计算机网络
Computer Network

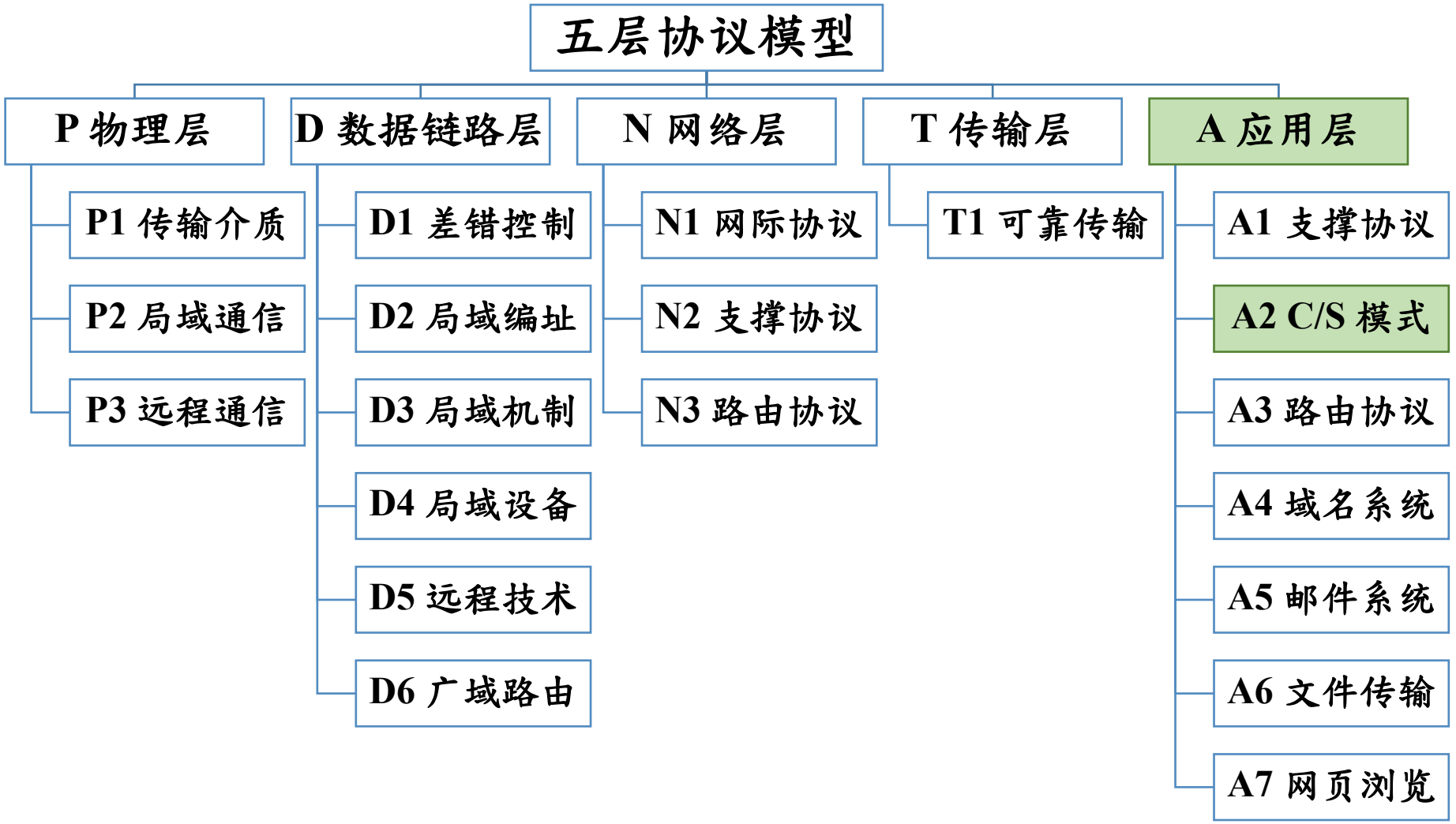
14

客户服务器模式和套接字API

理论课程

厦門大學 XIAMEN UNIVERSITY
信息学院（国家示范性软件学院） School of Informatics
黄 炜 博士，副教授 Dr. Wei Huang

知识框架


五层协议模型
P 物理层
D 数据链路层
N 网络层
T 传输层
A 应用层
P1 传输介质
P2 局域通信
P3 远程通信
D1 差错控制
D2 局域编址
D3 局域机制
D4 局域设备
D5 远程技术
D6 广域路由
N1 网际协议
N2 支撑协议
N3 路由协议
T1 可靠传输
A1 支撑协议
A2 C/S 模式
A3 路由协议
A4 域名系统
A5 邮件系统
A6 文件传输
A7 网页浏览

主要内容

- 客户端—服务器端（C/S）交互模式工作原理
- 并发的概念
- Socket结构、半相关与全相关
- 服务器与用户、服务器端与客户端，二者区别
- Socket API主要函数（Windows或Linux系统下）
- 流模式的客户端、服务器端Socket API调用流程
- 报文模式的客户端、服务器端Socket API调用流程

对应课本章节

- **PART I Introduction And Internet Applications**
  - **Chapter 3 Internet Applications And Network Programming**

内容纲要

客户和服务器

- 应用层的许多协议都是基于客户/服务器方式。
  - 将一项任务划分为服务的请求者和服务的提供者两部分。
- 客户和服务器都是指通信中所涉及的两个应用进程。
  - 客户（Client）和服务（Server）默认指客户端和服务器端。
  - 客户端是访问服务器提供的服务的程序。
  - 服务器是为客户程序或设备提供功能的计算机程序或设备。
- 当传入消息与应用程序指定的端口相匹配时，协议软件将该消息传递给应用程序。

客户和服务器

- 客户端或服务器端是指一个程序。
  - 客户和服务器默认指客户端和服务器端。
- 客户软件和服务器软件是指一个软件。
  - 软件由多个程序和数据共同完成功能。
  - 这些程序有客户端程序，也有服务器端程序。
- 客户设备和服务器设备是指一个硬件。
  - 硬件上可以运行多个软件。
  - 这些软件有客户软件，也有服务器软件。

客户端和服务器端工作模式

- 客户端
  - 主动打开：客户端使用远程主机的地址和在该机上特定服务器程序的著名端口地址通道打开通信。
  - 主动结束：虽然请求的响应可重复几次，但整个过程是有限的，最终结束。在那一刻，客户端关闭通信通道。
- 服务器端
  - 被动打开：服务器像被动打开的大门，响应客户端的请求。
  - 响应请求：当请求到达时，它会作出迭代的或同步的响应。

客户端软件（Client software）

- 客户端软件（Client software）
  - 任意的应用程序，需要远程访问时成为一个临时的客户端，但也可以在本地执行其他计算程序。
  - 直接由用户调用，并只执行一次会话。
  - 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。
  - 本地用户的主机上运行，不需特殊硬件或复杂的操作系统。

服务器端软件（Server software）

- 服务器端软件（Server software）
  - 一个专用于提供一个服务的专用程序，同时可以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。

客户和服务器的区别

- 轻重区别
  - 服务器是一个系统（硬件和软件），它响应计算机网络上的请求以提供或帮助提供网络服务。
  - 客户端是一个计算机硬件或软件访问服务器提供的服务的一块。
  - 客户–服务器（C/S）模式是一种分布式的应用结构计算，在任务或工作负载的资源或服务的提供者（服务器）和服务请求者（客户）之间。

一台计算机上的多种服务

- TCP / UDP复用与解复用
  - 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连到互联网。
  - 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哪一应用层程序接收数据报。
  - 相应地，UDP类似。

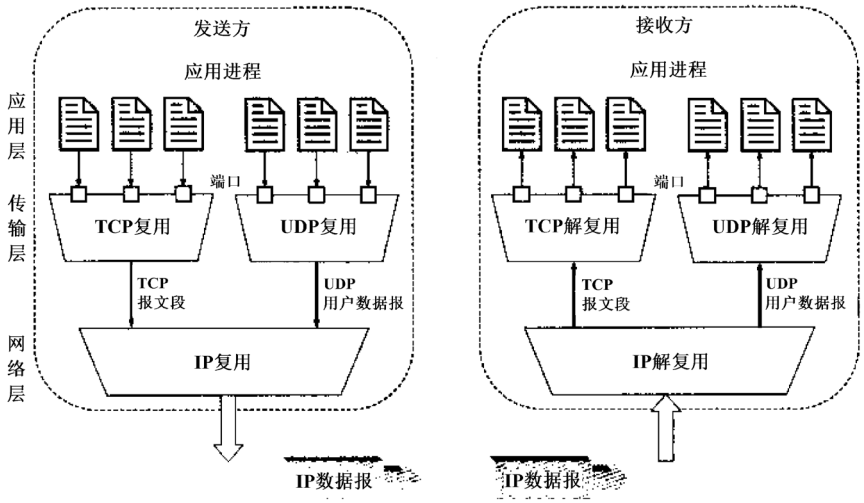


图 7-7　端口在进程之间的通信中所起的作用

一台计算机上的多种服务

- 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。

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

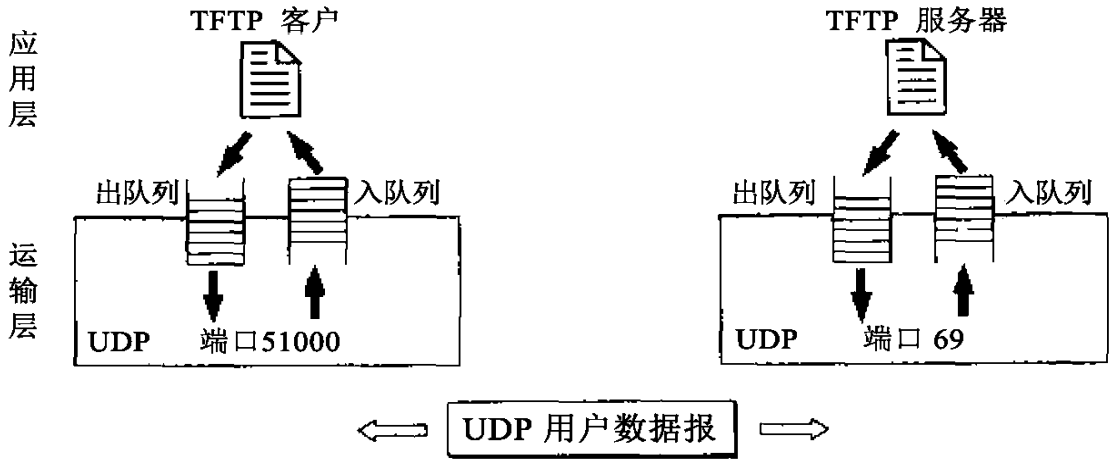
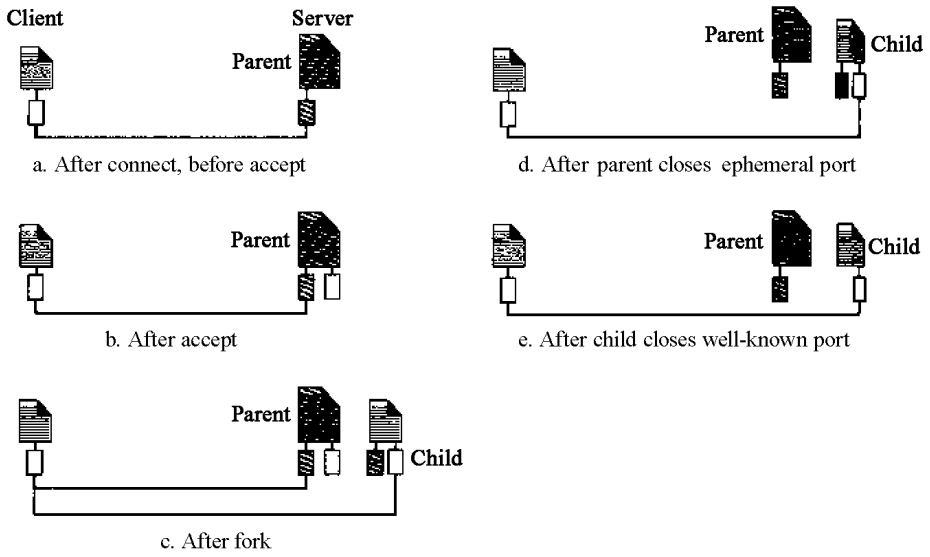


图7-9　UDP中的队列

并发（Concurrency）

- 并发是客户端-服务器交互模式的基础。
  - 允许多个应用在同一时间执行的计算机系统称为支持并发。
  - 一个拥有多个线程的程序称为并发程序。
  - 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。



内容纲要

系统调用和应用编程接口

- 大多数操作系统使用系统调用(system call)的机制在应用程序和操作系统之间传递控制权。
- 对程序员来说，每一个系统调用和一般程序设计中的函数调用非常相似，只是系统调用是将控制权传递给了操作系统。
- Application Program Interface（API）

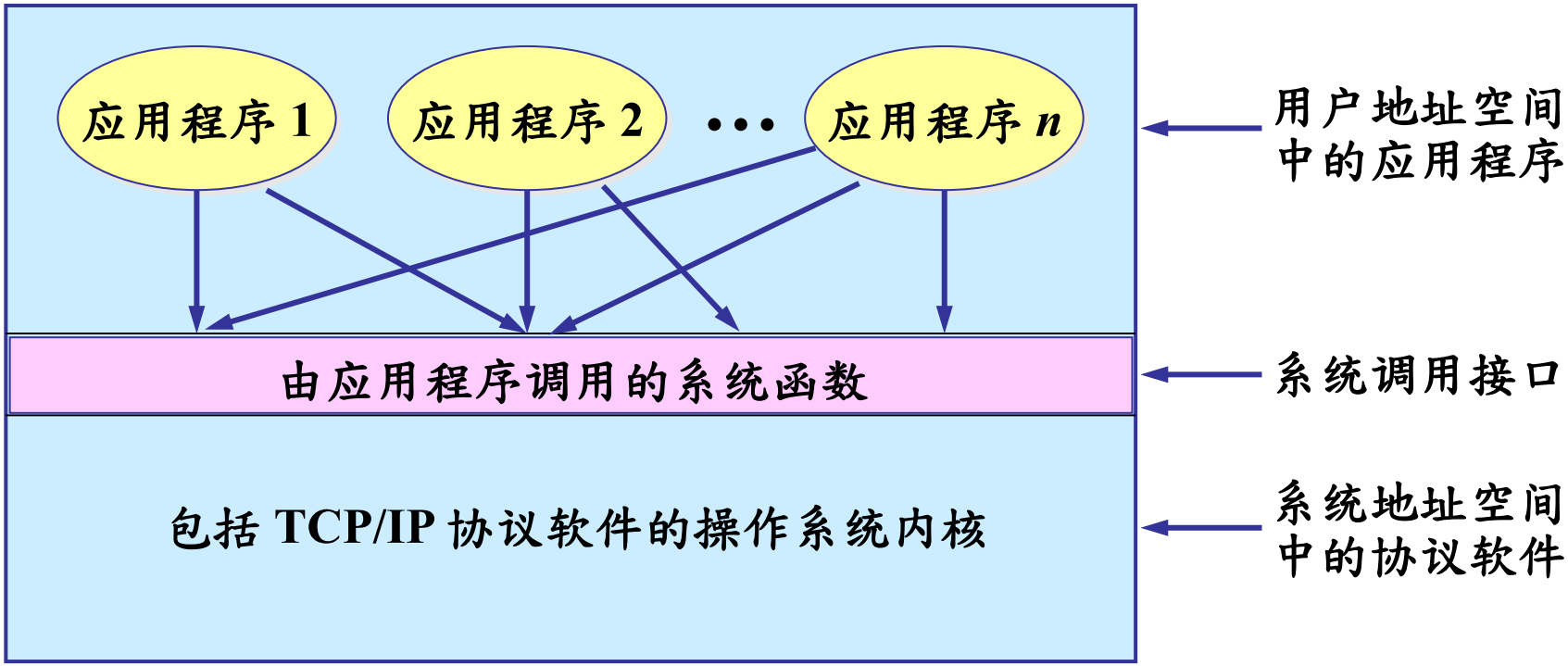
单机进程通信

- 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。

多个应用进程使用系统调用的机制



协议端口

- 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（包括数据结构和I/O缓冲区）。
  - 网络通信的最终地址就不仅是主机地址，还包括端口号。
  - 每个端口有端口号（Port Number），区别不同端口。
    - TCP有一个255号端口，UDP也有255号端口，常用端口号要记住
  - TCP/IP协议实现中，端口间操作类似一般I/O操作，进程获取端口，相当于获取本地I/O文件，可用一般读写原语访问。

套接字（Socket）

- 当应用进程需要使用网络进行通信时就发出系统调用，请求操作系统为其创建“套接字”，以便把网络通信所需要的系统资源分配给该应用进程。
- 操作系统为这些资源的总和用一个叫做套接字描述符的号码来表示，并把此号码返回给应用进程。应用进程所进行的网络操作都必须使用这个号码。
- 通信完毕后，应用进程通过一个关闭套接字的系统调用通知操作系统回收与该“号码”相关的所有资源。

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

连接建立阶段

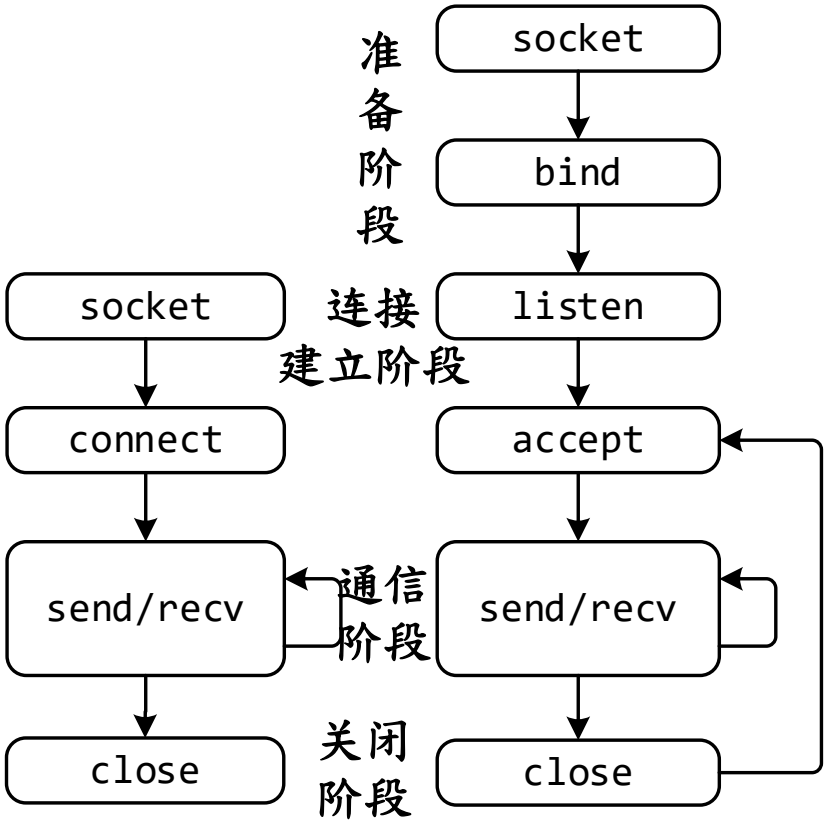
- 当套接字被创建后，它的端口号和 IP 地址都是空的，因此应用进程要调用 bind（绑定）来指明套接字的本地地址。在服务器端调用 bind 时就是把熟知端口号和本地IP地址填写到已创建的套接字中。这就叫做把本地地址绑定到套接字。
- 服务器在调用 bind 后，还必须调用 listen（监听）把套接字设置为被动方式，以便随时接受客户的服务请求。

连接建立阶段

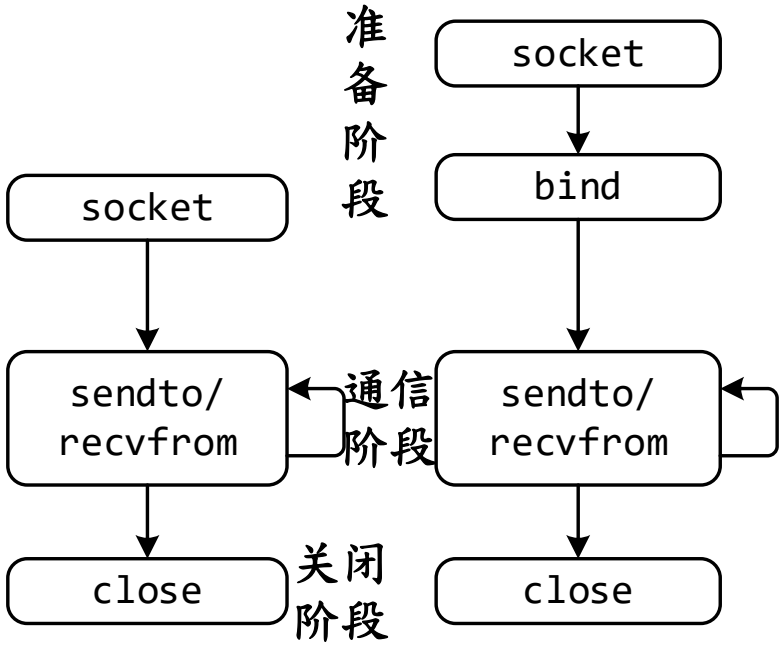
- UDP服务器由于只提供无连接服务，不使用 listen 系统调用。
- 服务器紧接着就调用 accept（接受），以便把远地客户进程发来的连接请求提取出来。系统调用 accept 的一个变量就是要指明从哪一个套接字发起的连接。

C/S中的套接字

- TCP面向连接



UDP无连接



内容纲要

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）。

Socket

- 本地套接字地址
  - 本地IP地址
  - 本地端口号
- 远程套接字地址
  - 远程IP地址
  - 远程端口号
- 协议
  - TCP, UDP, 原始(raw) IP

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`

函数 socket

- 函数 socket 创建一个绑定到特定传输服务提供者的插口。

```
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.

函数 socket

- 参数 1：地址族

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.

函数 socket

- 参数 2：插口类型

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.

函数 socket

- 参数 3：协议

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.

函数 socket

- 返回值
  - **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`.**
- 高级内容
  - **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.**

函数 bind

- 函数 bind 将本地地址关联到插口上。

```
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.

结构体 sockaddr

- **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.

函数 bind

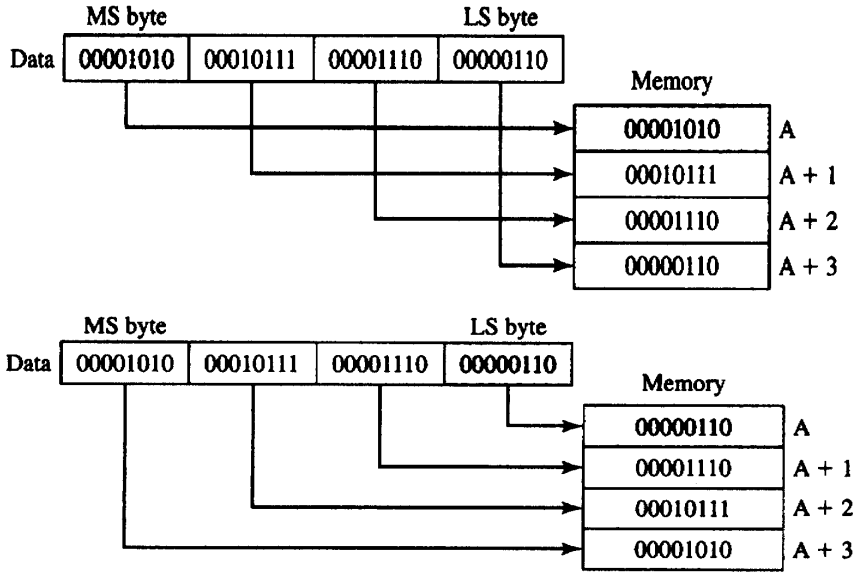
```
// Declare variables
SOCKET ListenSocket;
struct sockaddr_in saServer;
hostent* localHost;
char* localIP;

// 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) );
```

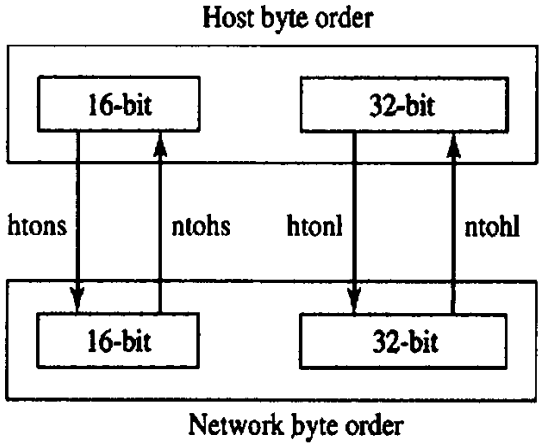
网络字节顺序（htons）

- 字节顺序
  - 大端序（Big-endian）和小端序（Little-endian byte order）
  - 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

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
- 转换：(host,network)to(h,n)(short, long)





函数 connect

- 函数 connect 建立到一个给定套接字的链接。

```
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.

函数 connect

```
    // 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");
```

函数 listen

- 函数 listen 将套接字置于侦听传入连接的状态中。

```
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.

函数 accept

- 函数 accept 允许套接字上的传入连接尝试。

```
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.

函数 accept

- 返回值
  - 如果没有发生错误，则接受返回SOCKET类型的值，该套接字是新套接字的描述符。此返回值是实际连接的套接字的句柄。
  - 否则，一个价值INVALID_SOCKET返回一个特定的错误代码可以通过调用WSAGetLastError检索。

函数 accept

```
// 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");
```

函数 send

- 函数 send 将数据发送到一个已连接的套接字。

```
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.

函数 send

- 参数 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.

- 返回值
  - 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.

函数 send

```
// 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;
}
```

函数 recv

- 函数 recv 从连接套接字或绑定的无连接套接字接收数据。

```
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.

函数 recv

- 参数 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.**

函数 recv

- 返回值
  - **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 );
```

函数 sendto/recvfrom

- **UDP 的 send/recv**

```
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
);
```

函数 sendto

```
    // 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) { ... }
```

函数 recvfrom

```
    // 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) { ... }
```

函数 WSAStartup

- 函数 WSAStartup 启动使用Winsock DLL的过程。
  - 当前版本的Windows套接字规范是版本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.

函数 WSAStartup

- 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 );
```

函数 closesocket

- 函数 closesocket 关闭已经存在的插口.

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

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

内容纲要

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 protocol number */

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

htons将u_short从主机顺序转换到TCP/IP网络字节顺序（大端）。

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 address to the socket */

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

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

识别未绑定的套接字的描述符。

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

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

        while (1) {
                alen = sizeof(cad);
                if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
                        fprintf(stderr, "accept failed\n");
                        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);
        }
}
```

一个死循环

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\n");
                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 );
}
```

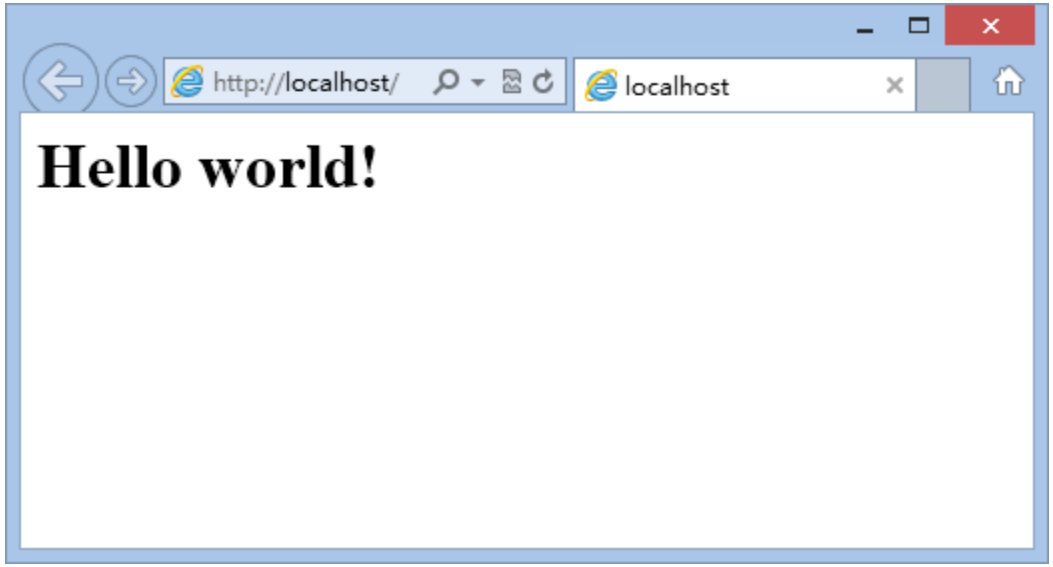
C++实例

```
    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>
```



计 算 机 网 络
Computer Network

谢谢观看
Thank you

14

客户服务器模式和套接字API

理论课程

厦門大學
XIAMEN UNIVERSITY

信息学院 黄 炜
（国家示范性软件学院） 博士，副教授
School of Informatics Dr. Wei Huang