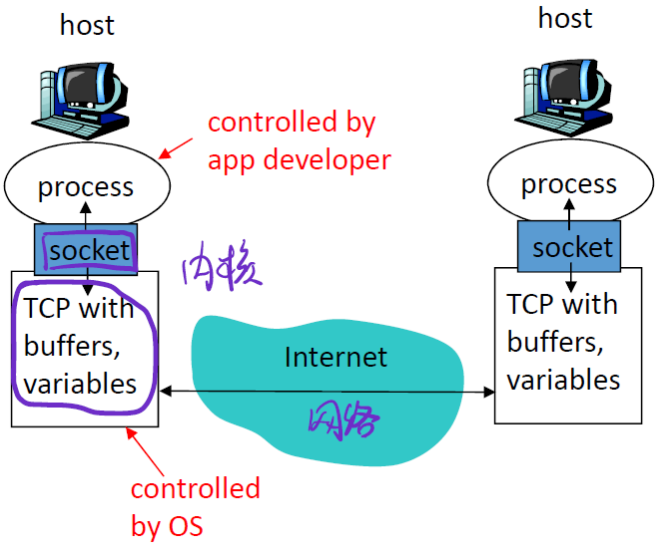
Socket-Review

1 Socket概述

socket套接字： 支持TCP/IP的操作系统为网络程序开发提供的典型网络编程界面

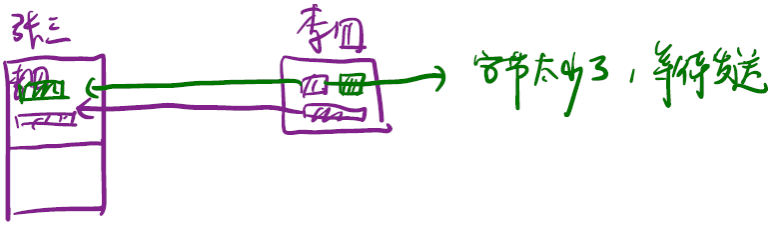
具体的通信流程：



- 数据报套接字：使用UDP协议，支持主机之间面向非连接、不可靠的数据传输
- 流式套接字：使用TCP协议，支持主机之间面向连接的、顺序的、可靠的全双工字节流传输

TCP传输和UDP传输的区别：

TCP传输：



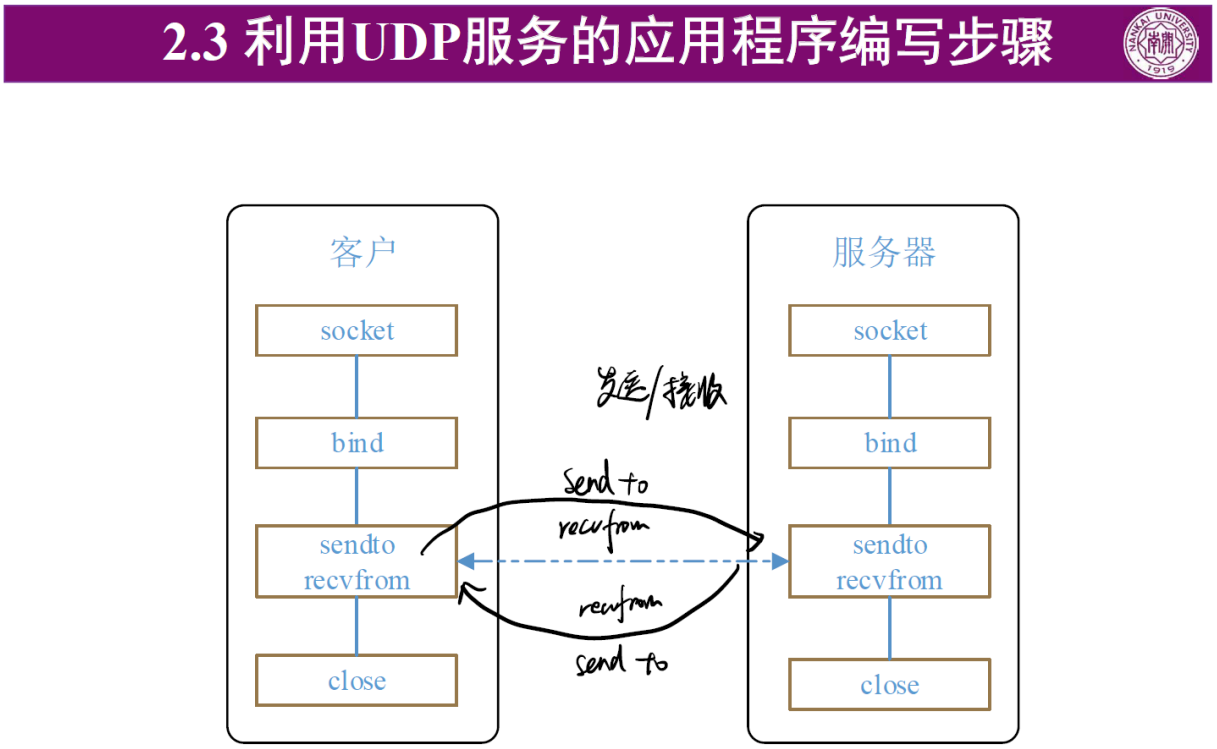
就是将传输的结果都分开放，一个一个的分开，然后会进行等待发送。也就是说，不同人的数据包会放在不同的地方，会等待合并数据包。

UDP传输：

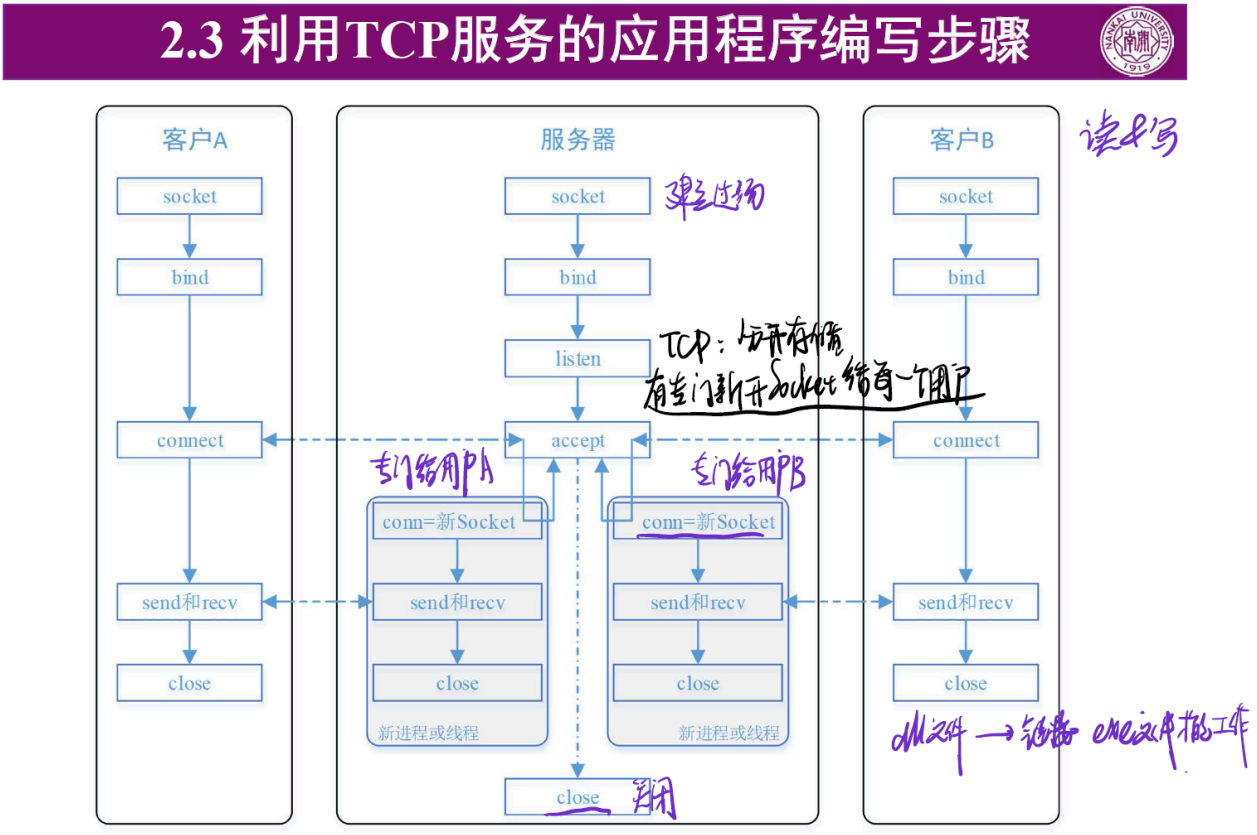
不同的数据包都放在一起，不会等待数据包，到了就发送。不同人的数据包放在一块，不会进行等待。

2 编程框架

UDP编程：



TCP编程：



3 Socket函数

3.1 WSAStartup

初始化 Socket DLL，协商使用的 Socket 版本

```
int WSAAPI WSAStartup (      //成功返回 0，否则为错误代码
  WORD wVersionRequested,    //调用者希望使用的最高版本
  LPWSADATA lpWSAData        //可用 Socket 的详细信息
);
```

- `wVersionRequested`：请求的 Winsock 版本号，通常使用 `MAKEWORD(2, 2)` 表示请求 Winsock 2.2 版本。
- `lpWSAData`：指向 `WSADATA` 结构的指针，用于接收关于 Winsock 实现的信息。

如果调用成功，不再使用时需要调用 WSACleanup 释放 Socket DLL 资源

示例：

```
WSAData wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData); //MAKEWORD（主版本号，副版本号）
```

3.2 WSAData

```
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 FAR       *lpVendorInfo;
} WSADATA;
```

- `wVersion`：请求的 Winsock 版本。`wVersion` 的低字节是主版本号，高字节是次版本号。
- `wHighVersion`：Winsock 实现的最高版本。

3.3 SOCKADDR

```
struct sockaddr {
  unsigned short sa_family;  // 地址家族，例如 AF_INET
  char sa_data[14];          // 地址数据，实际类型和长度取决于地址家族
} SOCKADDR, *PSOCKADDR, *LPSOCKADDR;
```

3.4 SOCKADDR_IN

`SOCKADDR_IN` 是在 Windows 套接字编程中用于表示 IPv4 地址的结构体。它是 `sockaddr_in` 结构的别名，具体类型为 `struct sockaddr_in`。`sockaddr_in` 结构是对通用的 `sockaddr` 结构的扩展，专门用于存储 IPv4 地址信息。

```
struct sockaddr_in {
  short sin_family;          // 地址家族，AF_INET 表示 IPv4 地址
  unsigned short sin_port;   // 端口号，使用网络字节序（big-endian）
  struct in_addr sin_addr;   // IPv4 地址结构
  char sin_zero[8];          // 用于填充结构体，使其和 sockaddr 兼容
} SOCKADDR_IN, *PSOCKADDR_IN, *LPSOCKADDR_IN;
```

示例：

```
ServerAddr.sin_family = AF_INET;
ServerAddr.sin_port = htons(Server_Port);
ServerAddr.sin_addr.S_un.S_addr = inet_addr(ServerIP.c_str());
```

3.5 in_addr

```
struct in_addr {
  union {
    struct {
      u_char s_b1;
      u_char s_b2;
      u_char s_b3;
      u_char s_b4;
    } S_un_b;

    struct {
      u_short s_w1;
      u_short s_w2;
    } S_un_w;

    u_long S_addr;
  } S_un;
};
```

3.6 大端序&小端序

Big-Endian和Little-Endian

- ***Q***：计算机中，整数是如何存储的？
- ***Little-Endian***：低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。
- ***Big-Endian***：高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

- 例如：0x12 34 56 78在内存中的存放形式为：
  - 内存　　低地址 ----------> 高地址
  - Big-Endian：　0x12 | 0x34 | 0x56 | 0x78
  - Little-Endian：　0x78 | 0x56 | 0x34 | 0x12

在网络中，一般使用大端序。

3.7 主机序与网络序的转换

1. `htons`（host to network short）和 `ntohs`（network to host short）：
   - `htons` 用于将本地字节序的 16 位短整数转换为网络字节序。
   - `ntohs` 用于将网络字节序的 16 位短整数转换为本地字节序。
2. `htonl`（host to network long）和 `ntohl`（network to host long）：
   - `htonl` 用于将本地字节序的 32 位长整数转换为网络字节序。
   - `ntohl` 用于将网络字节序的 32 位长整数转换为本地字节序。

3.8 WSACleanup

结束使用 Socket，释放 Socket DLL 资源

```
int WSAAPI WSACleanup();        //成功返回 0
```

调用失败后可利用WSAGetLastError获取详细错误信息

3.9 socket

创建一个 Socket，并绑定到一个特定的传输层服务

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

- `af`: 地址族（Address Family），指定套接字使用的地址族，常见的有 `AF_INET`（IPv4 地址族）和 `AF_INET6`（IPv6 地址族）。
- `type`: 套接字类型，指定套接字的通信语义，常见的有 `SOCK_STREAM`（流式套接字，用于 TCP）和 `SOCK_DGRAM`（数据报套接字，用于 UDP）。
- `protocol`: 指定套接字使用的协议，常见的有 `IPPROTO_TCP`（TCP 协议）和 `IPPROTO_UDP`（UDP 协议）。

返回：

- 正确：Socket 描述符
- 错误：INVALID_SOCKET。可通过 WSAGetLastError 获取错误详情

示例：

```
ClientSocket = socket(AF_INET, SOCK_DGRAM, 0); //数据报套接字
ClientSocket = socket(AF_INET, SOCK_STREAM, 0); //流式套接字
```

3.10 bind

将一个本地地址绑定到指定的 Socket

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

- `s`: 要绑定的套接字描述符。
- `name`: 指向 `sockaddr` 结构的指针，包含要绑定到套接字的本地地址信息。通常需要将 `sockaddr_in` 结构强制类型转换为 `sockaddr` 结构。
- `namelen`: 本地地址结构的长度，通常是 `sizeof(sockaddr_in)`。

返回：

- 如果函数成功，返回0。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例：

```
// 设置服务器地址信息
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(12345);  // 设置端口号
serverAddr.sin_addr.s_addr = INADDR_ANY;  // 允许任何地址连接

// 绑定套接字到本地地址
bind(serverSocket, (LPSOCKADDR)&serverAddr, sizeof(serverAddr));
```

3.11 listen

TCP

使 socket 进入监听状态，监听远程连接是否到来

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

- `s`: 要监听的套接字描述符。
- `backlog`: 指定等待连接队列的最大长度。等待连接队列是指已经连接到服务器套接字的客户端连接请求的队列。

返回值:

- 如果函数成功，返回0。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例:

```
// 将套接字设置为监听状态
listen(serverSocket, SOMAXCONN);
```

3.12 connect

TCP

向一个特定的 socket 发出建连请求

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

- `s`: 要连接的套接字描述符。
- `name`: 指向 `sockaddr` 结构的指针，包含要连接的远程主机的地址信息。通常需要将 `sockaddr_in` 结构强制类型转换为 `sockaddr` 结构。

- `namelen`：远程主机地址结构的长度，通常是 `sizeof(sockaddr_in)`。

返回值：

- 如果函数成功，返回0。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例：

```
// 连接到服务器
connect(clientSocket, (SOCKADDR*)&serverAddr, sizeof(serverAddr));
```

3.13 accept

TCP

接受一个特定 socket 请求等待队列中的连接请求

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

- `s`：要接受连接的套接字描述符，即服务器套接字。
- `addr`：指向 `sockaddr` 结构的指针，用于存储客户端的地址信息。可以为 `nullptr`，表示不获取客户端地址信息。
- `addrlen`：指向 `int` 的指针，用于存储 `addr` 结构的长度。在调用 `accept` 之前，`*addrlen` 应该被设置为 `sizeof(sockaddr_in)`。

返回值：

- 如果函数成功，返回一个新的套接字描述符，用于与客户端通信。
- 如果函数失败，返回 `INVALID_SOCKET`，可以通过调用 `WSAGetLastError` 获取错误码。

示例：

```
// 接受连接请求
sockaddr_in clientAddr;
int clientAddrLen = sizeof(clientAddr);
SOCKET clientSocket = accept(serverSocket, (sockaddr*)&clientAddr, &clientAddrLen);
```

3.14 send

TCP

向远程 socket 发送数据

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

- `s`: 要发送数据的已连接套接字描述符。
- `buf`: 指向包含要发送数据的缓冲区的指针。
- `len`: 要发送的数据的长度。
- `flags`: 发送标志，通常可以设置为 0。

返回值:

- 如果函数成功，返回发送的字节数。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例:

```
//使用 clientsocket 向已连接的套接字发送 buf 内的内容
send(clientSocket, buf, sizeof(buf), 0);
```

3.15 recv

TCP

接收远程 socket 发送的数据

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

- `s`: 要接收数据的已连接套接字描述符。
- `buf`: 指向用于存储接收数据的缓冲区的指针。
- `len`: 缓冲区的长度。
- `flags`: 接收标志，通常可以设置为 0。

返回值:

- 如果函数成功，返回接收到的字节数。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例:

```
//使用 clientsocket 接收来自已连接的套接字的数据并存储到 buf 中
recv(clientSocket, buffer, sizeof(buffer), 0)
```

3.16 sendto

UDP

向特定的目的地发送数据

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

- `s`: 要发送数据的套接字描述符。
- `buf`: 指向包含要发送数据的缓冲区的指针。
- `len`: 要发送的数据的长度。
- `flags`: 发送标志，通常可以设置为 0。
- `to`: 指向 `sockaddr` 结构的指针，包含目标地址信息。
- `tolen`: 目标地址结构的长度，通常是 `sizeof(sockaddr_in)`。

返回值:

- 如果函数成功，返回发送的字节数。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例:

```
//使用 client 端向 router 端发送数据 msg
sendto(ClientSocket, (char *)&msg, sizeof(msg), 0, (SOCKADDR *)&RouterAddr, RouterAddrLen);
```

3.17 recvfrom

UDP

从特定的目的地接收数据

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

- `s`: 要接收数据的套接字描述符。

- `buf`：指向用于存储接收数据的缓冲区的指针。
- `len`：缓冲区的长度。
- `flags`：接收标志，通常可以设置为 0。
- `from`：指向 `sockaddr` 结构的指针，用于存储发送端的地址信息。
- `fromlen`：指向 `int` 的指针，用于存储 `from` 结构的长度。在调用 `recvfrom` 之前，`*fromlen` 应该被设置为 `sizeof(sockaddr_in)`。

返回值：

- 如果函数成功，返回接收到的字节数。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

示例：

```
//使用 clientsocket 接收来自 routersocket 的信息，并将其存储到 msg 中
recvfrom(ClientSocket, (char *)&msg, sizeof(msg), 0, (SOCKADDR *)&RouterAddr,
&RouterAddrLen);
```

3.18 closesocket

关闭一个存在的 socket

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

- `s`：要关闭的套接字描述符。

返回值：

- 如果函数成功，返回0。
- 如果函数失败，返回 `SOCKET_ERROR`，可以通过调用 `WSAGetLastError` 获取错误码。

3.19 CreateThread

创建线程

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  SIZE_T                  dwStackSize,
  LPTHREAD_START_ROUTINE  lpStartAddress,
  LPVOID                  lpParameter,
  DWORD                   dwCreationFlags,
  LPDWORD                 lpThreadId
);
```

- `lpThreadAttributes`：返回句柄能不能被继承，一般为 `NULL`。
- `dwStackSize`：指定线程的初始栈大小，一般为 `0` 表示使用默认值。

- `lpStartAddress`：线程函数的起始地址，即线程创建后将执行的函数。
- `lpParameter`：传递给线程函数的参数。
- `dwCreationFlags`：指定线程的创建标志，一般为 `0`。
- `lpThreadId`：一个指向接收线程标识符的变量的指针，可以为 `NULL`。

返回值：

- 如果函数成功，返回新线程的句柄（`HANDLE`）。
- 如果函数失败，返回 `NULL`。可以通过调用 `GetLastError` 获取错误码。

示例：

```
HANDLE Thread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ThreadFunction, (LPVOID)num, 0, NULL);
```

4 UDP

4.1 client

```
void main()
{
    WSAStartup(wVersionRequested, &wsaData);
    SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, 0);
    SOCKADDR_IN addrClient;
    bind(sockClient, (SOCKADDR*)&addrClient, sizeof(SOCKADDR));
    SOCKADDR_IN addrSrv; //远程 IP 地址
    //使用 sockClient 向 addrSrv 发送 sendBuf 中的数据
    sendto(sockClient, sendBuf, sendlen, 0, (SOCKADDR *)&addrSrv, len);
    //使用 sockClient 接收来自 addrSrv 发送的数据，并存到 recvBuf 中
    recvfrom(sockClient, recvBuf, 50, 0, (SOCKADDR *)&addrSrv, &len);
    closesocket(sockClient);
    WSACleanup();
}
```

4.2 server

```
void main()
{
    WSAStartup(wVersionRequested, &wsaData);
    SOCKET sockSrv = socket(AF_INET, SOCK_DGRAM, 0);
    SOCKADDR_IN addrSrv;
    bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
    SOCKADDR_IN addrClient; //远程 IP 地址
    loop {
        //使用 sockSrv 接收来自 addrClient 发送的数据，并存到 sockSrv 中
        recvfrom(sockSrv, recvBuf, 50, 0,(SOCKADDR *)&addrClient,&len);
        //使用 sockSrv 向 addrClient 发送 sendBuf 中的数据
        sendto(sockSrv, sendBuf, sendlen, 0,(SOCKADDR *) &addrClient,len);
    }
    closesocket(sockClient);
    WSACleanup();
}
```

5 TCP

5.1 client

```
void main()
{
    WSAStartup(wVersionRequested, &wsaData);
    SOCKET sockClient = socket(AF_INET, SOCK_STREAM, 0);
    SOCKADDR_IN addrSrv; //要连接的远程主机地址信息
    //使用 sockClient 向 addrSrv 申请建立连接
    connect(sockClient, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
    //接收来自 sockClient 已连接的套接字发送的数据，并存到 recvBuf 中
    recv(sockClient, recvBuf, 50, 0);
    //向 sockClient 已连接的套接字发送 “hello” 数据
    send(sockClient, "hello", sendlen, 0);
    closesocket(sockClient);
    WSACleanup();
}
```

5.2 server

5.2.1 单线程

```
void main()
{
    WSAStartup(wVersionRequested, &wsaData);
    SOCKET sockSrv = socket(AF_INET, SOCK_STREAM, 0);
    SOCKADDR_IN addrSrv, addrClient;
    bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
    //将 sockSrv 设置成监听状态，最大连接数为 5
    listen(sockSrv, 5);
    while (1)
    {
        //创建一个新的套接字 sockConn 用于描述 sockSrv 接受的连接的套接字，地址信息存储在
addrClient 中
        SOCKET sockConn = accept(sockSrv, (SOCKADDR*)&addrClient, &len);
        //使用 sockConn 向已连接的客户端套接字发送 sendBuf 中的数据
        send(sockConn, sendBuf, strlen, 0);
        //接收来自 sockConn 已连接的客户端套接字发送的数据，并存储在 recvBuf 中
        recv(sockConn, recvBuf, 50, 0);
        closesocket(sockConn);
    }
    closesocket(sockSrv);
    WSACleanup();
}
```

5.2.2 多线程

```
void main()
{
    WSAStartup(wVersionRequested, &wsaData);
    SOCKET sockSrv = socket(AF_INET, SOCK_STREAM, 0);
    bind(sockSrv, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));
    listen(sockSrv, 5);
    while (1) {
```

```
        SOCKET sockConn = accept(sockSrv, (SOCKADDR*)&addrClient, &len);
        hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(sockConn), 0,
&dwThreadId);
        CloseHandle(hThread);
    }
    closesocket(sockSrv);
    WSACleanup();
}
DWORD WINAPI handlerRequest(LPVOID lparam)
{
    SOCKET ClientSocket = (SOCKET)(LPVOID)lparam;
    send(ClientSocket, sendBuf, strlen, 0);
    recv(ClientSocket, recvBuf, 50, 0);
    closesocket(ClientSocket);
    return 0;
}
```