了

Server和Client的区别

- 轻重区别
  - A server is a system (hardware and software) that responds to requests across a computer network to provide, or help to provide, a network service.
  - A client is a piece of computer hardware or software that accesses a service made available by a server.
  - The client–server (C/S) model is a distributed application structure in computing that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

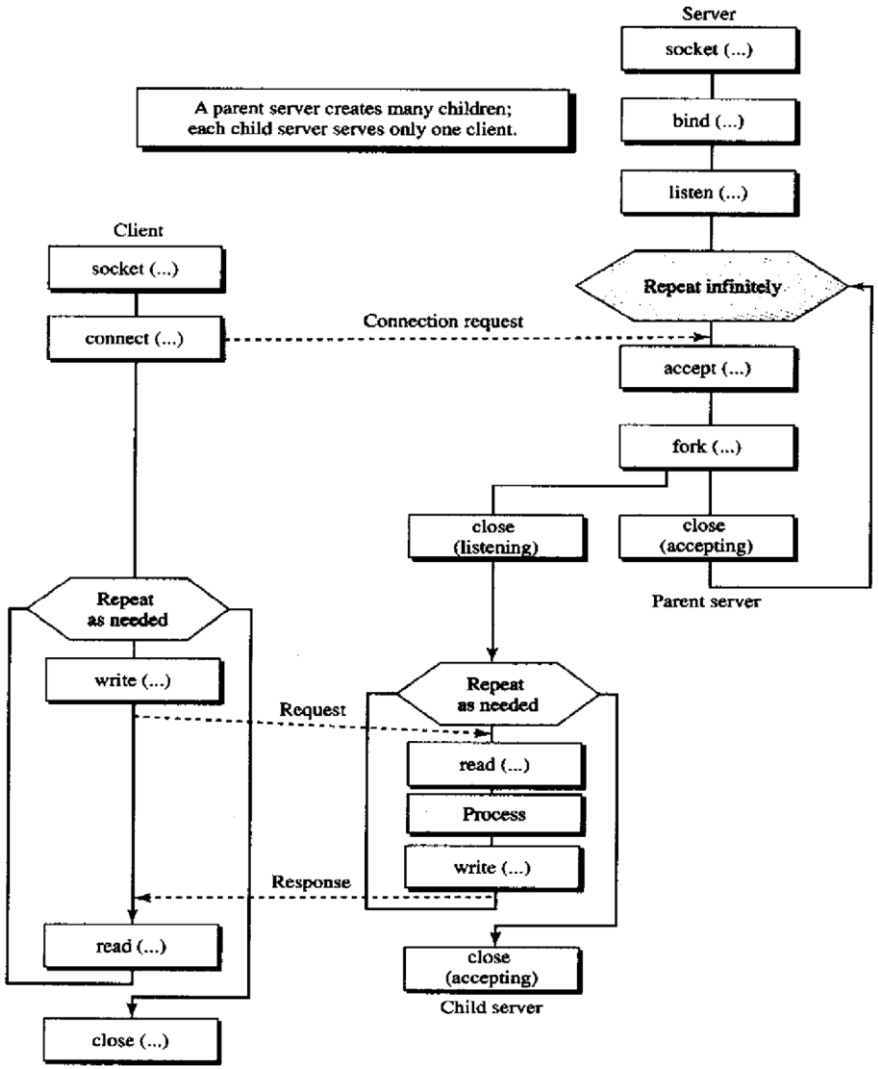
Socket Calls in a Client and Server



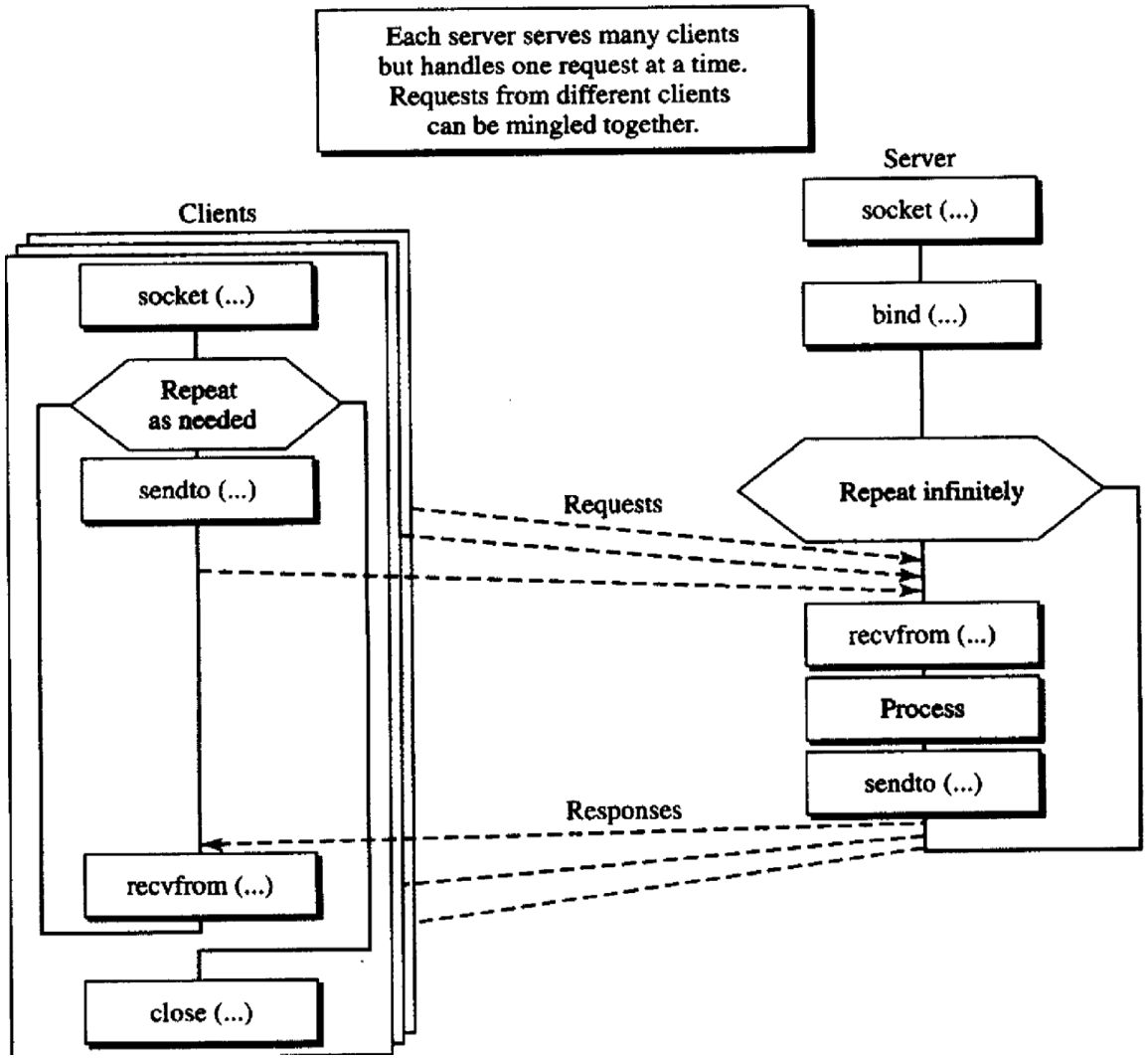
是否网络连接都是这样的流程？

UDP肯定不是这么连接的

面向连接的服务


Server
socket (...)
bind (...)
listen (...)
A parent server creates many children;
each child server serves only one client.
Client
socket (...)
connect (...)
Connection request
Repeat infinitely
accept (...)
fork (...)
close
(listening)
close
(accepting)
Parent server
Repeat
as needed
write (...)
Request
Repeat
as needed
read (...)
Process
write (...)
Response
read (...)
close (...)
close
(accepting)
Child server

无连接的服务


Each server serves many clients
but handles one request at a time.
Requests from different clients
can be mingled together.
Server
socket (...)
bind (...)
Repeat infinitely
recvfrom (...)
Process
sendto (...)
Clients
socket (...)
Repeat
as needed
sendto (...)
recvfrom (...)
close (...)
Requests
Responses

下面讲Windows系统的Socket

UNIX版本的Socket参见课本

半相关和全相关

- 半相关（half-association）
  - 网络中用三元组在全局唯一标志一个进程：
    - （协议，本地地址，本地端口号）指定连接的每半部分
- 全相关（association）
  - 一个完整的网间进程通信需要由两个进程组成，并且只能使用同一种高层协议。
  - 一个完整的网间通信需要一个五元组来标识

（协议，本地地址，本地端口号，远程地址，远程端口号）

- 两个协议相同的半相关才能组合成一个相关，完全指定组成一连接。

Socket

- Local socket address
  - Local IP Address
  - Local IP Port
- Remote socket address
  - Remote IP Address
  - Remote IP Port
- Protocol
  - TCP, UDP, raw IP

- IP地址如何选择？
  - 本机IP有多个怎么办？
- 端口如何选择？
  - 常用端口号的作用要记下来
- 协议如何选择？
  - 打电话还是发短信

系统要求

- **WinSock 2 Library**
  - 本节内容源自MSDN

Item	Value
Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	Winsock2.h
Library	Ws2_32.lib
DLL	Ws2_32.dll

Socket Functions

- **Used by Both Client and Server**
  - **3.17.1 The Socket Function**
  - **3.17.2 The Send Function**
  - **3.17.3 The Recv Function**
  - **3.17.4 Read and Write with Sockets**
  - **3.17.5 The Close Function**

Socket Functions

- **Used Only by a Client**
  - **The Connect Function**
- **Used Only by a Server**
  - **3.19.1 The Bind Function**
  - **3.19.2 The Listen Function**
  - **3.19.3 The Accept Function**

Socket Functions

- **Used with the Message Paradigm**
  - **3.20.1 Sendto and Sendmsg Socket Functions**
  - **3.20.2 Recvfrom and Recvmsg Functions**
- **Other Socket Functions**
  - **getpeername, gethostname**
  - **setsockopt, getsockopt**
  - **gethostbyname, gethostbyaddr**

Sockets, Threads, and Inheritance

- **SAPI works well with concurrent servers**
- **Principle of Implementations of the SAPI:**
  - **Each new thread that is created inherits a copy of all open sockets from the one that created it**
  - **A ref. count mechanism to control each socket**
    - **First created: sys. sets the socket's RC to 1**
    - **Create additional thread: the thread inherits a pointer to each open socket the program owns**
      - **sys increments the reference count of each socket by 1**
    - **Thread calls close: sys. decrements the RC for the socket, if the RC has reached zero, the socket is removed**

The socket Function

- The socket function creates a socket that is bound to a specific transport service provider.

```
SOCKET WSAAPI socket(
  _In_  int af,
  _In_  int type,
  _In_  int protocol
);
```

af [in]
The address family specification.

type [in]
The type specification for the new socket.

protocol [in]
The protocol to be used. The possible options for the protocol parameter are specific to the address family and socket type specified.

The socket Function

- **Param 1: Address Family**

Af	Meaning
AF_UNSPEC 0	The address family is unspecified.
AF_INET 2	The Internet Protocol version 4 (IPv4) address family.
AF_IPX 6	The IPX/SPX address family. This address family is only supported if the NWLink IPX/SPX NetBIOS Compatible Transport protocol is installed.
AF_APPLETALK 16	The AppleTalk address family. This address family is only supported if the AppleTalk protocol is installed.
AF_NETBIOS 17	The NetBIOS address family. This address family is only supported if the Windows Sockets provider for NetBIOS is installed.
AF_INET6 23	The Internet Protocol version 6 (IPv6) address family.
AF_IRDA 26	The Infrared Data Association (IrDA) address family. This address family is only supported if the computer has an infrared port and driver installed.
AF_BTH 32	The Bluetooth address family.

The socket Function

- **Param 2: Socket Type**

Type	Meaning
SOCK_STREAM 1	A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. This socket type uses the Transmission Control Protocol (TCP) for the Internet address family (AF_INET or AF_INET6).
SOCK_DGRAM 2	A socket type that supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. This socket type uses the User Datagram Protocol (UDP) for the Internet address family (AF_INET or AF_INET6).
SOCK_RAW 3	A socket type that provides a raw socket that allows an application to manipulate the next upper-layer protocol header. To manipulate the IPv4 header, the IP_HDRINCL socket option must be set on the socket. To manipulate the IPv6 header, the IPV6_HDRINCL socket option must be set on the socket.
SOCK_RDM 4	A socket type that provides a reliable message datagram. An example of this type is the Pragmatic General Multicast (PGM) multicast protocol implementation in Windows, often referred to as reliable multicast programming. This type value is only supported if the Reliable Multicast Protocol is installed.
SOCK_SEQPACKET 5	A socket type that provides a pseudo-stream packet based on datagrams.

The socket Function

- **Param 3: Protocol**

Protocol	Meaning
0	If a value of 0 is specified, the caller does not wish to specify a protocol and the service provider will choose the protocol to use.
IPPROTO_ICMP 1	The Internet Control Message Protocol (ICMP). This is a possible value when the af parameter is AF_UNSPEC, AF_INET, or AF_INET6 and the type parameter is SOCK_RAW or unspecified.
IPPROTO_IGMP 2	The Internet Group Management Protocol (IGMP). This is a possible value when the af parameter is AF_UNSPEC, AF_INET, or AF_INET6 and the type parameter is SOCK_RAW or unspecified.
BTHPROTO_RFCOMM 3	The Bluetooth Radio Frequency Communications (Bluetooth RFCOMM) protocol. This is a possible value when the af parameter is AF_BTH and the type is SOCK_STREAM.
IPPROTO_TCP 6	The Transmission Control Protocol (TCP). This is a possible value when the af parameter is AF_INET or AF_INET6 and the type parameter is SOCK_STREAM.
IPPROTO_UDP 17	The User Datagram Protocol (UDP). This is a possible value when the af parameter is AF_INET or AF_INET6 and the type parameter is SOCK_DGRAM.
IPPROTO_ICMPV6 58	The ICMPv6. This is a possible value when the af parameter is AF_UNSPEC, AF_INET, or AF_INET6 and the type parameter is SOCK_RAW or unspecified.
IPPROTO_RM 113	The PGM protocol for reliable multicast. This is a possible value when the af parameter is AF_INET and the type parameter is SOCK_RDM.

The `socket` Function

- **Return value**
  - **If no error occurs, socket returns a descriptor referencing the new socket. Otherwise, a value of `INVALID_SOCKET` (Preferred Style) is returned, and a specific error code can be retrieved by calling `WSAGetLastError`.**
- **Advanced**
  - **Sockets without the overlapped attribute can be created by `WSASocket`.**
  - **Connection-oriented sockets such as `SOCK_STREAM` provide full-duplex connections, and must be in a connected state before any data can be sent or received on it. A connection to another socket is created with a `connect` call. Once connected, data can be transferred using `send` and `recv` calls. When a session has been completed, a `closesocket` must be performed.**

The `bind` Function

- The `bind` function associates a local address with a socket.

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

`s` [in]
A descriptor identifying an unbound socket.

`name` [in]
A pointer to a `sockaddr` structure of the local address to assign to the bound socket.

`namelen` [in]
The length, in bytes, of the value pointed to by the name parameter.

The sockaddr Structure

- The sockaddr structure varies depending on the protocol selected.

```
struct sockaddr {
        ushort  sa_family;
        char    sa_data[14];
};

struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

The in_addr structure represents an IPv4 Internet address.

The bind Function

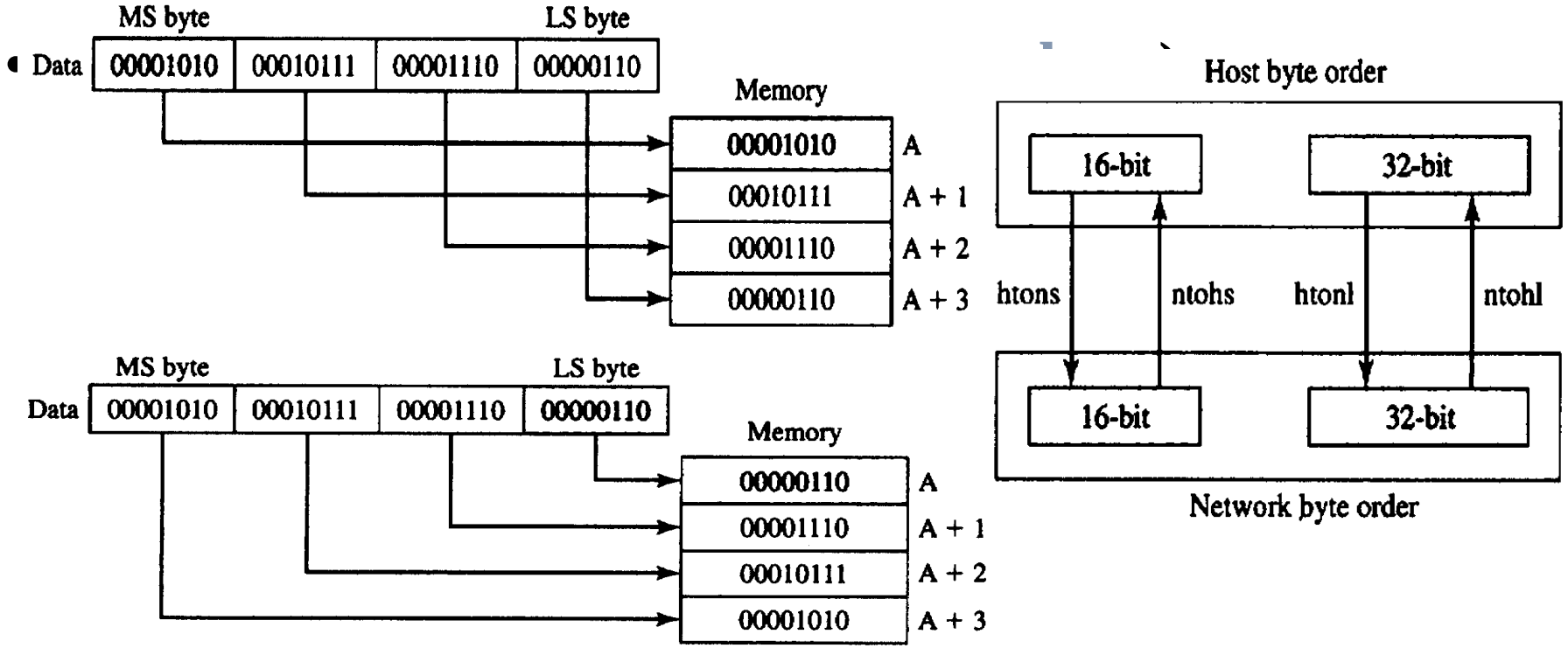
```
// Declare variables
SOCKET ListenSocket;
struct sockaddr_in saServer;
hostent* localHost;
char* localIP;

// Create a listening socket
ListenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// Get the local host information
localHost = gethostbyname("");
localIP = inet_ntoa (*(struct in_addr *)*localHost->h_addr_list);

// Set up the sockaddr structure
saServer.sin_family = AF_INET;
saServer.sin_addr.s_addr = inet_addr(localIP);
saServer.sin_port = htons(5150);

// Bind the listening socket using the
// information in the sockaddr structure
bind( ListenSocket,(SOCKADDR*) &saServer, sizeof(saServer) );
```

网络字节顺序（htons）

- 字节顺序
  - 不同计算机存放多字节值的顺序不同，有的在起始地址存放低位字节（小数在前），有的存高位字节（大数在前）。
- 网络字节顺序
  - 网络协议需指定字节顺序，保证数据的正确性
  - TCP/IP使用16位和32位整数的大数在前格式
- 主机中的字节顺序
  - Big-endian byte order
  - Little-endian byte order

网络字节顺序

- 上图大尾序：高地址为尾
- 下图小尾序：低地址为尾



The connect Function

- The connect function establishes a connection to a specified socket.

```
int connect(
   _In_   SOCKET s,
   _In_   const struct sockaddr *name,
   _In_   int namelen
);
```

s [in]
A descriptor identifying an unconnected socket.

name [in]
A pointer to a sockaddr structure to which the connection should be established.

namelen [in]
The length, in bytes, of the sockaddr structure pointed to by the name parameter.

The connect Function

```
    // The sockaddr_in structure specifies the address family,
    // IP address, and port of the server to be connected to.
    sockaddr_in clientService;
    clientService.sin_family = AF_INET;
    clientService.sin_addr.s_addr = inet_addr("127.0.0.1");
    clientService.sin_port = htons(27015);

    // Connect to server.
    iResult = connect(ConnectSocket, (SOCKADDR *) & clientService, sizeof
(clientService));
    if (iResult == SOCKET_ERROR) {
        wprintf(L"connect function failed with error: %ld\n", WSAGetLastError());
        iResult = closesocket(ConnectSocket);
        if (iResult == SOCKET_ERROR)
            wprintf(L"closesocket function failed with error: %ld\n",
WSAGetLastError());
        WSACleanup();
        return 1;
    }
    wprintf(L"Connected to server.\n");
```

The `listen` Function

- The `listen` function places a socket in a state in which it is listening for an incoming connection.

```
int listen(
  _In_  SOCKET s,
  _In_  int backlog
);
```

`s` [in]

A descriptor identifying a bound, unconnected socket.

`backlog` [in]

The maximum length of the queue of pending connections. If set to `SOMAXCONN`, the underlying service provider responsible for socket s will set the backlog to a maximum reasonable value. There is no standard provision to obtain the actual backlog value.

The `accept` Function

- The `accept` function permits an incoming connection attempt on a socket.

```
SOCKET accept(
  _In_      SOCKET s,
  _Out_     struct sockaddr *addr,
  _Inout_   int *addrlen
);
```

`s` [in]
A descriptor that identifies a socket that has been placed in a listening state with the `listen` function.

`addr` [out]
An optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer.

`addrlen` [in]
An optional pointer to an integer that contains the length of structure pointed to by the `addr` parameter.

The `accept` Function

- **Return value**
  - **If no error occurs, accept returns a value of type `SOCKET` that is a descriptor for the new socket. This returned value is a handle for the socket on which the actual connection is made.**
  - **Otherwise, a value of `INVALID_SOCKET` is returned, and a specific error code can be retrieved by calling `WSAGetLastError`.**

The accept Function

```
// Listen for incoming connection requests.
// on the created socket
if (listen(ListenSocket, 1) == SOCKET_ERROR) {
    wprintf(L"listen failed with error: %ld\n", WSAGetLastError());
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}

// Create a SOCKET for accepting incoming requests.
SOCKET AcceptSocket;
wprintf(L"Waiting for client to connect...\n");
// Accept the connection.
AcceptSocket = accept(ListenSocket, NULL, NULL);
if (AcceptSocket == INVALID_SOCKET) {
    wprintf(L"accept failed with error: %ld\n", WSAGetLastError());
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
} else
    wprintf(L"Client connected.\n");
```

The send Function

- The send function sends data on a connected socket.

```
int send(
   _In_   SOCKET s,
   _In_   const char *buf,
   _In_   int len,
   _In_   int flags
);
```

s [in]
A descriptor identifying a connected socket.

buf [in]
A pointer to a buffer containing the data to be transmitted.

len [in]
The length (bytes) of the data in buffer pointed to by the buf para.

flag [in]
A set of flags that specify the way in which the call is made.

The `send` Function

- **Parameter `flag`**
  - **Constructed by using the bitwise OR operator with any of the following.**

Value	Meaning
`0`	**Default.**
`MSG_DONTROUTE`	**Specifies that the data should not be subject to routing. A Windows Sockets service provider can choose to ignore this flag.**
`MSG_OOB`	**Sends OOB data (stream-style socket such as `SOCK_STREAM` only.**

- **Return value**
  - **If no error occurs, send returns the total number of bytes sent, which can be less than the number requested to be sent in the `len` parameter.**

    Otherwise, a value of `SOCKET_ERROR` is returned, and a specific error code can be retrieved by calling `WSAGetLastError`.

The send Function

```
// Send an initial buffer
iResult = send( ConnectSocket, sendbuf, (int)strlen(sendbuf), 0 );
if (iResult == SOCKET_ERROR) {
    wprintf(L"send failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
printf("Bytes Sent: %d\n", iResult);

// shutdown the connection since no more data will be sent
iResult = shutdown(ConnectSocket, SD_SEND);
if (iResult == SOCKET_ERROR) {
    wprintf(L"shutdown failed with error: %d\n", WSAGetLastError());
    closesocket(ConnectSocket);
    WSACleanup();
    return 1;
}
```

The `recv` Function

- The `recv` function receives data from a connected socket or a bound connectionless socket.

```
int recv(
  _In_  SOCKET s,
  _In_  char *buf,
  _In_  int len,
  _In_  int flags
);
```

`s` [in]
`buf` [in]
`len` [in]
`flag` [in]
Differs from `flag` parameter in `send`.

The `recv` Function

- Parameter `flag`

Value	Meaning
`0`	Default.
`MSG_PEEK`	Peeks at the incoming data. The data is copied into the buffer, but is not removed from the input queue. The function subsequently returns the amount of data that can be read in a single call to the `recv` (or `recvfrom`) function, which may not be the same as the total amount of data queued on the socket.
`MSG_OOB`	Processes Out Of Band (OOB) data.
`MSG_WAITALL`	The receive request will complete only when one of the following events occurs: • The buffer supplied by the caller is completely full. • The connection has been closed. • The request has been canceled or an error occurred. This flag is not supported on datagram sockets or message-oriented sockets.

The `recv` Function

- **Return value**
  - **If no error occurs, `recv` returns the number of bytes received and the buffer pointed to by the `buf` parameter will contain this data received. If the connection has been gracefully closed, the return value is zero.**

```
// Receive until the peer closes the connection
do {

    iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
    if ( iResult > 0 )
        wprintf(L"Bytes received: %d\n", iResult);
    else if ( iResult == 0 )
        wprintf(L"Connection closed\n");
    else
        wprintf(L"recv failed with error: %d\n", WSAGetLastError());

} while( iResult > 0 );
```

The sendto/recvfrom Function

- The send/recv for UDP.

```
int sendto(
  _In_   SOCKET s,
  _In_   const char *buf,
  _In_   int len,
  _In_   int flags,
  _In_   const struct sockaddr *to,
  _In_   int tolen
);

int recvfrom(
  _In_          SOCKET s,
  _Out_         char *buf,
  _In_          int len,
  _In_          int flags,
  _Out_         struct sockaddr *from,
  _Inout_opt_   int *fromlen
);
```

The sendto Function

```
    // Initialize Winsock
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != NO_ERROR) { ... }
    // Create a socket for sending data
    SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (SendSocket == INVALID_SOCKET) { ... }
    // Set up the RecvAddr structure with the IP address of
    // the receiver (in this example case "192.168.1.1")
    // and the specified port number.
    RecvAddr.sin_family = AF_INET;
    RecvAddr.sin_port = htons(Port);
    RecvAddr.sin_addr.s_addr = inet_addr("192.168.1.1");
    // Send a datagram to the receiver
    wprintf(L"Sending a datagram to the receiver...\n");
    iResult = sendto(SendSocket, SendBuf, BufLen, 0, (SOCKADDR *) & RecvAddr,
sizeof (RecvAddr));
    if (iResult == SOCKET_ERROR) { ... }
    // When the application is finished sending, close the socket.
    wprintf(L"Finished sending. Closing socket.\n");
    iResult = closesocket(SendSocket);
    if (iResult == SOCKET_ERROR) { ... }
```

The recvfrom Function

```
    // Initialize Winsock
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != NO_ERROR) { ... }
    // Create a receiver socket to receive datagrams
    RecvSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (RecvSocket == INVALID_SOCKET) { ... }
    // Bind the socket to any address and the specified port.
    RecvAddr.sin_family = AF_INET;
    RecvAddr.sin_port = htons(Port);
    RecvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    iResult = bind(RecvSocket, (SOCKADDR *) & RecvAddr, sizeof (RecvAddr));
    if (iResult != 0) { ... }
    // Call the recvfrom function to receive datagrams on the bound socket.
    wprintf(L"Receiving datagrams...\n");
    iResult = recvfrom(RecvSocket, RecvBuf, BufLen, 0, (SOCKADDR *) & SenderAddr,
&SenderAddrSize);
    if (iResult == SOCKET_ERROR) { ... }
    // Close the socket when finished receiving datagrams
    wprintf(L"Finished receiving. Closing socket.\n");
    iResult = closesocket(RecvSocket);
    if (iResult == SOCKET_ERROR) { ... }
```

The WSAStartup Function

- The WSAStartup function initiates use of the Winsock DLL by a process.
  - The current version of the Windows Sockets specification is version 2.2.

```
int WSAStartup(
  _In_   WORD wVersionRequested,
  _Out_  LPWSADATA lpWSAData
);
```

wVersionRequested [in]
The highest version of Windows Sockets specification that the caller can use. The high-order byte specifies the minor version number; the low-order byte specifies the major version number.

lpWSAData [out]
A pointer to the **WSADATA** data structure that is to receive details of the Windows Sockets implementation.

The WSAStartup Function

- 假如一个程序要使用2.1版本的Socket,那么程序代码如下

```
#define MAKEWORD(a, b)  ((WORD)(((BYTE)(((DWORD_PTR)(a)) & 0xff)) | \
                              ((WORD)((BYTE)(((DWORD_PTR)(b)) & 0xff))) << 8))

wVersionRequested = MAKEWORD( 2, 1 );
err = WSAStartup( wVersionRequested, &wsaData );
```

The `closesocket` Function

- The `closesocket` function closes an existing socket.

```
int closesocket(
  _In_  SOCKET s
);
```

s [in]
A descriptor identifying the socket to close.

简单实例讲解

结合一个客户端/服务器端的例子

服务器端编程

```
/* server.c - code for example server program that uses TCP */
#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT       5193            /* default protocol port number */
#define QLEN            6               /* size of request queue        */

int     visits      =   0;              /* counts client connections    */
```

平台间差异

服务器端编程

```
/*------------------------------------------------------------------
 * Program:   server
 *
 * Purpose:   allocate a socket and then repeatedly execute the following:
 *              (1) wait for the next connection from a client
 *              (2) send a short message to the client
 *              (3) close the connection
 *              (4) go back to step (1)
 *
 * Syntax:    server [ port ]
 *
 *               port  - protocol port number to use
 *
 * Note:      The port argument is optional.  If no port is specified,
 *            the server uses the default given by PROTOPORT.
 *
 *------------------------------------------------------------------
 */
main(argc, argv)
int     argc;
char    *argv[];
```

服务器端编程

```
{
        struct  hostent  *ptrh;  /* pointer to a host table entry       */
        struct  protoent *ptrp;  /* pointer to a protocol table entry   */
        struct  sockaddr_in sad; /* structure to hold server's address  */
        struct  sockaddr_in cad; /* structure to hold client's address  */
        int     sd, sd2;         /* socket descriptors                  */
        int     port;            /* protocol port number                */
        int     alen;            /* length of address                   */
        char    buf[1000];       /* buffer for string the server sends  */

#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET;           /* set family to Internet     */
        sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address   */
        /* Check command-line argument for protocol port and extract    */
        /* port number if one is specified.  Otherwise, use the default */
        /* port value given by constant PROTOPORT                       */
```

WSAStartup应为第一个调用的Windows Socket函数，用于定义版本号。

服务器端编程

```
        if (argc>1 && argv[1][0]=='?')
        {
                printf("Program:   server\n");
                printf("Purpose:   allocate a socket and then
repeatedly execute the following:\n");
                printf("             (1) wait for the next connection
from a client\n");
                printf("             (2) send a short message to the
client\n");
                printf("             (3) close the connection\n");
                printf("             (4) go back to step (1)\n");
                printf("Syntax:    server [ port ]\n");
                printf("             port  - protocol port number to
use\n");
                printf("Note:      The port argument is optional. If
no port is specified,\n");
                printf("           the server uses the default given
by PROTOPORT.\n");
                exit(1);
        }
```

服务器端编程

```
if (argc > 1) {                         /* if argument specified        */
        port = atoi(argv[1]);           /* convert argument to binary   */
} else {
        port = PROTOPORT;               /* use default port number      */
}
if (port > 0)                           /* test for illegal value       */
        sad.sin_port = htons((u_short)port);
else {                                  /* print error message and exit */
        fprintf(stderr,"bad port number %s\n",argv[1]);
        exit(1);
}

/* Map TCP transport protocol name to

if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
}
```

顺序转换到TCP/IP网络

Getprotobyname 检索协议名称对应的的协议信息。

服务器端编程

```
        /* Create a socket */
        sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
        if (sd < 0) {
                fprintf(stderr, "socket creation failed\n");
                exit(1);
        }

        /* Bind a local addre
        if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
                fprintf(stderr,"bind failed\n");
                exit(1);
        }

        /* Specify size of request queue */
        if (listen(sd, QLEN) < 0) {
                fprintf(stderr,"listen failed\n");
                exit(1);
        }
```

协议族 IPv4

新套接字的类型
供了序列、可靠、双向，基于连接的带OOB数据传输机制的字节流。

协议类型

识别未绑定的套接字的描述符。

socket函数创建一个绑定到一个特定运输服务提供商的套接字。

对套接字赋值的本地地址指针

bind函数将本地地址与套接字相关联。

队列长度

listen将套接字置于侦听传入连接的状态之下

服务器端编程

一个死循环

```
        /* Main server loop - accept and handle requests */

        while (1) {
                alen = sizeof(cad);
                if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
                        fprintf(stderr, "accept fail
                        exit(1);
                }
                visits++;
                sprintf(buf,"This server has been contacted %d time%s\n",
                        visits,visits==1?".":"s.");
                send(sd2,buf,strlen(buf),0);
                closesocket(sd2);
        }
}
```

Accept函数允许在套接字上传入的连接尝试。

可选的指针，指向到一个缓冲区，它接收连接实体的地址，如已知的通信层。

Send函数向一个已连接的套接字发送数据。

关闭套接字。
注意是哪个套接字

客户端编程

```
/* client.c - code for example client program that uses TCP */

#ifndef unix
#define WIN32
#include <windows.h>
#include <winsock.h>
#else
#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#endif

#include <stdio.h>
#include <string.h>

#define PROTOPORT       5193            /* default protocol port number */
extern  int             errno;
char    localhost[] =   "localhost";    /* default host name            */
```

客户端编程

```
/*------------------------------------------------------------------
 * Program:   client
 *
 * Purpose:   allocate a socket, connect to a server, and print all output
 *
 * Syntax:    client [ host [port] ]
 *
 *               host  - name of a computer on which server is executing
 *               port  - protocol port number server is using
 *
 * Note:      Both arguments are optional.  If no host name is specified,
 *            the client uses "localhost"; if no protocol port is
 *            specified, the client uses the default given by PROTOPORT.
 *
 *------------------------------------------------------------------
 */
main(argc, argv)
int     argc;
char    *argv[];
```

客户端编程

```
{
        struct  hostent  *ptrh;  /* pointer to a host table entry       */
        struct  protoent *ptrp;  /* pointer to a protocol table entry   */
        struct  sockaddr_in sad; /* structure to hold an IP address     */
        int     sd;              /* socket descriptor                   */
        int     port;            /* protocol port number                */
        char    *host;           /* pointer to host name                */
        int     n;               /* number of characters read           */
        char    buf[1000];       /* buffer for data from the server     */
#ifdef WIN32
        WSADATA wsaData;
        WSAStartup(0x0101, &wsaData);
#endif
        memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
        sad.sin_family = AF_INET;         /* set family to Internet     */

        /* Check command-line argument for protocol port and extract    */
        /* port number if one is specified.  Otherwise, use the default */
        /* port value given by constant PROTOPORT                       */
```

客户端编程

```
        if (argc>1 && argv[1][0]=='?')
        {
                printf("Program:   client\n");
                printf("Purpose:   allocate a socket, connect to a
server, and print all output\n");
                printf("Syntax:    client [ host [port] ]\n");
                printf("             host - name of a computer on
which server is executing\n");
                printf("             port - protocol port number
server is using\n");
                printf("Note:      Both arguments are optional. If no
host name is specified,\n");
                printf("           the client uses \"localhost\"; if
no protocol port is\n");
                printf("           specified, the client uses the
default given by PROTOPORT.\n");
                exit(1);
        }
```

客户端编程

```
if (argc > 2) {                    /* if protocol port specified   */
        port = atoi(argv[2]);      /* convert to binary            */
} else {
        port = PROTOPORT;          /* use default port number      */
}
if (port > 0)                      /* test for legal value         */
        sad.sin_port = htons((u_short)port);
else {                             /* print error message and exit */
        fprintf(stderr,"bad port number %s\n",argv[2]);
        exit(1);
}

/* Check host argument and assign host name. */

if (argc > 1) {
        host = argv[1];            /* if host argument specified   */
} else {
        host = localhost;
}
```

客户端编程

```
/* Convert host name to equivalent IP address and copy to sad. */
ptrh = gethostbyname(host);
if ( ((char *)ptrh) == NULL ) {
        fprintf(stderr,"invalid host: %s\n", host);
        exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

/* Map TCP transport protocol name to protocol number. */
if ( ((int)(ptrp = getprotobyname("tcp"))) == 0) {
        fprintf(stderr, "cannot map \"tcp\" to protocol number");
        exit(1);
}

/* Create a socket. */

sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
if (sd < 0) {
        fprintf(stderr, "socket creation failed\n");
        exit(1);
}
```

从主机名获取指向主机的指针

客户端编程

```
        /* Connect the socket to the specified server. */

        if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
                fprintf(stderr,"connect failed
                exit(1);
        }

        /* Repeatedly read data from socket and write to user's screen. */

        n = recv(sd, buf, sizeof(buf), 0);
        while (n > 0) {
                write(1,buf,n);
                n = recv(sd, buf, sizeof(buf), 0);
        }

        /* Close the socket. */
        closesocket(sd);

        /* Terminate the client program gracefully. */
        exit(0);
}
```

recv函数从连接的套接字或无连接的套接字绑定接收数据。

Microsoft .Net Framework

- System.Net.Sockets 命名空间
- 详见UDP课的PPT

C++实例

```
#using <System.dll>

using namespace System;
using namespace System::Text;
using namespace System::IO;
using namespace System::Net;
using namespace System::Net::Sockets;
String^ DoSocketGet( String^ server )
{

   //Set up variables and String to write to the server.
   Encoding^ ASCII = Encoding::ASCII;
   String^ Get =  "GET / HTTP/1.1\r\nHost: ";
   Get->Concat( server,  "\r\nConnection: Close\r\n\r\n" );
   array<Byte>^ByteGet = ASCII->GetBytes( Get );
   array<Byte>^RecvBytes = gcnew array<Byte>(256);
   String^ strRetPage = nullptr;

   // IPAddress and IPEndPoint represent the endpoint that will
   //   receive the request.
   // Get first IPAddress in list return by DNS.
```

C++实例

```
try
{
   // Define those variables to be evaluated in the next for loop and
   // then used to connect to the server. These variables are defined
   // outside the for loop to make them accessible there after.
   Socket^ s = nullptr;
   IPEndPoint^ hostEndPoint;
   IPAddress^ hostAddress = nullptr;
   int conPort = 80;

   // Get DNS host information.
   IPHostEntry^ hostInfo = Dns::Resolve( server );

   // Get the DNS IP addresses associated with the host.
   array<IPAddress^>^IPaddresses = hostInfo->AddressList;

   // Evaluate the socket and receiving host IPAddress and IPEndPoint.
   for ( int index = 0; index < IPaddresses->Length; index++ )
   {
      hostAddress = IPaddresses[ index ];
```

C++实例

```
        hostEndPoint = gcnew IPEndPoint( hostAddress,conPort );

        // Creates the Socket to send data over a TCP connection.
        s = gcnew Socket( AddressFamily::InterNetwork, SocketType::Stream,
ProtocolType::Tcp );

        // Connect to the host using its IPEndPoint.
        s->Connect( hostEndPoint );
        if (  !s->Connected )
        {
           // Connection failed, try next IPaddress.
           strRetPage =  "Unable to connect to host";
           s = nullptr;
           continue;
        }

        // Sent the GET request to the host.
        s->Send( ByteGet, ByteGet->Length, SocketFlags::None );
     }
```

C++实例

```
      // Receive the host home page content and loop until all the data is
received.
      Int32 bytes = s->Receive( RecvBytes, RecvBytes->Length,
SocketFlags::None );
      strRetPage =  "Default HTML page on ";
      strRetPage->Concat( server,  ":\r\n", ASCII->GetString( RecvBytes, 0,
bytes ) );
      while ( bytes > 0 )
      {
         bytes = s->Receive( RecvBytes, RecvBytes->Length, SocketFlags::None );
         strRetPage->Concat( ASCII->GetString( RecvBytes, 0, bytes ) );
      }
   }
   catch ( SocketException^ e )
   {
      Console::WriteLine(  "SocketException caught!!!" );
      Console::Write(  "Source : " );
      Console::WriteLine( e->Source );
      Console::Write(  "Message : " );
      Console::WriteLine( e->Message );
   }
```

C++实例

```
catch ( ArgumentNullException^ e )
{
   Console::WriteLine(  "ArgumentNULLException caught!!!" );
   Console::Write(  "Source : " );
   Console::WriteLine( e->Source );
   Console::Write(  "Message : " );
   Console::WriteLine( e->Message );
}
catch ( NullReferenceException^ e )
{
   Console::WriteLine(  "NULLReferenceException caught!!!" );
   Console::Write(  "Source : " );
   Console::WriteLine( e->Source );
   Console::Write(  "Message : " );
   Console::WriteLine( e->Message );
}
```

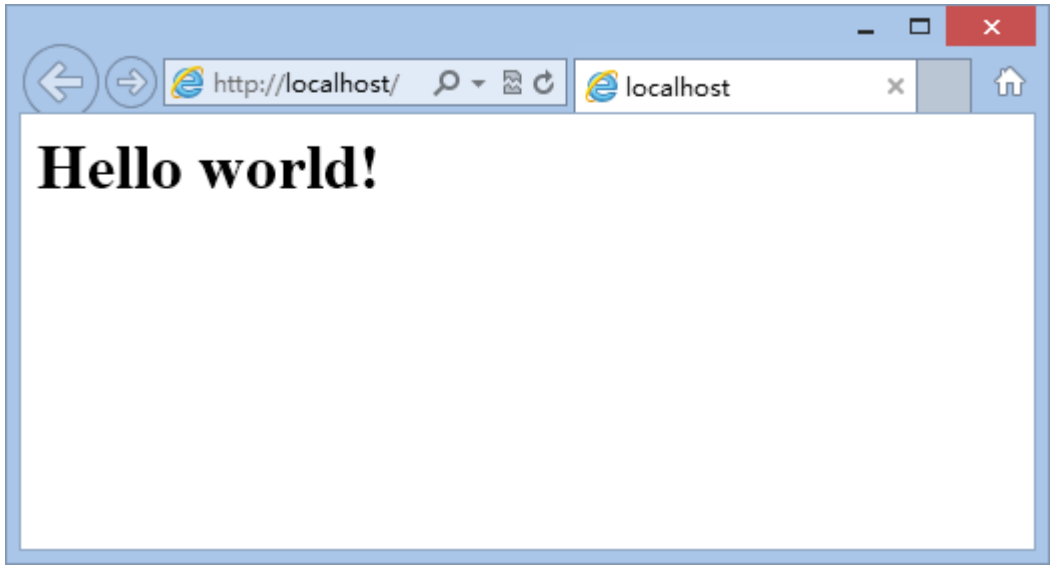
C++实例

```
 catch ( Exception^ e )
    {
       Console::WriteLine(  "Exception caught!!!" );
       Console::Write(  "Source : " );
       Console::WriteLine( e->Source );
       Console::Write(  "Message : " );
       Console::WriteLine( e->Message );
    }

    return strRetPage;
}

int main()
{
    Console::WriteLine( DoSocketGet(  "localhost" ) );
}
```

C++实例：输出

```
Default HTML page on 127.0.0.1:
HTTP/1.1 200 OK
Server: ASP.NET Development Server/11.0.0.0
Date: Mon, 29 Apr 2013 09:54:33 GMT
X-AspNet-Version: 4.0.30319
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 25
Connection: Close
```

```
<h1>Hello world!</h1>
```



思考题

- 一个应用可以对应几个IP，几个端口？
- 一个端口可以给几个应用使用？
- 能不能乱用官方规定的端口？

14. THANK YOU.

厦门大学软件学院

黄炜 助理教授