重庆邮电大学

Windows环境的网络编程

- Windows Sockets 规范
- WinSock 规范与Berkeley套接口的区别
- Winsock 1.1 的库函数。

3.1 Windows Sockets规范

- 3.1.1 概述
  - Microsoft公司以Berkeley Sockets规范为范例，定义了Windows Socktes规范，简称Winsock规范。这是Windows操作系统环境下的套接字网络应用程序编程接口（API）。

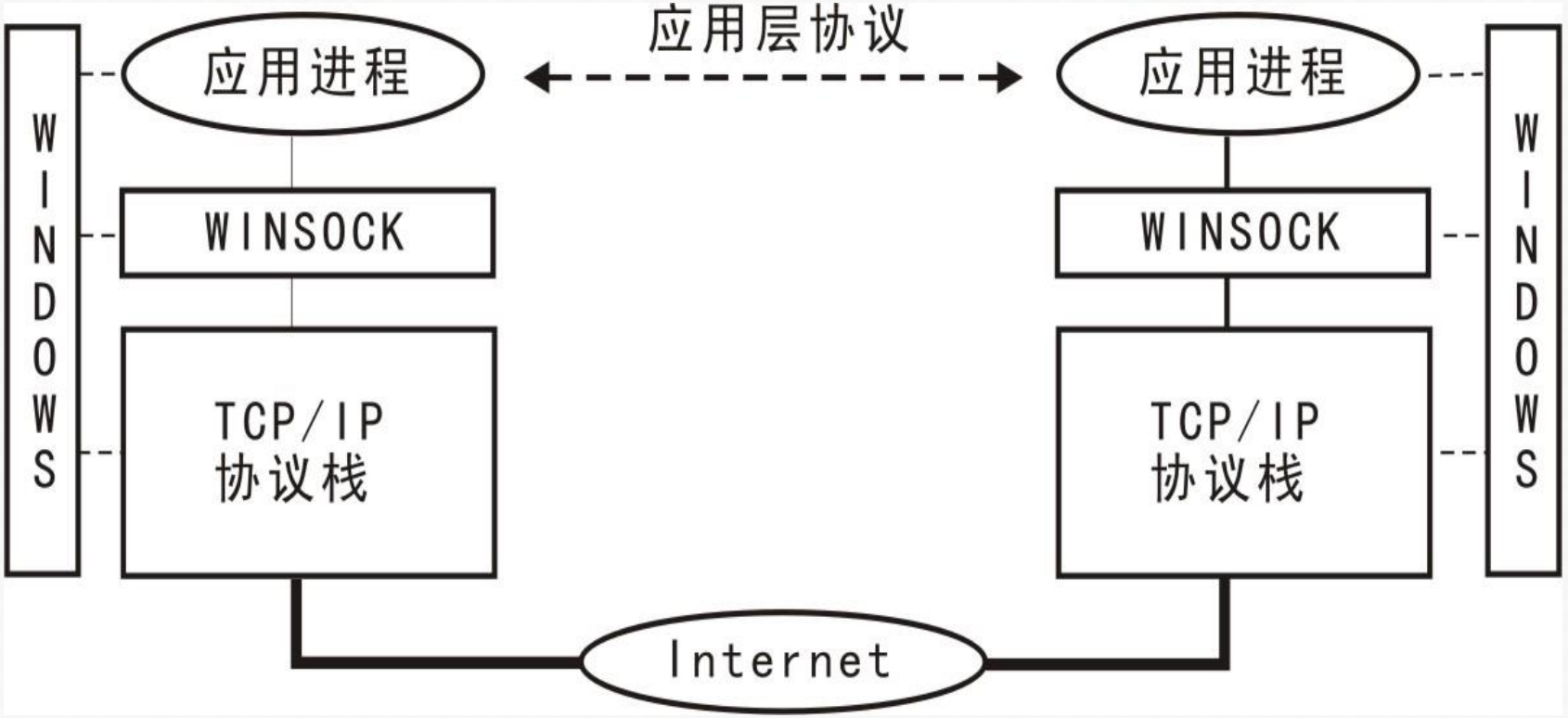

图3.1 网络应用进程利用WinSock进行通信

3.1 Windows Sockets规范

- 3.1.2 Windows Sockets规范
  - Windows Sockets 规范是一套开放的、支持多种协议的Windows下的网络编程接口。从1991年到1995年，从1.0版发展到2.0.8版，已成为Windows网络编程的事实上的标准。
  - 1. Windows Sockets 1.1版本
    - 在Winsock.h包含文件中，定义了所有WinSock 1.1版本库函数的语法、相关的符号常量和数据结构。库函数的实现在WINSOCK.DLL动态链接库文件中。

3.1 Windows Sockets规范

- 3.1.2 Windows Sockets规范
  - 1. Windows Sockets 1.1版本
    - （1）WinSock 1.1 全面继承了Berkeley Sockets规范，见表 3.1
    - （2）数据库函数
      - 表3.2列出了Winsock规范定义的数据库查询例程。其中六个采用getXbyY()的形式，大多要借助网络上的数据库来获得信息
    - （3）WinSock 1.1 扩充了Berkeley Sockets规范
      - 针对微软 Windows的特点，WinSock 1.1定义了一批新的库函数，提供了对于消息驱动机制的支持，有效地利用Windows多任务多线程的机制。见表3.3
    - （4）WinSock 1.1只支持TCP/IP协议栈

3.1 Windows Sockets规范

- 3.1.2 Windows Sockets规范
  - 2. WinSock 2.0
    - WinSock 2.0在源码和二进制代码方面与WinSock 1.1兼容，
    - WinSock 2.0增强了许多功能。
      - （1）支持多种协议
      - （2）引入了重叠I/O的概念
      - （3）使用事件对象异步通知
      - （4）服务的质量（QOS）
      - （5）套接口组
      - （6）扩展的字节顺序转换例程
      - （7）分散/聚集方式I/O
      - （8）新增了许多函数

3.1 Windows Sockets规范

- 3.1.2 Windows Sockets规范
  - 3．WinSock 1.1中的阻塞问题
    - 阻塞是在把应用程序从Berkeley套接口环境中移植到Windows环境中的一个主要焦点。阻塞是指唤起一个函数，该函数直到相关操作完成时才返回。
    - 在Berkeley套接口模型中，一个套接口的操作的缺省行为是阻塞方式的，除非程序员显式地请求该操作为非阻塞方式。
    - 在Windows环境下，我们强烈推荐程序员在尽可能的情况下使用非阻塞方式（异步方式）的操作。因为非阻塞方式的操作能够更好地在非占先的Windows环境下工作。

3.1 Windows Sockets规范

- 3.1.3 WinSock规范与Berkeley套接口的区别
  - 1. 套接口数据类型和该类型的错误返回值
    - 在UNIX中，包括套接口句柄在内的所有句柄，都是非负的短整数，
    - 在WinSock规范中定义了一个新的数据类型，称作SOCKET，用来代表套接字描述符。
      - typedef u_int SOCKET;
    - SOCKET可以取从0到INVALID_SOCKET-1之间的任意值。
  - 2. select()函数和FD_*宏
    - 在Winsock中，使用select()函数时，应用程序应坚持用FD_XXX宏来设置，初始化，清除和检查fd_set结构。

3.1 Windows Sockets规范

- 3.1.3 WinSock规范与Berkeley套接口的区别
  - **3. 错误代码的获得**
    - 在UNIX 套接字规范中，如果函数执行时发生了错误，会把错误代码放到errno或h_errno变量中。
    - 在Winsock中，错误代码可以使用WSAGetLastError()调用得到。
  - **4. 指针**
    - 所有应用程序与Windows Sockets使用的指针都必须是FAR指针。
  - **5. 重命名的函数**
    - （1）close()改变为closesocket()
    - （2）ioctl()改变为ioctlsocket()

3.1 Windows Sockets规范

- 3.1.3 WinSock规范与Berkeley套接口的区别
  - **6．Winsock支持的最大套接口数目**
    - 在WINSOCK.H中缺省值是64，在编译时由常量FD_SETSIZE决定。
  - **7．头文件**
    - Berkeley头文件被包含在WINSOCK.H中。一个Windows Sockets应用程序只需简单地包含WINSOCK.H就足够了。
  - **8．Winsock规范对于消息驱动机制的支持**
    - 体现在异步选择机制、异步请求函数、阻塞处理方法、错误处理、启动和终止等方面。

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 1. 初始化函数WSAStartup()
    - Winsock 应用程序要做的第一件事，就是必须首先调用WSAStartup()函数对Winsock进行初始化。初始化也称为注册。注册成功后，才能调用其他的Winsock API函数。
    - （1）WSAStartup()函数的调用格式
      - int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
    - （2）WSAStartup()函数的初始化过程
      - 图3.2说明了初始化的过程

3.2 Winsock 1.1的库函数

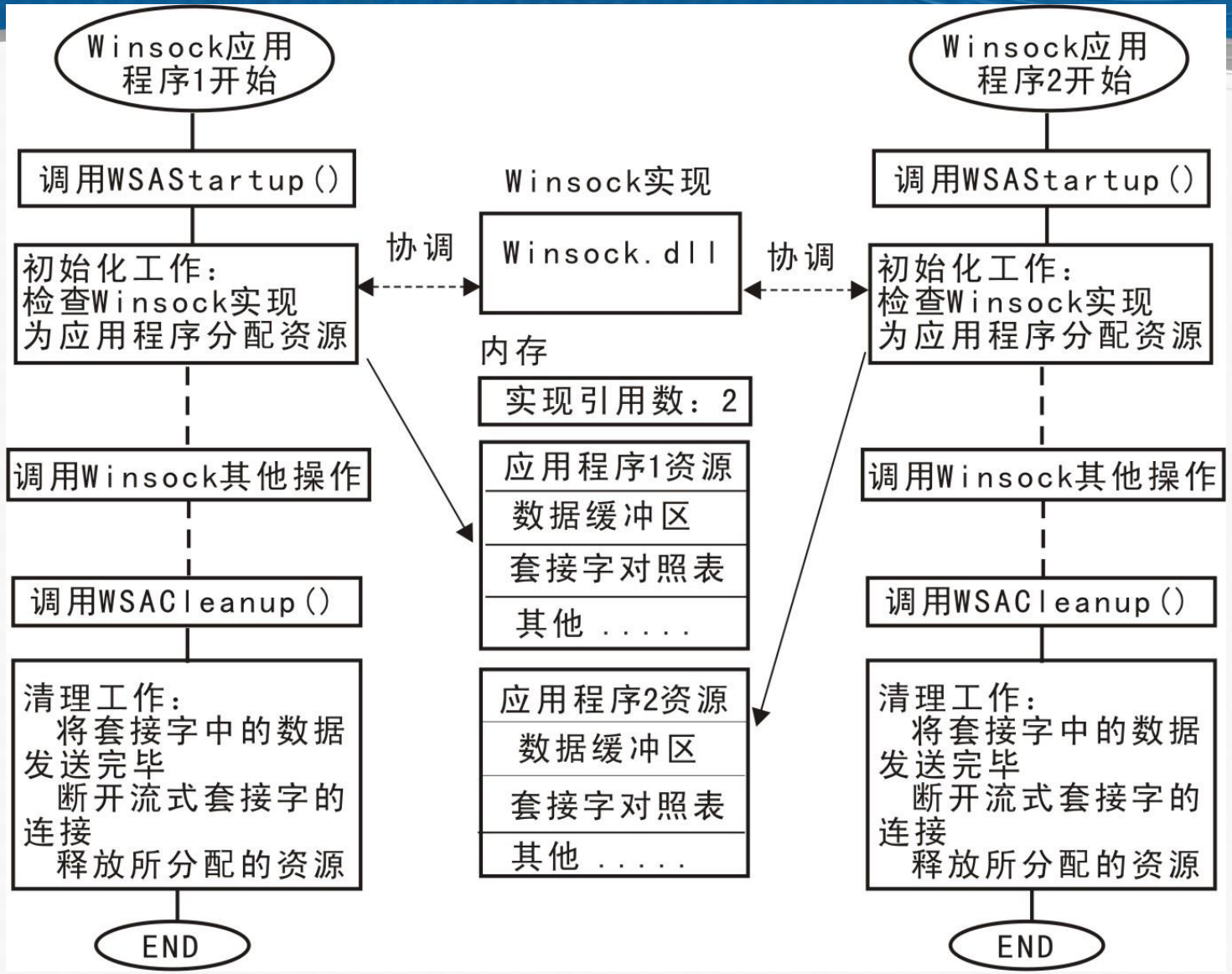

图3.2 在一台计算机中，使用同一Windock实现的多个网络应用程序

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 1．初始化函数WSAStartup()
    - **（3）WSADATA结构的定义**

```
#define WSADESCRIPTION_LEN      256
#define WSASYS_STATUS_LEN       128
typedef struct WSAData {
 WORD   wVersion;
 WORD   wHighVersion;
char    szDescription[WSADESCRIPTION_LEN+1];
char    szSystemStatus[WSASYS_STATUS_LEN+1];
unsigned short      iMaxSockets;
unsigned short      iMaxUdpDg;
char *          lpVendorInfo;
} WSADATA;
```

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 1. 初始化函数WSAStartup()
    - （4）初始化函数可能返回的错误代码
      - WSASYSNOTREADY：
        - » 网络通信依赖的网络子系统没有准备好。
      - WSAVERNOTSUPPORTED：
        - » 找不到所需的Winsock API相应的动态连接库。
      - WSAEINVAL：
        - » DLL不支持应用程序所需的Winsock版本。
      - WSAEINPROGRESS：
        - » 正在执行一个阻塞的Winsock 1.1操作。
      - WSAEPROCLIM：
        - » 已经达到Winsock支持的任务数上限。
      - WSAEFAULT：
        - » 参数lpWSAData不是合法指针。

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 1．初始化函数WSAStartup()
    - （5）初始化Winsock的示例

```
#include <winsock.h>
// 对于Winsock 2.0，应包括 Winsock2.h文件
aa() {
WORD wVersionRequested;
// 应用程序所需的Winsock版本号
WSADATA wsaData;
// 用来返回Winsock 实现的细节信息。
Int err;
// 出错代码。
```

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 1．初始化函数WSAStartup()
    - （5）初始化Winsock的示例

```
#include <winsock.h>
// 对于Winsock 2.0，应包括 Winsock2.h文件
aa() {
WORD wVersionRequested;
// 应用程序所需的Winsock版本号
WSADATA wsaData;
// 用来返回Winsock 实现的细节信息。
Int err;
// 出错代码。
```

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 1. 初始化函数WSAStartup()
    - （5）初始化Winsock的示例

```
wVersionRequested =MAKEWORD(1,1);
// 生成版本号1.1。
err = WSAStartup(wVersionRequested, &wsaData );
// 调用初始化函数。
if (err!=0 ) { return;}          / / 通知用户找不到合适的DLL文件。
// 确认返回的版本号是客户要求的1.1
if ( LOBYTE(wsaData.wVersion )!=1 ||
   HYBYTE(wsaData.wVersion )!=1) {
WSACleanup(); return;
}
/* 至此，可以确认初始化成功，Winsock.DLL可用。
}
```

3.2 Winsock 1.1的库函数

- 3.2.1 Winsock的注册与注销
  - 2. 注销函数WSACleanup()
    - 当程序使用完Winsock.DLL提供的服务后，应用程序必须调用WSACleanup()函数，来解除与Winsock.DLL库的绑定，释放Winsock实现分配给应用程序的系统资源，中止对Windows Sockets DLL的使用。
    - int WSACleanup (void);

3.2 Winsock 1.1的库函数

- 3.2.2 Winsock的错误处理函数
  - 1．WSAGetLastError()函数

    int WSAGetLastError (void);

    本函数返回本线程进行的上一次Winsock函数调用时的错误代码。

  - 2．WSASetLastError()函数

    void WSASetLastError (int iError);

    本函数允许应用程序为当前线程设置错误代码，并可由后来的WSAGetLastError()调用返回。

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 1. 创建套接口SOCKET()
    - SOCKET socket (int af, int type, int protocol);
    - 举例：

```
SOCKET sockfd=SOCKET( AF_INET,
  SOCK_STREAM, 0); /* 创建一个流式套接字。
SOCKET sockfd=SOCKET( AF_INET,
  SOCK_DGRAM, 0); /* 创建一个数据报套接字。
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 2. 将套接口绑定到指定的网络地址BIND()
    - int bind(SOCKET s, const struct sockaddr * name, int namelen);
    - 相关的三种Winsock地址结构
    - 有许多函数都需要套接字的地址信息，像UNIX 套接字一样，Winsock也定义了三种关于地址的结构，经常使用。
      - ①通用的Winsock地址结构，针对各种通信域的套接字，存储它们的地址信息。

```
struct sockaddr {
u_short sa_family;   /* 地址家族
char sa_data[14];    /* 协议地址
}
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 2. 将套接口绑定到指定的网络地址BIND()
    - ②专门针对Internet 通信域的Winsock地址结构

```
struct sockaddr_in {
short.        sin_family;  /*  指定地址家族，一定是AF_INET.
u_short       sin_port;    /*  指定将要分配给套接字的传输层
   端口号,
struct in_addr  sin_addr;    /*  指定套接字的主机的IP 地址
char          sin_zero[8];  /* 全置为0，是一个填充数。
}
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 2. 将套接口绑定到指定的网络地址BIND()
    - ③专用于存储IP地址的结构

```
Struct in_addr {
Union {
Struct {u_char s_b1,s_b2,s_b3,s_b4;} S_un_b;
Struct {u_short s_w1,s_w2;} S_un_w;
U_long  S_addr;
}
}
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 2. 将套接口绑定到指定的网络地址BIND()
    - 在使用Internet域的套接字时，这三个数据结构的一般用法是：
      - 首先，定义一个Sockaddr_in的结构实例变量，并将它清零。
      - 然后，为这个结构的各成员变量赋值，
      - 第三步，在调用BIND()绑定函数时，将指向这个结构的指针强制转换为 sockaddr*类型。

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 2．将套接口绑定到指定的网络地址BIND()
    - 举例：

```
SOCKET serSock;
 // 定义了一个SOCKET 类型的变量。
sockaddr_in my_addr;
// 定义一个Sockaddr_in型的结构实例变量。
int err;    // 出错码。
int slen=sizeof( sockaddr);  // sockaddr 结构的长度。
serSock = SOCKET(AF_INET, SOCK_DGRAM,0 );
// 创建数据报套接字。
memset(my_addr，0);        // 将Sockaddr_in的结构实例变量清零。
my_addr.sin_family = AF_INET; // 指定通信域是Internet。
my_addr.sin_port = htons(21);
// 指定端口，将端口号转换为网络字节顺序。
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 2. 将套接口绑定到指定的网络地址BIND()
    - 举例：

```
/* 指定IP地址，将IP地址转换为网络字节顺序。
my_addr.sin_addr.s_addr = htonl( INADDR-ANY);
/* 将套接字绑定到指定的网络地址，对&my_addr进行了强制类
   型转换。
if  (BIND(serSock, (LPSOCKADDR )&my_addr, slen) ==
   SOCKET_ERROR )
{
/* 调用WSAGetLastError()函数，获取最近一个操作的错误代码。
err = WSAGetLastError();
/* 以下可以报错，进行错误处理。
}
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 3．启动服务器监听客户端的连接请求LISTEN()

    int listen(SOCKET s, int backlog);

  - 4．接收连接请求 ACCEPT()

    SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen);

  - 5．请求连接CONNECT()

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

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 举例

```
struct sockaddr_in daddr;
memset((void *)&daddr,0,sizeof(daddr));
daddr.sin_family=AF_INET;
daddr.sin_port=htons(8888);
daddr.sin_addr.s_addr=inet_addr("133.197.22.4");
connect(ClientSocket,(struct sockaddr *)&daddr,size
  of(daddr));
```

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - **6. 向一个已连接的套接口发送数据SEND()**

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

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数

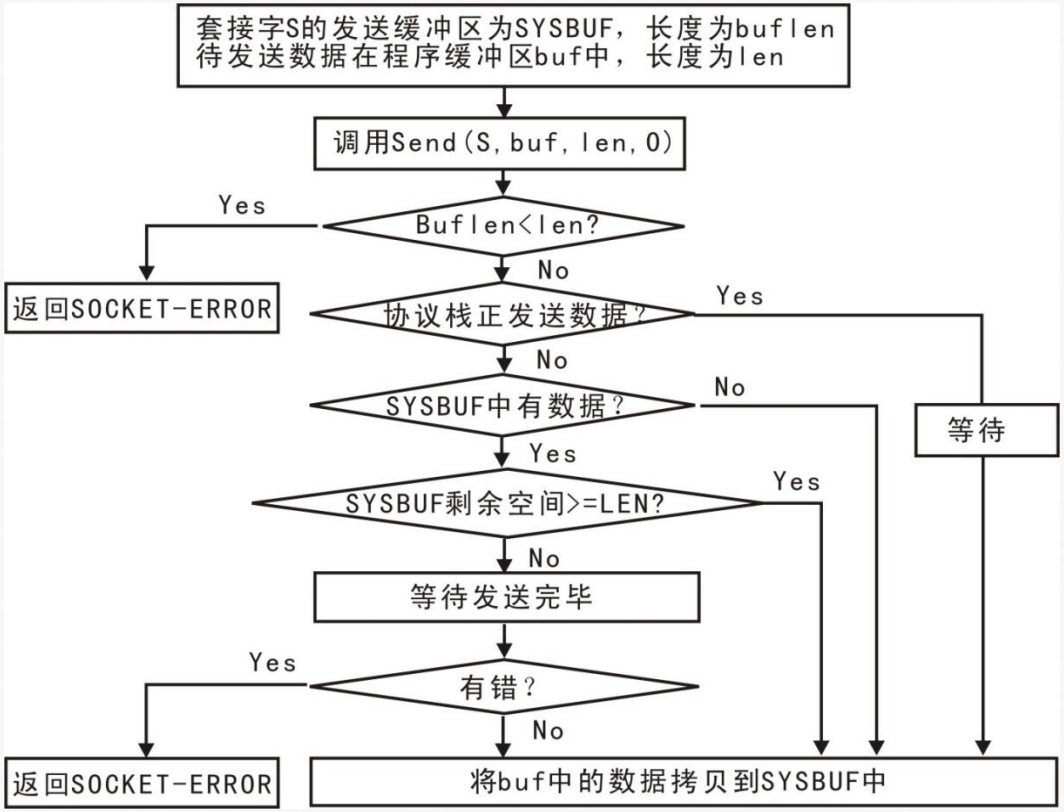

图3.3 同步套接字的Send()函数的执行流程

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 7. 从一个已连接套接口接收数据RECV()
    - int recv(SOCKET s, char * buf, int len, int flags);
    - 图3-4说明了send和recv的作用，套接字缓冲区与应用进程缓冲区的关系，以及协议栈所作的传送。

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数

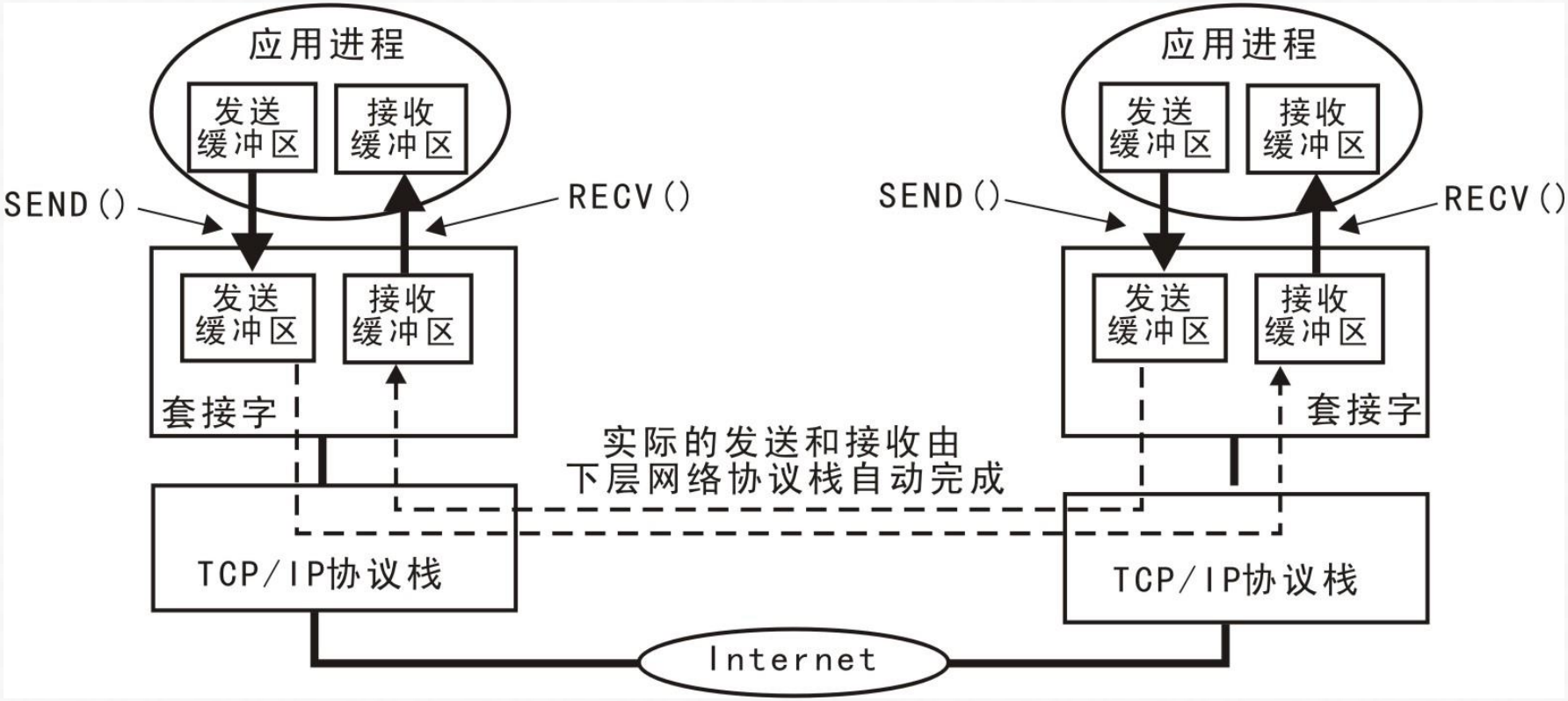

图3.4 Send()和Recv()都是对本地套接字的操作

3.2 Winsock 1.1的库函数

- 3.2.3 主要的Winsock函数
  - 8. 按照指定目的地向数据报套接字发送数据 SENDTO()
    - int sendto(SOCKET s, char * buf, int len, int flags, struct sockaddr * to, int tolen);
  - 9. 接收一个数据报并保存源地址，从数据报套接字接收数据RECVFORM()
    - int recvfrom(SOCKET s, char * buf, int len, int flags, struct sockaddr* from, int* fromlen);
  - 10. 关闭套接字CLOSESOCKET()
    - int closesocket(SOCKET s);
  - 11. 禁止在一个套接口上进行数据的接收与发送 SHUTDOWN()
    - int shutdown(SOCKET s, int how);

- 3.2.4 Winsock的辅助函数
  - 1. Winsock中的字节顺序转换函数
    - 图3.5 两种本机字节顺序。

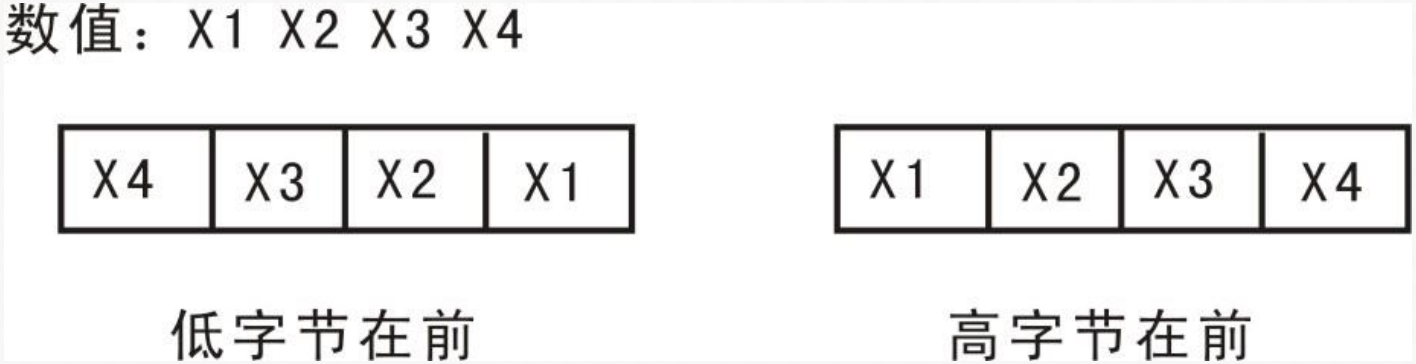

图3-5 两种本机字节顺序

3.2 Winsock 1.1的库函数

- 3.2.4 Winsock的辅助函数
  - 1. Winsock中的字节顺序转换函数
    - Winsock API特为此设置了四个函数
      - （1）**htonl()**
        - 将主机的无符号长整型数本机顺序转换为网络字节顺序(Host to Network Long)，用于IP地址。
        - u_long PASCAL FAR htonl(u_long hostlong);
        - hostlong是主机字节顺序表达的32位数。htonl()返回一个网络字节顺序的值。
      - （2）**htons()**
        - 将主机的无符号短整型数转换成网络字节顺序(Host to Network Short)，用于端口号。
        - u_short PASCAL FAR htons(u_short hostshort);
        - hostshort：主机字节顺序表达的16位数。htons()返回一个网络字节顺序的值。

3.2 Winsock 1.1的库函数

- 3.2.4 Winsock的辅助函数
  - 1. Winsock中的字节顺序转换函数
    - Winsock API特为此设置了四个函数
      - （3）ntohl()
        - 将一个无符号长整型数从网络字节顺序转换为主机字节顺序。(Network to Host Long)，用于IP地址。
        - u_long PASCAL FAR ntohl(u_long netlong);
        - netlong是一个以网络字节顺序表达的32位数，ntohl()返回一个以主机字节顺序表达的数。
      - （4）ntohs()
        - 将一个无符号短整型数从网络字节顺序转换为主机字节顺序。(Network to Host Sort)，用于端口号
        - u_short PASCAL FAR ntohs(u_short netshort);
        - netshort是一个以网络字节顺序表达的16位数。ntohs()返回一个以主机字节顺序表达的数。

3.2 Winsock 1.1的库函数

- 3.2.4 Winsock的辅助函数
  - **2．获取与套接口相连的端地址GETPEERNAME()**
    - int getpeername(SOCKET s, struct sockaddr * name, int * namelen);
  - **3．获取一个套接口的本地名字GETSOCKNAME()**
    - int getsockname(SOCKET s, struct sockaddr * name, int * namelen);
  - **4．将一个点分十进制形式的IP地址转换成一个长整型数INET_ADDR()**
    - unsigned long inet_addr (const char * cp);
  - **5．将网络地址转换成点分十进制的字符串格式INET_NTOA()**
    - char * inet_ntoa(struct in_addr in);

3.2 Winsock 1.1的库函数

- 3.2.5 Winsock的信息查询函数
  - Winsock API提供了一组信息查询函数，让我们能方便地获取套接口所需要的网络地址信息以及其它信息，
    - （1）Gethostname()
      - 用来返回本地计算机的标准主机名。
      - int gethostname(char* name, int namelen);
    - （2）Gethostbyname()
      - 返回对应于给定主机名的主机信息。
      - struct hostent* gethostbyname(const char* name);

3.2 Winsock 1.1的库函数

- 3.2.5 Winsock的信息查询函数
  - Winsock API提供了一组信息查询函数，让我们能方便地获取套接口所需要的网络地址信息以及其它信息，
    - （3）Gethostbyaddr()
      - 根据一个IP地址取回相应的主机信息。
      - struct hostent* gethostbyaddr(const char* addr, int len, int type);
    - （4）Getservbyname()
      - 返回对应于给定服务名和协议名的相关服务信息。
      - struct servent* getservbyname(const char* name, const char* proto);
    - （5）Getservbyport()
      - 返回对应于给定端口号和协议名的相关服务信息。
      - struct servent * getservbyport(int port, const char *proto);

3.2 Winsock 1.1的库函数

- 3.2.5 Winsock的信息查询函数
  - Winsock API提供了一组信息查询函数，让我们能方便地获取套接口所需要的网络地址信息以及其它信息，
    - （6）Getprotobyname()
      - 返回对应于给定协议名的相关协议信息。
      - struct protoent * getprotobyname(const char * name);
    - （7）Getprotobynumber ()
      - 返回对应于给定协议号的相关协议信息。
      - struct protoent * getprotobynumber(int number);

3.2 Winsock 1.1的库函数

- 3.2.5 Winsock的信息查询函数
  - 除了Gethostname()函数以外，其它六个函数有以下共同的特点：
    - ①函数名都采用GetXbyY的形式。
    - ②如果函数成功地执行，就返回一个指向某种结构的指针，该结构包含所需要的信息。
    - ③如果函数执行发生错误，就返回一个空指针。应用程序可以立即调用WSAGetLastError()来得到一个特定的错误代码。

3.2 Winsock 1.1的库函数

- 3.2.5 Winsock的信息查询函数
  - 除了Gethostname()函数以外，其它六个函数有以下共同的特点：
    - ④函数执行时，可能在本地计算机上查询，也可能通过网络向域名服务器发送请求，来获得所需要的信息，这取决于用户网络的配置方式。
    - ⑤为了能让程序在等待响应时能作其他的事情，Winsock API扩充了一组作用相同的异步查询函数，不会引起进程的阻塞。并且可以使用Windows的消息驱动机制。也是六个函数，与GetXbyY各函数对应，在每个函数名前面加上了WSAAsync前缀，名字采用WSAAsyncGetXByY()的形式。它们的工作机制在后面详述

3.2 Winsock 1.1的库函数

- 3.2.6 WSAAsyncGetXByY类型的扩展函数
  - WSAAsyncGetXByY类型的扩展函数是GetXByY函数的异步版本，这些函数可以很好地利用Windows的消息驱动机制。
    - 1. WSAAsyncGetHostByName()函数
      - HANDLE WSAAsyncGetHostByName (HWND hWnd, unsigned int wMsg,
      - const char * name, char * buf, int buflen);
    - 2. WSAAsyncGetHostByAddr()函数
      - HANDLE WSAAsyncGetHostByAddr (HWND hWnd, unsigned int wMsg,
      - const char * addr, int len, int type, char * buf, int buflen);

3.2 Winsock 1.1的库函数

- 3.2.6 WSAAsyncGetXByY类型的扩展函数
  - WSAAsyncGetXByY类型的扩展函数是GetXByY函数的异步版本，这些函数可以很好地利用Windows的消息驱动机制。
    - 3．WSAAsyncGetServByName()函数
      - HANDLE WSAAsyncGetServByName (HWND hWnd, unsigned int wMsg,
      - const char * name, const char * proto, char * buf, int buflen);
    - 4．WSAAsyncGetServByPort()
      - HANDLE WSAAsyncGetServByPort (HWND hWnd, unsigned int wMsg,
      - int port, const char * proto, char * buf, int buflen);

3.2 Winsock 1.1的库函数

- 3.2.6 WSAAsyncGetXByY类型的扩展函数
  - WSAAsyncGetXByY类型的扩展函数是GetXByY函数的异步版本，这些函数可以很好地利用Windows的消息驱动机制。
    - 5．WSAAsyncGetProtoByName()函数
      - HANDLE WSAAsyncGetProtoByName (HWND hWnd, unsigned int wMsg,
      - const char * name, char * buf, int buflen);
    - 6．WSAAsyncGetProtoByNumber()函数
      - HANDLE WSAAsyncGetProtoByNumber (HWND hWnd, unsigned int wMsg,
      - int number, char * buf, int buflen);

3.3 网络应用程序的运行环境

- 1．开发Windows Sockets网络应用程序的软、硬件环境
  - 采用支持Windows Sockets API的Windows98SE以上的操作系统。
  - 采用可视化和面向对象技术的编程语言，如Microsoft Visual C++ 6.0
  - 采用TCP/IP网络通信协议。

3.3 网络应用程序的运行环境

- 1．开发Windows Sockets网络应用程序的软、硬件环境
  - 网络中的所采用的计算机应满足Windows运行的配置要求。
  - 网络中各节点上的计算机需安装网卡，并安装网卡的驱动程序。可以采用以太网交换机将若干台计算机组建成局域网。
  - 在配置网络时，首先实现对等网，使各计算机节点能在“网上邻居”中找到自己和其它各计算机，并能实现文件资源相互共享。
  - 其次，网络配置中，应添加TCP/IP协议，设定相应的IP地址。

3.3 网络应用程序的运行环境

- 2．调用Windows Sockets接口的基本步骤
  - 采用不同套接字的应用程序的调用套接字函数时，应遵循相应的步骤。
- 3．使用Visual C++ 6.0进行Windows Sockets程序开发的其它技术要点
  - （1）首先做好初始化处理。
  - （2）通信双方的程序应采用统一的界面形式。
  - （3）尽量采用多线程（Multithreaded）编程技术。
  - （4）应充分利用Windows Sockets的基于消息的网络事件异步选择机制。