湘潭大学信息工程学院
The College of Information Engineering of Xiangtan University

TCPIP网络编程基础教程

Dr. Wang

教学大纲（四个部分）：

1.基于套接字的TCP/IP网络通信原理与模型

套接字的基本概念、基于套接字的网络通信实现流程、BSD UNIX套接字API函数、 Windows 套接字API函数、 TCP/IP网络通信模型及其C语言实现方法

2.循环服务器软件的实现原理与方法

循环服务器软件的实现原理及其C语言实现方法

3.服务器与客户进程中的并发机制与实现方法

服务器与客户进程中的并发机制、多进程并发机制的实现原理与方法、多线程并发TCP服务器软件的实现原理与方法、单线程并发机制的实现原理与方法、基于POOL和EPOLL的并发机制与实现方法

4.客户/服务器系统中的死锁问题与解决方法

死锁的定义、产生死锁的原因、处理死锁的基本方法

第一章：基于套接字的TCP/IP网络通信原理与模型

1.1 TCP/IP协议概述

1.1.1 TCP/IP参考模型

【王老师提问】**Question 1**：TCP/IP 是什么意思，其中英文全称是什么？

◆ TCP/IP（Transmission Control Protocol/Internet Protocol），即传输控制协议/因特网协议，是一个由多种协议组成的协议族（Protocol Family），定义了计算机通过网络互相通信及协议族各层次之间通信的规范。

第一章：基于套接字的TCP/IP网络通信原理与模型

◆ TCP/IP参考模型是一个抽象的分层模型，在该模型中，属于TCP/IP协议族的所有网络协议都被归类到了以下四个抽象的“层”之中：主机-网络层（Host to Network Layer）、互联网络层（Internet Layer）、传输层（Transport Layer） 、应用层（Application Layer）

◆传输层主要负责在互联网中源主机与目的主机的对等进程实体之间提供可靠的端到端的数据传输。在TCP/IP参考模型的传输层中定义了以下这两种协议： TCP协议（Transmission Control Protocol，传输控制协议）、 UDP协议（User Datagram Protocol，用户数据报协议）

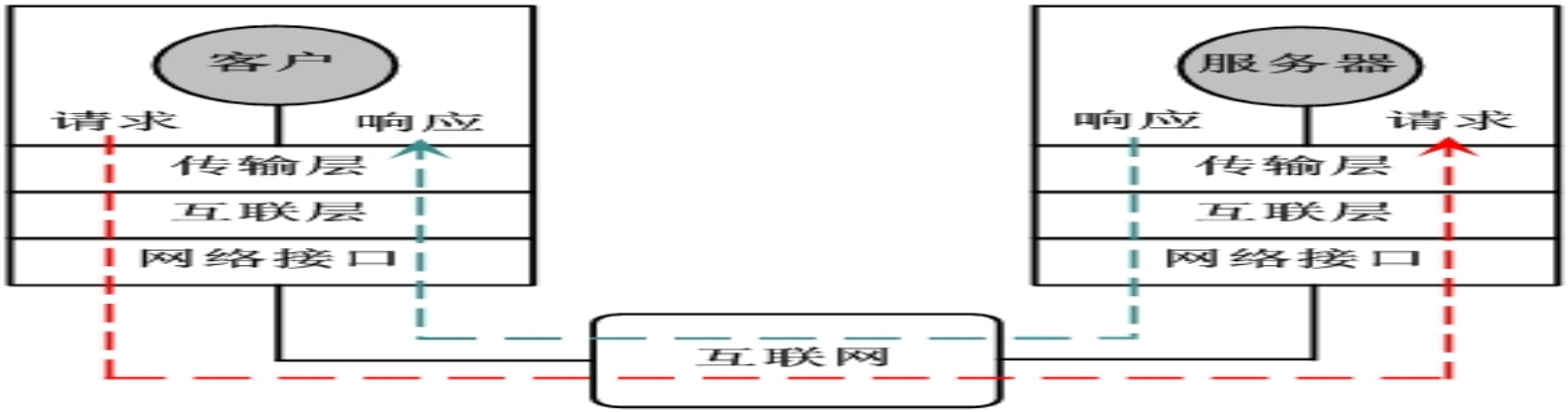
第一章：基于套接字的TCP/IP网络通信原理与模型

1.1.2 TCP/IP网络通信中的客户/服务器模型

◆如图1.1所示，在TCP/IP协议体系中，进程之间的相互作用采用客户/服务器（Client/Server，简称为C/S）模型

【王老师提问】Question 2：在C/S模型中，客户与服务器分别表示的是什么？

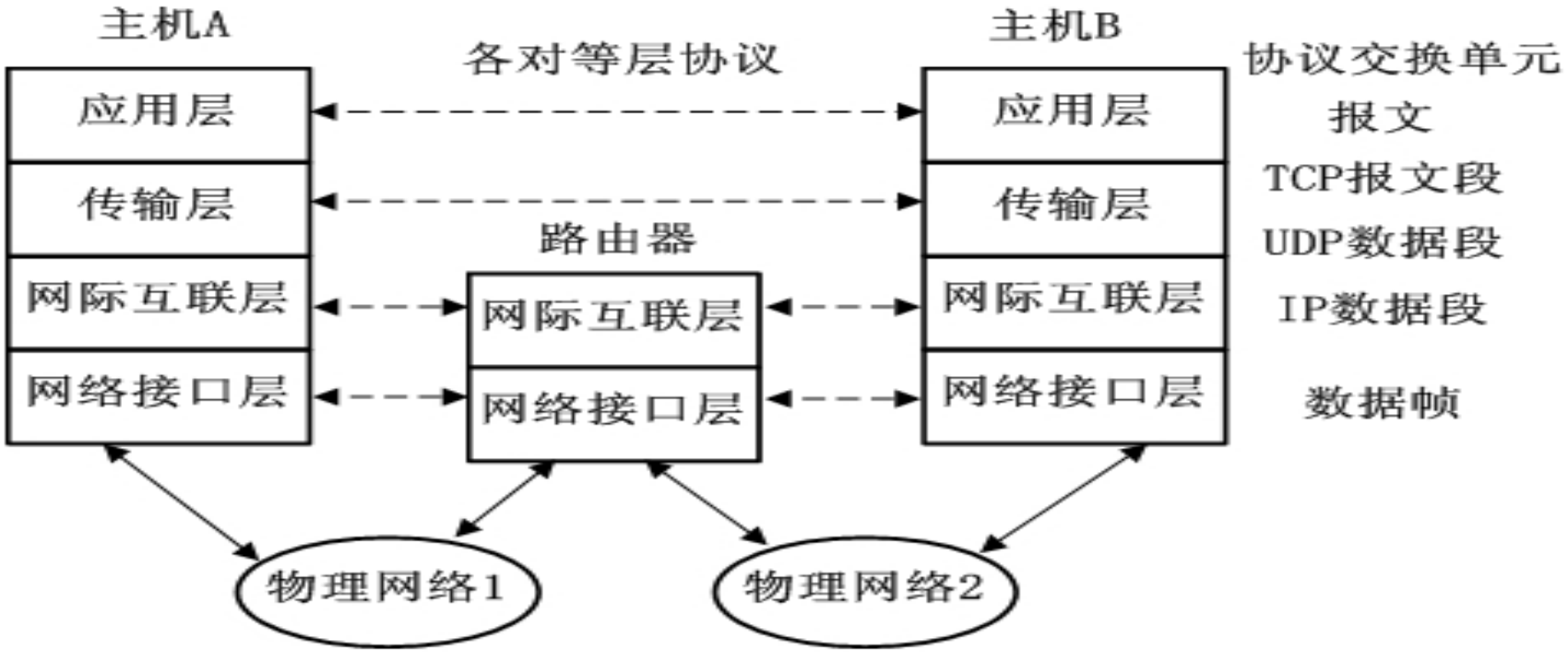
◆客户与服务器分别表示相互通信的两个应用程序进程。



第一章：基于套接字的TCP/IP网络通信原理与模型

1.1.3 TCP/IP参考模型的通信原理

◆ TCP/IP参考模型的通信原理如图1.2所示，其中，第一至二层为串联的，而第三至四层则是端到端（End to End）的。



第一章：基于套接字的TCP/IP网络通信原理与模型

◆由图1.2可知，网际互联层与网络接口层实现了计算机网络中处于不同位置的主机之间的数据通信，但是数据通信不是计算机网络的最终目的，计算机网络最本质的活动是实现分布在不同地理位置的主机之间的进程间通信（InterProcess Communication，IPC），以实现各种网络服务功能。

◆设置传输层的主要目的就是要实现上述这种分布式进程之间的通信功能。

第一章：基于套接字的TCP/IP网络通信原理与模型

网络环境下的分布式进程间通信要解决的是：不同主机进程间的相互通信问题（显然，同机进程间通信只是其中的一个特例）。为此，传输层需要解决在网络环境下分布式进程间通信所面临的以下两个方面的问题：

◆进程的命名与寻址：如何标识通信的进程？

◆多重协议的识别：操作系统支持的网络协议众多，不同协议的工作方式与地址格式均不相同。

【王老师提问】Question 3：在网络环境下，你觉得应该怎样才能标识一个进程？

第一章：基于套接字的TCP/IP网络通信原理与模型

◆ TCP/IP参考模型提出了协议端口（Protocol Port，简称端口）的概念，用于标识通信的进程。其中，**端口**是一种抽象的软件结构（包括一些数据结构和I/O缓冲区）。与文件描述符类似，每个端口均拥有一个唯一的被称为**端口号**（Port Number）的16位无符号整数型标识符，范围是0~65535，用于区别不同的端口。

【王老师提问】Question 4：请问，你觉得端口号应该如何分配给进程？

端口号一般有以下两种基本的**分配方式**：第一种为全局分配，这是一种集中分配方式，由一个公认权威的中央机构根据用户

的需要进行统一分配，并将结果公布于众，通过该方式分配的端口号也称为**熟知端口号**；

第二种为本地分配，又称动态分配，是当进程需要访问传输层服务时，向本地操作系统提出申请，再由操作系统分配本地唯一的端口号。

由于同一台机器上的不同进程分配到的端口号不同，因此，同一台机器上的不同进程就可以用端口号来唯一标识。

但在网络环境中，显然，若要标识一个完整的进程，除了端口号外，还需用到本地主机的IP地址（本地地址）来唯一标识进程所在的主机（因为不同机器上的进程可有相同端口号）。

第一章：基于套接字的TCP/IP网络通信原理与模型

在TCP/IP网络环境下，一个完整的网间通信需要由两个进程完成，并且这两个进程之间只能使用相同的传输层协议才能进行通信，也就是说，不可能通信的一端使用TCP协议，而另一端使用UDP协议。因此，一个完整的网间通信需要使用一个五元组<协议，本地地址，本地端口号，远程地址，远程端口号>才能唯一标识。

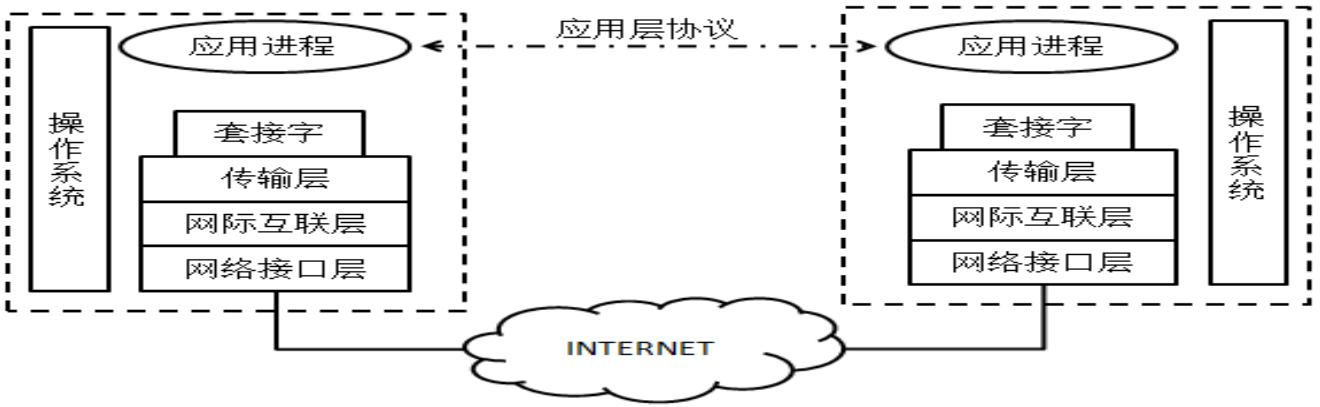
其中，二元组<本地地址，本地端口号>称为网间进程通信中的**本地端点地址**（Endpoint Address），二元组<远程地址，远程端口号>称为网间进程通信中的**远程端点地址**，

第一章：基于套接字的TCP/IP网络通信原理与模型

1.2 基于套接字的网络通信原理

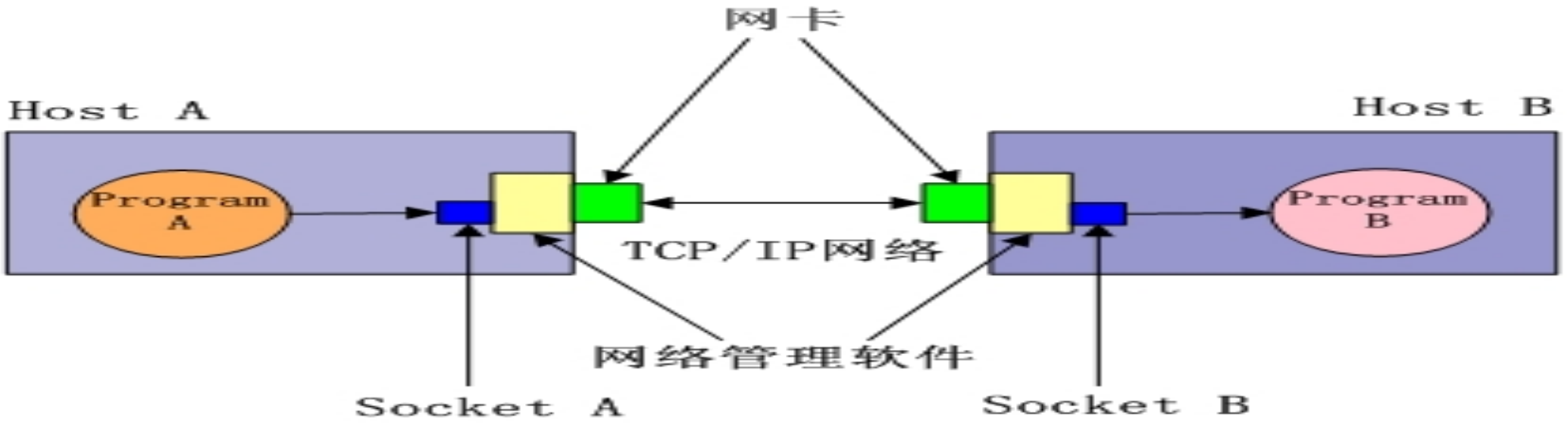
1.2.1 套接字概述

所谓**套接字**（Socket），就是对网络中不同主机上的应用进程之间进行双向通信的端点的抽象。如图1.3所示，从所处的地位来讲，套接字上联应用进程，下联网络协议栈，是应用程序通过网络协议栈进行通信的接口，是应用程序与网络协议栈进行交互的接口。



第一章：基于套接字的TCP/IP网络通信原理与模型

套接字是实现网络通信的基石，在采用基于套接字的网络通信过程中，其通信原理如下图1.4所示：当主机A上的应用程序进程A需要和主机B上的应用程序进程B进行通信时，进程A首先将一段信息写入Socket A，然后再由Socket A将该段信息通过TCP/IP网络发送到Socket B之中，最后再由Socket B将该段信息传送给应用程序进程B。



第一章：基于套接字的TCP/IP网络通信原理与模型

【王老师提问】Question 5：你觉得怎样才能标识一个套接字？

由以上描述可知，一个套接字可以看成是应用程序进程进行网间通讯的端点。而在网络环境下，一个应用程序进程又通常可用一个半相关<协议，本地地址，本地端口号>来进行唯一标识，因此，一个套接字显然也可以用上述半相关<协议，本地地址，本地端口号>来进行唯一标识，其中，二元组<本地地址，本地端口号>通常也称为**套接字的端点地址**。

通常将运行于客户端的套接字称为**客户端套接字**（Client Socket），而将运行于服务器端的套接字称为**服务器端套接字**（Server Socket）。

第一章：基于套接字的TCP/IP网络通信原理与模型

在实际网络通信中，将由客户端套接字提出连接请求，而要连接的目标就是服务器端套接字。为此，通常又将客户端套接字称为**主动套接字**，将服务器套接字称为**被动套接字**。

【王老师提问】Question 6：由于客户端套接字在向服务器端套接字提出连接请求之前，首先必须知道服务器端套接字的端点地址（即服务器的IP地址和服务器进程的端口号），你觉得客户端怎样才能知道服务器端套接字的端点地址？

为了让服务器套接字的端点地址预先被客户端知道，服务器端套接字必须使用**熟知（Well-Known）端口号**。

第一章：基于套接字的TCP/IP网络通信原理与模型

1.2.2 基于套接字的TCP/IP网络通信原理

为了形象地说明基于套接字的TCP/IP网络通信原理，本节以图1.4中所示的客户端主机A上的进程A与服务器端主机B上的进程B之间的网络通信过程为例来进行阐述。

1. 针对客户端主机A上的进程A

① TCP通信过程

进程A、B之间的TCP通信过程类似A、B之间的手机通话过程，其中：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ 客户端进程A ≌ 主叫方A

※ 服务器端进程B ≌ 被叫方B

※ 客户端套接字A ≌ 主叫方A的手机

※ 服务器端套接字B ≌ 被叫方B的手机

※ 客户端套接字A的端点地址 ≌ 主叫方A的手机号码

※ 服务器端套接字B的端点地址 ≌ 被叫方B的手机号码

步骤1（手机通话过程）：若主叫方A想要打电话给被叫方B，首先需要知道被叫方B的手机号码。

※ 对应的TCP通信过程：若客户端进程A想要与服务器端进程B通信，首先需要知道服务器端套接字B的端点地址。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤2（手机通话过程）：接下来，主叫方A还需要新购一台手机A。

※ 对应的TCP通信过程：客户端进程A还需要新建一个套接字A，即客户端套接字A。

步骤3（手机通话过程）：主叫方A还需要为新购的手机A新申请一个本地手机号码。

※ 对应的TCP通信过程：客户端进程A还需要新建的客户端套接字A申请一个本地端点地址。

步骤4（手机通话过程）：主叫方A利用手机A拨打被叫方B的手机B。

※ 对应的TCP通信过程：客户端进程A利用客户端套接字A向服务器端套接字B发送TCP连接建立请求。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤5（手机通话过程）：主叫方A拨通被叫方B的电话之后，与被叫方B之间进行手机通话。

※ 对应的TCP通信过程：客户端进程A与服务器端进程B在建立了TCP连接之后，与服务器端进程B之间进行TCP通信。

步骤6（手机通话过程）：通话结束后，主叫方A挂机。

※ 对应的TCP通信过程：通信结束后，客户端进程A关闭客户端套接字A以释放TCP连接以及与套接字相关的资源。

② UDP通信过程

进程A、B之间的UDP通信过程类似A、B之间的短信收发过程，其中：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ 客户端进程A ≌ 发信方A

※ 服务器端进程B ≌ 收信方B

※ 客户端套接字A ≌ 发信方A的手机

※ 服务器端套接字B ≌ 收信方B的手机

※ 客户端套接字A的端点地址 ≌ 发信方A的手机号码

※ 服务器端套接字B的端点地址 ≌ 收信方B的手机号码

步骤1（短信收发过程）：若发信方A想要发送短信给收信方B，首先需要知道收信方B的手机号码。

※ 对应的UDP通信过程：若客户端进程A想要与服务器端进程B通信，首先需要知道服务器端套接字B的端点地址。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤2（短信收发过程）：接下来，发信方A还需要新购一台手机A。

※ 对应的UDP通信过程：客户端进程A还需要新建一个套接字A，即客户端套接字A。

步骤3（短信收发过程）：发信方A还需要为新购的手机A新申请一个本地手机号码。

※ 对应的UDP通信过程：客户端进程A还需要新建的客户端套接字A申请一个本地端点地址。

步骤4（短信收发过程）：发信方A利用手机A与收信方B的手机B之间进行短信收发（与打电话不同，这里没有一个拨打电话的过程，发信方A不需要关心收信方B的手机是否开机与可以接通）。

第一章：基于套接字的TCP/IP网络通信原理与模型

※ 对应的UDP通信过程：客户端进程A利用客户端套接字A与服务器端进程B的套接字B进行UDP通信。（与TCP通信不同，这里没有一个建立连接的过程，客户端进程A不需要关心服务器端进程B及其套接字是正常工作）。

步骤5（短信收发过程）：短信收发结束后，发信方A关机。

※ 对应的UDP通信过程：短信发送结束后，客户端进程A关闭客户端套接字A以释放与套接字相关的资源。

二、针对服务器端主机B上的进程B

① TCP通信过程

进程A、B间的TCP通信过程类似A、B之间的手机通话过程，其中：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ 客户端进程A ≌ 主叫方A

※ 服务器端进程B ≌ 被叫方B

※ 客户端套接字A ≌ 主叫方A的手机

※ 服务器端套接字B ≌ 被叫方B的手机

※ 客户端套接字A的端点地址 ≌ 主叫方A的手机号码

※ 服务器端套接字B的端点地址 ≌ 被叫方B的手机号码

步骤1（手机通话过程）：若被叫方B想要接到主叫方A打过来的电话，首先被叫方B需要新购一台手机B。

※ 对应的TCP通信过程：若服务器端进程B想要与客户端进程A通信，首先需要新建一个套接字B，即服务器端套接字B。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤2（手机通话过程）：接下来，被叫方B还需要为新购的手机B新申请一个本地手机号码。

※ 对应的TCP通信过程：服务器端进程B需要为新建的服务器端套接字B新申请一个本地端点地址。

步骤3（手机通话过程）：为了能接收到主叫方A打过来的电话，被叫方B还必须使得手机B处于被叫待机模式（就是B必须得一直开机等候，因为B不知道A什么时候会打电话过来）。

※ 对应的TCP通信过程：服务器端进程B需要将服务器端套接字B设置为被动模式。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤4（手机通话过程）：当显示有电话拨入时，被叫方B利用手机B接通主叫方A利用手机A拨打过来的电话。

※ 对应的TCP通信过程：当收到客户的连接请求时，服务器端进程B利用服务器端套接字B与客户端进程A的客户端套接字A之间建立一个TCP连接。

步骤5（手机通话过程）：电话接通之后，被叫方B与主叫方A之间进行手机通话。

※ 对应的TCP通信过程：TCP连接建立之后，服务器端进程B与客户端进程A之间进行TCP通信。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤6（手机通话过程）：通话结束后，被叫方B挂机，继续等待下一个电话的到来。

※ 对应的TCP通信过程：通信结束之后，服务器端进程B关闭所建立的TCP连接，并返回步骤4以接受来自下一个客户的连接请求。

步骤7（手机通话过程）：被叫方B不想再接电话了，则关机。

※ 对应的TCP通信过程：服务器端进程B退出时，关闭该服务器端套接字B以释放与套接字相关的资源。

② UDP通信过程

进程A、B间的UDP通信过程类似A、B之间的短信收发过程，其中：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ 客户端进程A ≌ 发信方A

※ 服务器端进程B ≌ 收信方B

※ 客户端套接字A ≌ 发信方A的手机

※ 服务器端套接字B ≌ 收信方B的手机

※ 客户端套接字A的端点地址 ≌ 发信方A的手机号码

※ 服务器端套接字B的端点地址 ≌ 收信方B的手机号码

步骤1（短信收发过程）：若收信方B想要接到发信方A发送过来的短信，首先收信方B需要新购一台手机B。

※ 对应的UDP通信过程：若服务器端进程B想要与客户端进程A通信，首先需要新建一个套接字B，即服务器端套接字B。

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤2（短信收发过程）：接下来，收信方B还需要为新购的手机B新申请一个本地手机号码。

※ 对应的UDP通信过程：服务器端进程B还需要为新建的服务器端套接字B新申请一个本地端点地址。

步骤3（短信收发过程）：在开机状态下，当收到发信方A发送过来的短信时，收信方B利用手机B与发信方A的手机A之间进行短信收发。

※ 对应的UDP通信过程：在开机状态下，若收到客户端进程A发送过来的信息时，服务器端进程B利用服务器端套接字B与客户端进程A的客户端套接字A之间进行UDP通信。

步骤4（短信收发过程）：收信方B不想再收发短信了，则关机。

※ 对应的UDP通信过程：服务器端进程B退出，关闭服务器端套接字B以释放与套接字相关的资源。

【王老师提问】Question 7：请问，TCP与UDP通信过程的主要异同有哪些？

1.2.3 基于套接字的TCP/IP网络通信软件实现流程

根据上述基于套接字的TCP/IP网络通信原理，可给出基于套接字的TCP/IP网络通信软件的实现流程如下：

1. 基于套接字的TCP通信软件设计流程

（1）TCP客户算法的设计流程

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤1：找到期望通信的服务器套接字端点地址（IP地址+协议端口号）；

步骤2：创建本地客户端套接字；

步骤3：为该套接字申请一个本地端点地址（由TCP/IP协议软件自动选取）；

步骤4：建立该套接字到服务器套接字之间的一个TCP连接；

步骤5： 基于建立的TCP连接，与服务器进行通信（发送请求与等待应答）；

步骤6：通信结束之后，关闭该套接字以释放与之相关的资源（包括TCP连接的释放等）。

第一章：基于套接字的TCP/IP网络通信原理与模型

（2）TCP服务器算法的设计流程

步骤1：创建本地服务器端套接字；

步骤2：为该套接字申请一个本地端点地址（将该套接字绑定到它所提供服务的熟知端口上）；

步骤3：将该套接字设置为被动模式（被动套接字）；

步骤4：从该套接字上接受一个来自客户的连接请求，并建立与该客户之间的一个TCP连接；

步骤5：构造响应，并基于建立的TCP连接，与该客户进行通信（发送应答与等待请求）；

步骤6：与客户完成交互之后，关闭所建立的TCP连接，并返回步骤4以接受来自下一个客户的新的连接请求。

步骤7：当服务器关机时，关闭该套接字以释放与之相关的资源。

2. 基于套接字的UDP通信软件设计流程

（1）UDP客户算法的设计流程

步骤1：找到期望通信的服务器套接字端点地址（IP地址+协议端口号）；

步骤2：创建本地客户端套接字；

步骤3：为该套接字申请一个本地端点地址（由TCP/IP协议软件自动选取）；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤4：基于期望通信的服务器套接字端点地址，与服务器进行通信（发送请求与等待应答）；

步骤5：通信结束之后，关闭该套接字以释放与之相关的资源。

（2）UDP服务器算法的设计流程

步骤1：创建本地服务器端套接字；

步骤2：为该套接字申请一个本地端点地址（将该套接字绑定到它所提供服务的熟知端口上）；

步骤3：重复地读取来自客户的请求，然后构造响应，并按照应用协议与该客户进行通信（发送应答与等待请求）。

步骤4：当服务器关机时，关闭该套接字以释放与之相关的资源。

第一章：基于套接字的TCP/IP网络通信原理与模型

1.3 基于套接字的TCP/IP网络通信过程中的相关问题

1.3.1客户算法中服务器套接字端点地址查找问题

【王老师提问】**Question 8**：请问，你觉得客户软件可以有哪些方法来找到服务器端套接字的端点地址？

客户软件可用以下多种方法找到某个服务器套接字的端点地址：

（1）在编译程序时，将服务器套接字的端点地址说明为常量；

（2）要求用户在启动程序时输入服务器套接字的端点地址；

（3）从本地文件中获取服务器套接字端点地址的有关信息；

（4）通过某个组播或广播协议来查找服务器套接字的端点地址。

第一章：基于套接字的TCP/IP网络通信原理与模型

为了避免不必要的麻烦和对计算环境的依赖，大多数客户使用在启动客户程序时，要求用户输入服务器IP地址和协议端口号的方法，来获取某个服务器套接字的端点地址。按照这种方法来构建客户软件，不但可以使得客户软件更具一般性，而且还可以使得改变服务器的位置成为了可能。

1.3.2客户算法中本地端点地址的选择问题

在客户-服务器模型中，由于所有客户均需知道服务器端应用程序进程的端口号，因此，要求服务器端应用程序进程必须运行于某个熟知的协议端口之上，

第一章：基于套接字的TCP/IP网络通信原理与模型

【王老师提问】Question 9：对客户端应用程序进程而言，你认为协议端口号该如何分配？

对客户端应用程序进程而言，并不需要它工作于某个预分配的端口上，而只需要为其分配的端口号没有被其他的应用程序进程使用、且没有被预分配给某个熟知服务即可。

另外，对于拥有多个IP地址的主机，客户端应用程序进程往往难以进行IP地址的选择，这主要是因为正确的IP地址选择一般要依赖于选路信息，而应用程序却很少使用选路信息。

为此，为了解决上述客户算法中本地端点地址的分配问题，客户端应用程序进程选择将本地端点地址放置不填，改由TCP/IP协议软件自动地选取正确的本地IP地址与未使用的协议端口号来进行本地端点地址的填充。

1.3.3套接字端点地址的存储结构问题

为允许协议族自由地选择其地址表示方法，套接字抽象为每种类型的地址定义了一个地址族，一个协议族可以使用一种或多种地址族来定义其端点地址的表示方式。同时，套接字软件也为应用程序存储端点地址在头文件<sys/socket.h>中提供了以下预定义的C结构声明：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
struct sockaddr {        //来存储断电地址信息的结构
    u_char sa_len;       //表示整个 sockaddr 结构体的长度
    u_short sa_family;   //地址族（长度为 2 字节）
    char sa_data[14];    //14 字节的协议端点地址
};
```

在上述 sockddr 结构中，包含一个占 2 字节的地址族标识符和一个占 14 字节用于存储实际端点地址的数组。

第一章：基于套接字的TCP/IP网络通信原理与模型

上述sockaddr结构虽然可适用于TCP/IP协议族中的端点地址，但由于使用套接字的每个协议族都精确地定义了它的端点地址，例如：每个TCP/IP端点地址是由以下字段构成：一个用来标识地址类型的2字节字段、一个2字节的端口号字段、一个4字节的IP地址字段（IPv4），以及一个未使用的8字节字段；

因此，套接字软件在头文件<netinet/in.h>中还为TCP/IP协议族提供了以下预定义结构sockaddr_in来指明这种格式：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
struct sockaddr_in {
    u_char sin_len;                 //表示整个 sockaddr_in 结构体的长度
    short int sin_family;           //地址族（长度为 2 字节）
    unsigned short int sin_port;    //端口号（长度为 2 字节）
    struct in_addr sin_addr;        //存储 IP 地址的结构（长度为 4 字节）
    unsigned char sin_zero[8];      //填充 0 以保持与 struct sockaddr 同样大小
};
```

其中，存储 IP 地址的结构 struct in_addr 的定义如下：

```
struct in_addr {                    //存储 IP 地址的结构
    unsigned long s_addr;   //IP 地址
};
```

第一章：基于套接字的TCP/IP网络通信原理与模型

显然，只使用TCP/IP协议的应用程序可以只使用上述sockaddr_in结构，而无需使用sockaddr结构。另外，由于TCP/IP协议族（表示为PF_INET）中各协议均使用一种单一的地址表示方式，其地址族用符号AF_INET表示，因此，在上述sockaddr_in结构中，地址类型字段sin_family应赋值为AF_INET。

1.3.4客户-服务器模型中的汇聚点问题

在TCP/IP网络通信中，由于参与通信的两个应用程序进程一般位于两台处于不同地理位置的独立的计算机之上，因此将可能会导致以下**汇聚点**（Rendezvous）问题：

第一章：基于套接字的TCP/IP网络通信原理与模型

即：当两个人在分别启动这两个处于不同地理位置的独立的计算机上的应用程序进程时，由于两个人的操作速度不同，而计算机的运行速度要比人快许多数量级，这将导致速度快的那个人在启动了一个应用程序进程之后，将会判断出其对等应用程序进程还不存在，于是，它将发出一条错误消息然后退出。

【王老师提问】Question 10：请问，你觉得可以用什么方法来解决上述汇聚点问题？

为解决上述汇聚点问题，在C/S模型中，要求每一次通信均由客户进程随机启动，而服务器进程则必须处于无限循环等待状态，以等待来自客户的服务请求。

第一章：基于套接字的TCP/IP网络通信原理与模型

发起对等通信的一方称为客户，无限期地等待接收客户通信请求的一方则称为服务器。

1.3.5网络字节顺序与主机字节顺序问题

主机字节顺序是指占用内存多于一个字节类型的数据在主机的内存之中的存放顺序，不同的CPU有不同的字节顺序类型，通常有小端、大端两种字节顺序，其中：

小端字节顺序（Little Endian）是指低字节数据存放在内存的低地址处，高字节数据存放在内存的高地址处；

大端字节顺序（Big Endian）：与小端字节顺序相反。

【王老师提问】Question 11：请问，在网络通信中，如何才能解决通信双方可能存在的采用了不同的主机字节顺序的问题？

第一章：基于套接字的TCP/IP网络通信原理与模型

网络字节顺序是TCP/IP中规定好的一种数据表示格式，以保证数据在不同的主机之间传输时能够被正确解释。网络字节顺序采用的是Big Endian的排序方式。

为了进行主机字节顺序与网络字节顺序之间的转换，套接字软件提供了以下四个转换函数：

① **htons函数：**htons就是host-to-network-for type 'short'的意思，功能是把unsigned short类型的数据从主机字节顺序转换到网络字节顺序，调用成功时，将返回一个网络字节顺序的16位无符号短整型（unsigned short）值；若调用出错则返回-1。其函数原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

#include< netinet/in.h> //含有sockaddr_in结构与字节顺序转换函数的定义

uint16_t htons(uint16_t hostshort);

在上述htons()函数的原型中，各参数的含义如下：

※ **hostshort**：一个16位无符号短整型值。

② **htonl函数**：htonl就是host-to-network-for type 'long'的意思，该函数的功能是把unsigned long类型的数据从主机字节顺序转换到网络字节顺序，调用成功时返回一个网络字节顺序的32位无符号长整型（unsigned short）值；若调用出错则返回-1。其函数原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
#include< netinet/in.h>
uint32_t htonl(uint32_t hostlong);
```

在上述htonl()函数的原型中，各参数的含义如下：

※ **hostlong**：一个32位无符号长整型值。

③ **ntohs函数**：ntohs就是network-to-host-for type 'short'的意思，该函数的功能是把unsigned short类型的数据从网络字节顺序转换到主机字节顺序，调用成功时返回一个主机字节顺序的16位无符号短整型值；若调用出错则返回-1。其函数原型如下：

```
#include< netinet/in.h>
uint16_t ntohs(uint16_t netshort);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述ntohs()函数的原型中，各参数的含义如下：

※ **netshort**：一个16位无符号短整型值。

④ **ntohl函数**：ntohl就是network-to-host-for type 'long'的意思，该函数的功能是把unsigned long类型的数据从网络字节顺序转换到主机字节顺序，调用成功时返回一个主机字节顺序的32位无符号长整型值；若调用出错则返回-1。其函数原型如下：

```
#include< netinet/in.h>
uint32_t ntohl(uint32_t netlong);
```

在上述ntohl()函数的原型中，各参数的含义如下：

※ **netlong**：一个32位无符号长整型值。

第一章：基于套接字的TCP/IP网络通信原理与模型

1.3.6 IP地址与端口号的查找问题

在前面的介绍中指出，大多数客户使用在启动客户程序时，一般采用要求用户输入服务器IP地址和协议端口号的方法来获取服务器套接字的端点地址。用户在输入服务器的IP地址信息时，一般既可输入用点分十进制表示的IP地址（如：128.10.2.3），又可输入服务器的域名（如：merlin.cs.purdue.edu）；而在用户输入协议端口号时，既可输入用十进制表示的实端口号（如：23），又可输入该端口号所提供的服务名（如：TELNET）。为此，需要提供相关的系统函数以解决十进制IP地址转换为二进制IP地址、域名转换为二进制IP地址、以及服务名转换为协议端口号等问题。

第一章：基于套接字的TCP/IP网络通信原理与模型

（1）将点分十进制IP地址转换为网络字节顺序二进制IP地址

当用户输入的是用点分十进制表示的服务器的IP地址时，由于客户必须使用sockaddr_in结构来保存服务器的IP地址，因此，这就意味着客户需要将用点分十进制表示的服务器的IP地址转换为用二进制表示的32位的网络字节顺序的IP地址，套接字软件提供了库例程inet_addr、inet_aton、inet_ntoa来实现上述转换，各函数的原型如下：

① **inet_addr函数：**调用成功时返回一个用二进制表示的32位的网络字节顺序的IP地址；若调用出错则返回-1。其函数原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

#include< arpa/inet.h > //含有inet_addr, inet_aton, inet_ntoa等函数的定义

in_addr_t inet_addr(const char *cp);

在上述inet_addr()函数的原型中，各参数的含义如下：

※ **cp**：指向一个用点分十进制表示的IP地址字符串；

② **inet_aton函数**：调用成功时返回1，表示将cp指向的一个用点分十进制表示的IP地址字符串转换为了一个用二进制表示的32位的网络字节顺序的IP地址，转换后的IP地址存储在参数inp中；若调用出错则返回0。函数原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
#include< arpa/inet.h >
int inet_aton(const char * cp,struct in_addr * inp);
```

在上述inet_aton()函数的原型中，各参数的含义如下：

※ **cp**：指向一个用点分十进制表示的IP地址字符串；

※ **inp**：指向一个用二进制表示的32比特的IP地址结构。

③ **inet_ntoa函数：** 调用成功时返回一个指向字符串指针，表示将32位二进制形式的IP地址转换为用点分十进制形式表示的IP地址，结果在函数返回值中返回；若调用出错则返回NULL。其函数原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
#include< arpa/inet.h >
char * inet_ntoa(struct in-addr in);
```

在上述inet_ntoa()函数的原型中，各参数的含义如下：

※ **in**：指向一个用二进制表示的32比特的IP地址结构。

（2）由域名查找IP地址

当用户输入的不是服务器的IP地址，而是服务器的域名时，由于客户必须使用sockaddr_in结构保存服务器的IP地址，因此，这就意味着客户需要将服务器的域名转换为用二进制表示的32位的网络字节顺序的IP地址，套接字软件提供了库例程gethostbyname来实现上述转换。

第一章：基于套接字的TCP/IP网络通信原理与模型

该函数若调用成功时将返回一个指向包含有IP地址信息的HOSTENT结构指针；若调用出错则返回NULL。该函数的原型如下：

#include<netdb.h > //含有hostent结构与gethostbyname函数的定义

struct hostent * gethostbyname(const char * name);

在上述gethostbyname()函数的原型中，各参数的含义如下：

※ **name**：指向一个包含域名或主机名的字符串指针。

其中，HOSTENT结构的定义如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
struct hostent {
        char *h_name;          //主机的规范名称
        char **h_aliases;      //主机的别名
        char  h_addrtype;      /*主机 IP 地址的类型，如：是 IPv4(AF_INET)，还
                                 是 IPv6(AF_INET6)*/
        char h_length;         //主机 IP 地址的长度
        char **h_addr_list;    /*以网络字节序存储的主机的 IP 地址列表（一个主
                                 机可能会有多个 IP 地址）*/
};
#define h_addr h_addr_list[0]    /*指向主机 IP 地址列表中的第一个位置，这样
                                   应用程序就可以将 h_addr 当作 HOSTENT 结
                                   构中的一个字段来使用了*/
```

第一章：基于套接字的TCP/IP网络通信原理与模型

（3）由服务名查找端口号

当用户输入的不是服务器所提供的特定服务的协议端口号，而是服务器所提供的特定服务的服务名时，由于客户必须使用sockaddr_in结构保存服务器的端口号，因此，这就意味着客户需要将服务器所提供的特定服务的服务名转换为服务器所提供的特定服务的协议端口号，套接字软件提供了库例程getservbyname来实现上述转换，该函数若调用成功时将返回一个指向包含有协议端口号信息的SERVENT结构指针；若调用出错则返回NULL。该函数的原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

#include<netdb.h > //含有servent结构与getservbyname函数的定义

struct servent *getservbyname(const char *name, const char *proto);

在上述getservbyname()函数的原型中，各参数的含义如下：

※ **name**：指向一个包含服务器所提供的特定服务的服务名的字符串指针；

※ **proto**：连接该服务时用到的协议名。

其中，SERVENT结构的定义如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
struct servent {
    char *s_name;       //主机的规范名称
    char **s_aliases;   //主机的别名
    short s_port;       //以网络字节顺序存储的服务的协议端口号
    char *s_proto;      //连接该服务时用到的协议名
};
```

第一章：基于套接字的TCP/IP网络通信原理与模型

1.3.7由协议名查找协议号的问题

套接字软件提供一种机制，允许客户或服务器将协议名映射为分配给该协议的整数常量（也称为协议号）。库函数getprotobyname执行这种查找，调用该函数时以一个字符串参数的形式传递协议名，它返回的是一个protoent类型的结构地址，该函数若调用成功时将返回一个指向包含有协议号信息的PROTOENT结构指针；若调用出错则返回NULL。其原型如下：

#include<netdb.h > //含有protoent结构与getprotobyname函数的定义

struct protoent *getprotobyname(const char *name);

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述getprotobyname()函数的原型中，各参数的含义如下：

※name：指向一个包含协议名的字符串指针。

其中，PROTOENT结构的定义如下：

```
struct protoent {
    char *p_name;       //协议的规范名称
    char **s_aliases;   //协议的别名
    int p_proto;        //规范的协议号
};
```

第一章：基于套接字的TCP/IP网络通信原理与模型

1.3.8服务器算法中熟知端口的绑定问题

在服务器算法中，如果在将套接字绑定到某个端口号时，若它指定了某个特定的IP地址，则该套接字将只能接受客户发送到该IP地址上的请求，而不能接受客户发送到该机器其他IP地址上的请求。

然而，在很多时候路由器或多接口机器一般会拥有多个IP地址，显然，要想使得服务器可以接受客户发送到其所有IP地址的请求，则在套接字绑定到某个端口号时，不能只指定某个特定的IP地址。

【王老师提问】Question 12：请问，你觉得可以用什么方法来解决上述服务器可能具有的多IP地址的问题？

第一章：基于套接字的TCP/IP网络通信原理与模型

为了解决这个问题，套接字接口定义了一个特殊的常量——INADDR_ANY，它指明了一个通配地址（Wildcard Address），可以与该主机的任何一个IP地址都匹配。即：在服务器算法中当为套接字指明本地端点时，若服务器使用常量INADDR_ANY来取代某个特定的IP地址，则表示允许该套接字接收发给该机器上的任何一个IP地址的客户请求。

1.4套接字API 概述

1.4.1 BSD UNIX套接字API系统函数简介

第一章：基于套接字的TCP/IP网络通信原理与模型

应用程序接口（API，Application Programming Interface）又称为应用编程接口，是一组能用来操作组件、应用程序或者操作系统的函数，其主要目的是提供应用程序与开发人员以访问一组例程的能力，而又无需访问源码或理解内部工作机制的细节，从而使得程序员通过使用 API 函数开发应用程序时可以减轻编程任务。

在20世纪80年代初，美国加利福尼亚大学伯克利分校的研究人员为方便在BSD UNIX系统中实现TCP/IP网络通信而开发了一个专门用于网络通讯应用的API，该API就是套接字API（Socket API）。

第一章：基于套接字的TCP/IP网络通信原理与模型

BSD UNIX套接字API中包括了一个用C语言写成的**应用程序开发库**，主要用于实现进程之间的通讯，目前已在计算机网络通讯方面被广泛使用，形成了事实上的网络套接字标准。

BSD UNIX套接字API中提供的主要系统函数（也称为库函数）如下表1.1所示：

第一章：基于套接字的TCP/IP网络通信原理与模型

函数名	功能
socket	创建用于网络通信的套接字描述符
connect	连接远程对等实体（客户）
send	通过TCP连接外发数据
recv	从TCP连接中获得传入数据
close	终止通信并释放套接字描述符
bind	将本地IP地址和协议端口号绑定到套接字上（服务器）
listen	将套接字设置为被动模式，并设置排队的TCP连接个数（服务器）
accept	接收下一个传入连接（服务器）
recvfrom	接收下一个传入的数据报并记录其源端点地址
sendto	依据调用recvfrom预先记录下的端点地址，发送外发的数据报
shutdown	在一个或两个方向上终止TCP连接
getpeername	在连接到达后，从套接字中获得远程机器的端点地址
getsockopt	获得套接字的当前选项
setsockopt	改变/设置套接字的当前选项

第一章：基于套接字的TCP/IP网络通信原理与模型

（1）socket()函数：系统函数socket()用于创建一个新套接字，该套接字可用于网络通信。socket()调用若执行成功，将返回一个套接字描述符（即：一个整型的数值），若出错则返回-1。

socket()函数的原型如下：

```
#include<sys/types.h>   //提供各种数据类型的定义
#include<sys/socket.h>  //提供socket函数及数据结构的定义
int socket(int domain, int type, int protocol);
```

在上述socket()函数的原型之中，所包含的各个参数的含义如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **domain**：该参数用于指明建立socket所使用的协议族，通常赋值为符号常量PF_INET或AF_INET，表示互联网协议族（TCP/IP协议族）；

※ **type**：该参数用于指定创建该socket的应用程序所希望采用的通信服务类型。同一协议簇可能提供多种不同的服务类型，例如：TCP/IP协议族可提供数据流（TCP）和数据报（UDP）两种服务类型。为此，该参数通常赋值为符号常量SOCK_STREAM （表示数据流服务类型）或SOCK_DGRAM（表示数据报服务类型）。

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **protocol**：该参数用于指明该socket所使用的具体协议的协议号。由于有些协议簇中不止一种协议支持同一类型的服务，因此单纯用前面两个参数domain和type还不能惟一确定一个协议。但在TCP/IP协议族中，用domain和type这两个参数一般即可惟一确定一个协议，因此在TCP/IP协议族中，protocol参数通常赋值为0。

应用程序在调用socket()函数返回一个socket描述符后，socket()函数将建立一个socket，这里“建立一个Socket”实际上是意味着为一个socket数据结构分配了存储空间，而返回的socket描述符则是一个指向该内部数据结构的指针。

第一章：基于套接字的TCP/IP网络通信原理与模型

在通过socket()函数建立了一个socket之后，在使用该socket进行网络通信以前，还必须配置该socket。其中，面向连接的socket客户端是通过执行connect()函数来在socket数据结构中保存本地和远端信息的；而无连接的socket客户端和服务端以及面向连接的socket服务端则都是通过调用bind()函数来配置本地信息的。

（2）connect()函数：系统函数connect()用于配置socket并与远端服务器建立一个TCP连接，只有面向连接的客户程序使用socket时才需要调用connect()函数来将socket与远端主机相连。connect()函数执行成功返回一个整型数值，若出错返回-1。

第一章：基于套接字的TCP/IP网络通信原理与模型

connect()函数的原型如下：

```
#include<sys/types.h>
#include<sys/socket.h>
int connect(int sockfd, struct sockaddr* serv_addr,int addrlen);
```

在上述connect()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：套接字文件描述符，它是由系统函数socket()返回的。

※ **serv_addr**：指向数据结构sockaddr的指针，其中包含远端机器的端点地址（远端机器的端口号和IP地址）。

※ **addrlen**：远端地址结构serv_addr的长度，可以使用sizeof(struct sockaddr)来获得。

第一章：基于套接字的TCP/IP网络通信原理与模型

（3）bind()函数：系统函数bind()用于将socket与本机的一个端点地址（端口号+IP地址）相关联，bind()函数若调用成功，将返回一个整型数值，若出错则返回-1。bind()函数主要用于服务器端，其原型如下：

```
#include<sys/types.h>
#include<sys/socket.h>
int bind(int sockfd, truct sockaddr *my_addr, int addrlen);
```

在上述bind()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：是系统函数socket()返回的套接字描述符。

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **my_addr**：指向数据结构sockaddr的指针，其中包含本机的端点地址（本机的端口号和IP地址）。在给结构指针变量m_addr赋值时，可通过将my_addr.sin_port置为0来自动选择一个未占用的端口号，通过将my_addr.sin_addr.s_addr置为INADDR_ANY来自动填入本机IP地址。

※ **addrlen**：本机地址结构my_addr的长度，可以使用sizeof(struct sockaddr)来获得。

注：在调用bind()函数时，需要将sin_port和s_addr转换成为网络字节优先顺序。另外，调用bind()函数时一般不要将端口号设置为小于1024的值，因为1到1024是保留端口号

第一章：基于套接字的TCP/IP网络通信原理与模型

（4）listen()函数：系统函数listen()用于使socket处于被动的监听模式，并为该socket建立一个输入数据队列，将到达的客户服务请求保存在此队列中，直到程序处理它们。listen()函数若调用成功，将返回一个整型数值，若出错则返回-1。listen()函数的原型如下：

```
#include<sys/types.h>
#include<sys/socket.h>
int listen(int sockfd, int backlog);
```

在上述listen()函数的原型之中，所包含的各个参数的含义如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **sockfd**：是系统函数socket()返回的套接字描述符。

※ **backlog**：指定进入队列中所允许的连接的个数，进入队列的客户连接请求在使用系统调用accept()应答之前将在进入队列中等待。这个值是队列中最多可以拥有的客户请求的个数。大多数系统的缺省设置为20，也可以设置为5或10。如果一个客户请求到来时，输入队列已满，则socket将拒绝客户的连接请求，客户将收到一个出错信息。

第一章：基于套接字的TCP/IP网络通信原理与模型

（5）accept()函数：用于从等待连接队列（该等待连接队列为系统函数listen()所创建）中抽取第一个客户连接请求，然后建立与该客户之间的连接，并为该接连创建一个新的套接字，此后，就由这个新的套接字来负责通过与该客户之间的连接与该远端客户进行通讯。

如果队列中没有正在等待的客户连接，且套接字为阻塞方式，则accept()函数将阻塞调用进程直至新的客户连接请求出现；如果套接字为非阻塞方式且队列中没有正在等待的客户连接请求，则accept()函数将返回一个错误代码。accept()函数若调用成功，将返回一个新的套接字描述符，若出错则返回-1。

第一章：基于套接字的TCP/IP网络通信原理与模型

accept()函数的原型如下：

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

在上述accept()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：是系统函数socket()返回的套接字描述符。

※ **addr**：是一个用于回传的指向数据结构sockaddr的指针，accept()函数将从客户的连接请求之中自动抽取远端客户主机的端点地址信息并存入到该addr指针所指向的数据结构之中。

※ **addrlen**：远端地址结构addr的长度，可以使用sizeof(struct sockaddr)来获得。

第一章：基于套接字的TCP/IP网络通信原理与模型

（6）send()函数：系统函数send()用于给TCP连接的另一端发送数据，其中，客户程序一般用send()函数向服务器发送请求，而服务器则通常调用send()函数来向客户程序发送应答。send()函数只可用于基于连接的套接字，send()函数和write()函数唯一的不同点是标志的存在，当标志为0时，send()函数等同于write()函数。send()函数的原型如下：

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int s, const void *buf, size_t len, int flags);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述send()函数的原型之中，所包含的各个参数的含义如下：

※ **s**：指明用来发送数据的套接字描述符（如：系统函数accept()返回的套接字描述符）。

※ **buf**：指明一个存放应用程序要发送数据的缓冲区。

※ **len**：指明实际要发送的数据的字节数。

※ **flags**：指明调用执行方式，一般设置为0。

系统函数send()若调用出错将返回-1，若调用成功，则将返回实际发送出的字节数，该字节数可能会少于实际欲发送的数据的字节数len，在程序中应该将send()函数的返回值与实际欲发送的数据的字节数进行比较。当send()函数的返回值与len不匹配时，应对这种情况进行处理。

第一章：基于套接字的TCP/IP网络通信原理与模型

注：即便是已成功地调用了send()函数也并不意味着数据一定会正确地传送到达对端机器，因为send()函数仅仅只是在把buf中的数据成功拷贝到了s的发送缓冲区的剩余空间里之后就返回了，因此，如果协议在后续的数据传送过程中出现网络错误的话，那么send()函数并不能保证这些数据会被正确传到连接的另一端。

（7）recv()函数：系统函数recv()用于客户端和服务器程序从TCP连接的接收来自另一端的数据，recv()函数只可用于基于连接的套接字，recv()函数和read()函数唯一的不同点是标志的存在，当标志为0时，recv()函数等同于read()函数。recv()函数的原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
#include <sys/types.h>
#include <sys/socket.h>
int recv(int s, void *buf, int len, unsigned int flags);
```

在上述recv()函数的原型之中，所包含的各个参数的含义如下：

※ **s**：指明用来接收数据的套接字描述符。

※ **buf**：指明一个应用程序用来存放接收到的数据的缓冲区。

※ **len**：指明缓冲区的长度。

※ **flags**：指明调用执行方式，一般设置为0。

系统函数recv()若调用成功，将返回实际接收的字节数，若出现错误，则返回-1。

第一章：基于套接字的TCP/IP网络通信原理与模型

（8）sendto()函数：系统函数sendto()用于在无连接的数据报socket方式下进行数据的发送，由于在无连接的数据报socket方式下，本地socket并没有与远端机器之间建立连接，因此，在调用sendto()函数来发送数据时应指明目的机器的端点地址。与send()函数类似，sendto()函数也返回实际发送的数据字节长度或在出现发送错误时返回-1。sendto()函数的原型如下：

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockfd, const void *msg, int len, unsigned int
flags, const struct sockaddr *to, int tolen);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述sendto()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：指明用来发送数据的套接字描述符。

※ **msg**：指明一个存放应用程序要发送数据的缓冲区。

※ **len**：指明缓冲区的长度。

※ **flags**：指明调用执行方式，一般设置为0。

※ **to**：是一个指向数据结构sockaddr的指针，其中包含远端机器上的对等方套接字的端点地址。在客户端，该参数中所包含的远端机器的端点地址即为服务器的IP地址和熟知端口号；在服务器端，该参数即为recvfrom()函数中的参数from。

※ **tolen**：远端地址结构serv_addr的长度，可使用sizeof (struct sockaddr) 来获得。

第一章：基于套接字的TCP/IP网络通信原理与模型

（9）recvfrom()函数：系统函数recvfrom()用于实现在无连接的数据报socket方式下进行数据的接收，与recv()函数类似，recvfrom()函数也返回实际接收到的数据字节长度或在出现接收错误时返回-1。recvfrom()函数的原型如下：

```
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,
struct sockaddr *from, int *fromlen);
```

在上述recvfrom()函数的原型之中，所包含的各参数含义如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **sockfd**：指明用来接收数据的套接字描述符。

※ **buf**：指明一个存放应用程序用于接收数据的缓冲区。

※ **len**：指明缓冲区的长度。

※ **flags**：指明函数执行方式，一般设置为0。

※ **from**：是一个指向数据结构sockaddr的指针，用于存放远端机器上的对等方套接字的端点地址。其中，recvfrom()函数将从远端机器通过sendto()函数发送过来的数据报中自动抽取远端机器的端点地址信息存入到from指针所指向的数据结构之中。

※ **fromlen**：远端地址结构serv_addr的长度，可使用sizeof (struct sockaddr) 来获得。

第一章：基于套接字的TCP/IP网络通信原理与模型

（10）close()函数：系统函数close()用于在服务器与客户端之间的所有数据收发操作结束以后关闭套接字、以释放该套接字所占用的资源。close()函数若调用成功将返回0，若出错则返回-1。close()函数的原型如下：

#include <unistd.h> //提供通用的文件、目录、进程等操作的函数原型定义

#include <sys/socket.h>

int close(sockfd);

在上述close()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：指明要关闭的套接字描述符。

第一章：基于套接字的TCP/IP网络通信原理与模型

注：系统中的每一个文件或套接字都有一个引用计数，表示当前打开或者正在引用该文件或套接字的进程的个数。调用close()函数终止一个连接时，它只是减少了描述符的引用计数，并不直接关闭对应的连接，只有当描述符的引用计数为0时，才真正关闭该连接。

因此，在只有一个进程使用某个套接字时，close()函数会将该套接字的读写都关闭，这就容易导致发起关闭的一方还没有读完所有数据就关闭连接了，从而使得对方发来的数据将会被丢弃；

第一章：基于套接字的TCP/IP网络通信原理与模型

但如果有多个进程共享某个套接字时，则系统函数close()每被调用一次，都只是会使得其引用计数减1，而只有等到其引用计数为0时，也就是说，只有等到所用进程都调用了close()函数来关闭该套接字时，该套接字才会被最终释放；

为此，在有多个进程共享某个套接字的时候，当某个进程调用close()函数关闭了一个套接字之后，除了使得该进程将不能再访问该套接字之外，其它正在引用该套接字的进程均可继续正常使用该套接字与远端机器进行通信。

第一章：基于套接字的TCP/IP网络通信原理与模型

（11）shutdown()函数：系统函数shutdown()用于在套接字的某个方向上关闭数据传输，而一个方向上的数据传输仍可继续进行。例如：可以关闭某个套接字的写操作而允许继续在该套接字上接收数据，直至读入所有的数据。shutdown()函数若调用成功将返回0，若出错则返回-1。shutdown()函数的原型如下：

```
#include <sys/types.h>
#include <sys/socket.h>
int shutdown(int sockfd, int howto);
```

在上述shutdown()函数的原型之中，所包含各参数的含义如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **sockfd**：指明要关闭的套接字描述符。

※ **howto**：指明关闭方向，该参数的取值包括以下三种：

◆0：仅关闭读；套接字将不再接收任何数据，且将套接字接收到的缓冲区中的现有数据全部丢弃。

◆1：仅关闭写；当前留在缓冲区中的数据将被发送完，进程将不能够再对该套接字调用任何写函数。

◆2：同时关闭读和写；与close()函数的功能类似，不同的是shutdown()函数在关闭描述符时不考虑该套接字描述符的引用计数，而是直接关闭该套接字。

第一章：基于套接字的TCP/IP网络通信原理与模型

注：与close()函数不同，在有多个进程共享某个套接字的时候，如果一个进程调用了shutdown()函数关闭某个套接字之后，将会使得其它的所有进程都无法再使用该套接字进行通信。

（12）getpeername()函数：系统函数getpeername()用于获取所连接的远端机器上的对等方套接字的名称。getpeername()函数若执行成功将返回0，若出错则返回-1。getpeername()函数的原型如下：

```
#include <sys/types.h>
#include<sys/socket.h>
int getpeername(int sockfd, struct sockaddr* addr, int* addrlen);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述getpeername()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：指明连接的套接字描述符。

※ **addr**：是一个指向数据结构sockaddr的指针，用于存放远端机器上的对等方套接字的端点地址。

※ **addrlen**：远端地址结构serv_addr的长度，可使用sizeof (struct sockaddr) 来获得。

（13）setsockopt()函数：系统函数setsockopt()用于设置与某个套接字关联的选项。setsockopt()函数若执行成功将返回0，若出错则返回-1。setsockopt()函数的原型如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

```
#include <sys/types.h>
#include <sys/socket.h>
int setsockopt(int sock,int level,int optname,const void
*optval,socklen_t optlen);
```

在上述setsockopt()函数的原型之中，所包含的各个参数的含义如下：

※ **sockfd**：指明将要被设置选项的套接字描述符。

※ **level**：指明选项所在的协议层，选项可能存在于多层协议中，但它们总会出现在最上面的套接字层。因此，当操作套接字选项时，为了设置套接字层的选项，应将level的值指定为SOL_SOCKET。

第一章：基于套接字的TCP/IP网络通信原理与模型

※ **optname**：指明需要设置的选项名，SOL_SOCKET层所包含的选项如下表2.1所示。

※ **optval**：指向包含新选项值的缓冲区，该参数的类型将根据选项名称的数据类型进行转换。

※ **optlen**：选项值的长度。

第一章：基于套接字的TCP/IP网络通信原理与模型

选项名称	说明	数据类型
SO_BROADCAST	允许发送广播数据	int
SO_DEBUG	允许调试	int
SO_DONTROUTE	不查找路由	int
SO_ERROR	获得套接字错误	int
SO_KEEPALIVE	保持连接	int
SO_LINGER	延迟关闭连接	struct linger
SO_OOBINLINE	带外数据放入正常数据流	int
SO_RCVBUF	接收缓冲区大小	int
SO_SNDBUF	发送缓冲区大小	int
SO_RCVLOWAT	接收缓冲区下限	int
SO_SNDLOWAT	发送缓冲区下限	int
SO_RCVTIMEO	接收超时	struct timeval
SO_SNDTIMEO	发送超时	struct timeval
SO_REUSERADDR	允许重用本地地址和端口	int
SO_TYPE	获得套接字类型	int
SO_BSDCOMPAT	与BSD系统兼容	int

第一章：基于套接字的TCP/IP网络通信原理与模型

（14）getsockopt()函数：系统函数getsockopt()用于获取与某个套接字关联的选项。getsockopt()函数若调用成功将返回0，若出错则返回-1。getsockopt()函数的原型如下：

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockopt(int sockfd,int level,int optname,void
*optval,socklen_t *optlen);
```

在上述getsockopt()函数原型中，所包含各个参数的含义如下：

※ **sockfd**：指明将要被获取选项的套接字描述符。

※ **level**：指明选项所在的协议层，选项可能存在于多层协议中，但它们总会出现在最上面的套接字层。因此，当操作套接字选项时，为了获得套接字层的选项，应将level的值指定为SOL_SOCKET。

※ **optname**：指明需要访问的选项名，SOL_SOCKET层所包含的选项如表2.1所示。

※ **optval**：指向返回选项值的缓冲区，该参数的类型将根据选项名称的数据类型进行转换。

※ **optlen**：选项值的长度。

1.4.2 Windows套接字API 扩展系统函数简介

第一章：基于套接字的TCP/IP网络通信原理与模型

以U.C. Berkeley大学BSD UNIX中流行的Socket接口为范例，Microsoft也定义了一套Windows环境下的网络编程接口。在该接口中，不仅包含了上述BSD UNIX套接字API系统函数；而且还包含了一组针对Windows环境的扩展系统函数，以使开发人员能充分地利用Windows的消息驱动机制进行编程。

在Windows环境下，Socket是以DLL（动态链接库，Dynamic Link Library）的形式实现的，通常简称为Winsock DLL。

Windows套接字API中提供的主要系统函数与扩展系统函数分别如下表1.3和表1.4所示：

第一章：基于套接字的TCP/IP网络通信原理与模型

函数名	功能
socket	创建用于网络通信的套接字描述符
connect	连接远程对等实体（客户端）
send	通过TCP连接外发数据
recv	从TCP连接中获得传入数据
closesocket	终止通信并释放套接字描述符
bind	将本地IP地址和协议端口号绑定到套接字上（服务器）
listen	将套接字设置为被动模式，并设置排队的TCP连接个数（服务器）
accept	接收下一个传入连接请求（服务器专用）
recvfrom	接收下一个传入的数据报并记录其源端点地址
sendto	依据调用recvfrom预先记录下的端点地址，发送外发的数据报
shutdown	在一个或两个方向上终止TCP连接
getpeername	在连接到达后，从套接字中获得远程机器的端点地址
getsockopt	获得套接字的当前选项

第一章：基于套接字的TCP/IP网络通信原理与模型

（1）socket()函数：功能与对应参数的定义与1.4.1节中BSD UNIX套接字API中的socket()函数完全一致，函数原型如下：

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

（2）connect()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的connect()函数完全一致，其函数原型如下：

```
#include <winsock.h>
int connect (SOCKET s , const struct sockaddr FAR *name, int namelen);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

（3）bind()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的bind()函数完全一致，其函数原型如下：

```
#include <winsock.h>
int bind ( SOCKET s , const struct sockaddr FAR *addr, int namelen);
```

（4）listen()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的listen()函数完全一致，其函数原型如下：

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

第一章：基于套接字的TCP/IP网络通信原理与模型

（5）accept()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的accept()函数完全一致，其函数原型如下：

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

（6）send()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的send()函数完全一致，其函数原型如下：

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

第一章：基于套接字的TCP/IP网络通信原理与模型

（7）**recv()函数**：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的recv()函数完全一致，其函数原型如下：

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

（8）**sendto()函数**：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的sendto()函数完全一致，其函数原型如下：

```
#include <winsock.h>
int PASCAL FAR sendto( SOCKET s, const char FAR* buf, int len, int flags, const struct sockaddr FAR* to, int tolen);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

（9）recvfrom()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的recvfrom()函数完全一致，其函数原型如下：

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

（10）closesocket()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的close ()函数完全一致，其函数原型如下：

```
#include <winsock2.h>
BOOL PASCAL FAR closesocket(SOCKET s);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

（11）shutdown()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的shutdown()函数完全一致，其函数原型如下：

```
#include <winsock2.h>
int PASCAL FAR shutdown( SOCKET s, int how);
```

（12）getpeername()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的getpeername()函数完全一致，其函数原型如下：

```
#include <winsock2.h>
int PASCAL FAR getpeername( SOCKET s, struct sockaddr
FAR* name, int FAR* namelen);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

（13）setsockopt()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的setsockopt()函数完全一致，函数原型如下：

```
#include <winsock2.h>
int PASCAL FAR setsockopt( SOCKET s, int level, int optname,
const char FAR* optval, int optlen);
```

（14）getsockopt()函数：功能与参数的定义与1.4.1节中BSD UNIX套接字API中的getsockopt()函数完全一致，函数原型如下：

```
#include <winsock2.h>
Int PASCAL FAR getsockopt(SOCKET s, int level, int optname,
char FAR* optval, int FAR* optlen);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

表1.4 Windows套接字API中提供的主要扩展系统函数

函数名	功能
WSAStartup()	初始化Winsock DLL
WSAAsyncSelect()	通知套接字端口有请求事件发生
WSACancelAsyncRequest()	取消一次异步操作
WSAIsBlocking()	判断是否有阻塞调用正在进行
WSASetBlockingHook()	建立一个应用程序指定的阻塞钩子函数
WSAUnhookBlockingHook()	恢复缺省的阻塞钩子函数
WSACancelBlockingCall()	取消一次正在进行中的阻塞调用
WSAAsyncGetServByPort()	获得对应于一个端口号的服务信息
WSAAsyncGetServByName()	获得对应于一个服务名的服务信息
WSAAsyncGetProtoByNumber()	获得对应于一个协议号的协议信息
WSAAsyncGetProtoByName()	获得对应于一个协议名称的协议信息
WSAAsyncGetHostByName()	获得对应于一个主机名的主机信息
WSAAsyncGetHostByAddr()	获得对应于一个主机地址的主机信息
WSAGetLastError()	获得上次发生的网络错误
WSASetLastError()	设置可被WSAGetLastError()接收的错误代码

第一章：基于套接字的TCP/IP网络通信原理与模型

（1）**WSAStartup()**：用于初始化Winsock DLL（由于Winsock是以动态链接库的形式实现的，因此，在调用之前必须先要对其进行初始化，这也就是说，在Windows环境下，应用程序进程在调用其它任何Socket API系统函数之前必须首先成功调用WSAStartup()函数）。函数调用若执行成功将返回0，若出错则返回表1.5中所列的错误代码之一。函数原型如下：

```
#include <winsock2.h>
#include <windows.h>
#pragma comment(lib,"ws2_32.lib")
int PASCAL FAR WSAStartup(WORD wVersionRequested,
LPWSADATA lpWSAData);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述WSAStartup()函数的原型之中，所包含的各个参数的含义如下：

※wVersionRequested：指明欲使用的 Windows Socket API 版本，该值可以通过WORD MAKEWORD(BYTE,BYTE) 函数来获取，例：wVersionRequested =MAKEWORD(2,2)表示使用2.2版本的Windows Socket API。

※lpWSAData：指向WSAData数据结构的指针，用来保存函数WSAStartup()返回的Windows Socket初始化信息。

第一章：基于套接字的TCP/IP网络通信原理与模型

（2）**WSAAsyncSelect()**：用于通知套接字端口有请求事件发生。WSAAsyncSelect()函数调用若执行成功将返回0，若出错则返回SOCKET_ERROR，并可通过调用WSAGetLastError()函数返回特定的错误代码。WSAAsyncSelect()函数的原型如下：

```
#include <winsock2.h>
intPASCAL FAR WSAAsyncSelect(SOCKET s,
HWND hWnd,
unsigned int wMsg,
long lEvent);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

在上述WSAAsyncSelect()函数的原型之中，所包含的各个参数的含义如下：

※s：标识一个需要事件通知的套接口的描述符。

※hWnd：标识一个在网络事件发生时需要接收消息的窗口句柄。

※wMsg：在网络事件发生时要接收的消息。

※lEvent：位屏蔽码，用于指明应用程序感兴趣的网络事件集合。

其中，可能的网络事件如下表1.6所示：

第一章：基于套接字的TCP/IP网络通信原理与模型

可能的网络事件	含义
FD_READ	套接口s准备读
FD_WRITE	套接口s准备写
FD_OOB	带外数据准备好在套接口s上读
FD_ACCEPT	套接口s准备接收新的将要到来的连接
FD_CONNECT	套接口s上的连接完成
FD_CLOSE	由套接口s标识的连接已关闭

第一章：基于套接字的TCP/IP网络通信原理与模型

（14）WSAGetLastError()：系统函数WSAGetLastError()用于获得上次发生的网络错误。WSAGetLastError()函数的返回值为进行上一次Windows Socket API函数调用时产生的错误代码。

WSAGetLastError()函数的原型如下：

```
#include <winsock2.h>
int PASCAL FAR WSAGetLastError (void);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

（16）WSACleanup()：系统函数WSACleanup()用于解除与Socket库的绑定并终止Winsock DLL (Ws2_32.dll) 的使用，同时释放Socket库所占用的系统资源。WSACleanup()函数若调用成功将返回0，否则将返回SOCKET_ERROR，并可以通过调用WSAGetLastError()函数获取错误代码。WSACleanup()函数的原型如下：

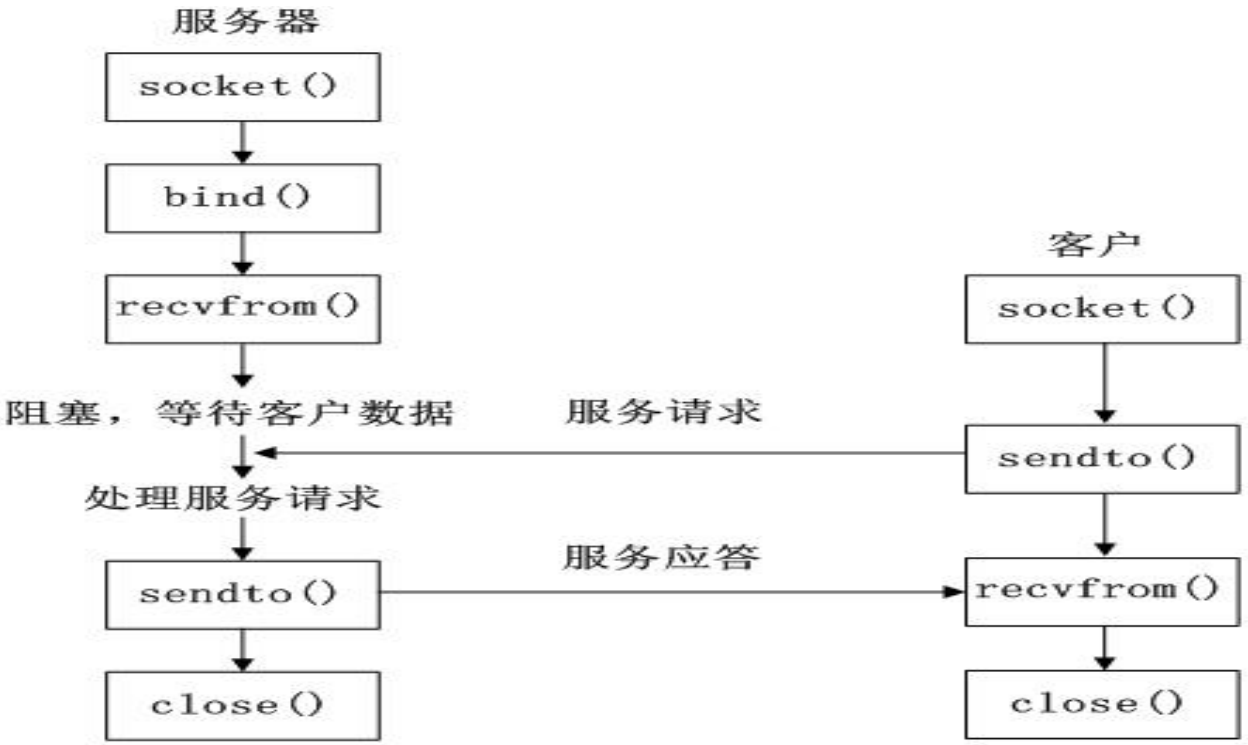
```
#include <winsock2.h>
int PASCAL FAR WSACleanup (void);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

1.5基于套接字的TCP/IP网络通信模型与实现方法

1.5.1 UNIX/LINUX环境下UDP套接字通信模型与实现方法

UNIX/LINUX环境下基于套接字的UDP通信模型可表示为下图1.5所示的形式：



第一章：基于套接字的TCP/IP网络通信原理与模型

由图1.5可知，在UNIX/LINUX环境下基于套接字的UDP通信模型中，服务器和客户端算法的实现流程可概略描述如下：

UDP服务器端算法的实现流程

（1）UDP服务器端算法的步骤描述

步骤1：调用socket()函数创建服务器端UDP套接字；

步骤2：调用bind()函数将该UDP套接字绑定到本机的一个可用的端点地址；

步骤3：调用recvfrom()函数从该UDP套接字接收来自远程客户端的数据并存入到缓冲区中，同时获得远程客户端的套接字端点地址并保存；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤4：基于保存的远程客户端的套接字端点地址，调用sendto()函数将缓冲区中的数据从该UDP套接字发送给该远程客户端；

步骤5：与客户交互完毕，调用close()函数将该UDP套接字关闭。

（2）UDP服务器端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int msock;                              //声明套接字描述符变量
msock = socket(AF_INET, SOCK_DGRAM, 0); //调用socket()函数创建套接字
if (msock<0){                           //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
#define SERVER_PORT 10000          //定义端口号为10000
int ret;
struct sockaddr_in servaddr;       //声明端点地址结构体变量
memset(&servaddr,0,sizeof(struct sockaddr_in));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;       //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);  //给IP地址字段赋值
servaddr.sin_port=htons(SERVER_PORT);     //给端口号字段赋值
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用bind()函数将套接字与端点地址绑定*/
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct
sockaddr_in));
if(ret<0){                              //调用bind()函数出错
    printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
    exit(-1);
}
```

③ 步骤3的C语言实现方法

```
#define BUFSIZE 4096                    //定义数据缓冲区大小为4M
char buf[BUFSIZE];                      //声明数据缓冲区变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
int num=0;
struct sockaddr_in clientaddr;          //声明端点地址结构体变量
int len = sizeof(clientaddr);
memset(&clientaddr,0,sizeof(struct sockaddr_in));
memset(buf,'\0',sizeof(buf));
/*以下语句用于调用recvfrom()函数从套接字接收客户端发来的
数据*/
num=recvfrom(msock,buf,sizeof(buf),0,(struct
sockaddr*)&clientaddr,&len);
if(num<0){                              //调用recvfrom()函数出错
    printf("Receive Data Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

④ 步骤4的C语言实现方法

```
num=0;
/*以下语句用于调用sendto()函数从套接字发送数据给客户端*/
num=sendto(msock, buf, strlen(buf),0,(struct
sockaddr*)&clientaddr, len);
if(num != strlen(buf)){                    //调用sendto()函数出错
      printf("Send Data Failed!\n");
      exit(-1);
}
```

⑤ 步骤5的C语言实现方法

```
close(msock);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

UDP客户端算法的实现流程

（1）UDP客户端算法的步骤描述

步骤1：调用socket()函数创建客户端UDP套接字；

步骤2：找到期望与之通信的远程服务器的IP地址和协议端口号；然后再调用sendto()函数将缓冲区中的数据从UDP套接字发送给远程服务器端；

步骤3：调用recvfrom()函数从该UDP套接字接收来自远程服务器端的数据并存入缓冲区中；

步骤4：与服务器交互完毕，调用close()函数将该UDP套接字关闭，释放所占用的系统资源。

第一章：基于套接字的TCP/IP网络通信原理与模型

（2）UDP客户端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int tsock;                              //声明套接字描述符变量
tsock = socket(AF_INET, SOCK_DGRAM, 0);  //调用socket()函数创建套接字
if (tsock<0){                           //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
#define SERVERIP "172.0.0.1"            //定义IP地址常量
#define SERVER_PORT 10000               //定义端口号为10000
char *buffer = "HELLO!";                //声明数据缓冲区变量
struct sockaddr_in servaddr;            //声明端点地址结构体变量
memset(&servaddr,0,sizeof(struct sockaddr_in));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;            //给协议族字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);     //给IP地址字段赋值
servaddr.sin_port = htons(SERVERPORT);      //给端口号字段赋值
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
int num=0;
int len = sizeof(struct sockaddr_in);
/*以下语句用于调用sendto()函数从套接字发送数据给服务器端*/
num=sendto(tsock, buffer, strlen(buffer),0,(struct
sockaddr*)&servaddr, len);
if(num != strlen(buf)){                    //调用sendto()函数出错
    printf("Send Data Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

③ 步骤3的C语言实现方法

```
#define BUFSIZE 4096                    //定义数据缓冲区大小为4M
char buf[BUFSIZE];                      //声明数据缓冲区变量
int num=0;
struct sockaddr_in clientaddr;          //声明端点地址结构体变量
memset(&clientaddr,0,sizeof(struct sockaddr_in));
memset(buf,'\0',sizeof(buf));
/*以下语句调用recvfrom() 从套接字接收服务器端发来的数据*/
num=recvfrom(tsock,buf,sizeof(buf),0,(struct
sockaddr*)&clientaddr,&len);
```

```
if(num<0){                          //调用recvfrom()函数出错
    printf("Receive Data Failed!\n");
    exit(-1);
}
```

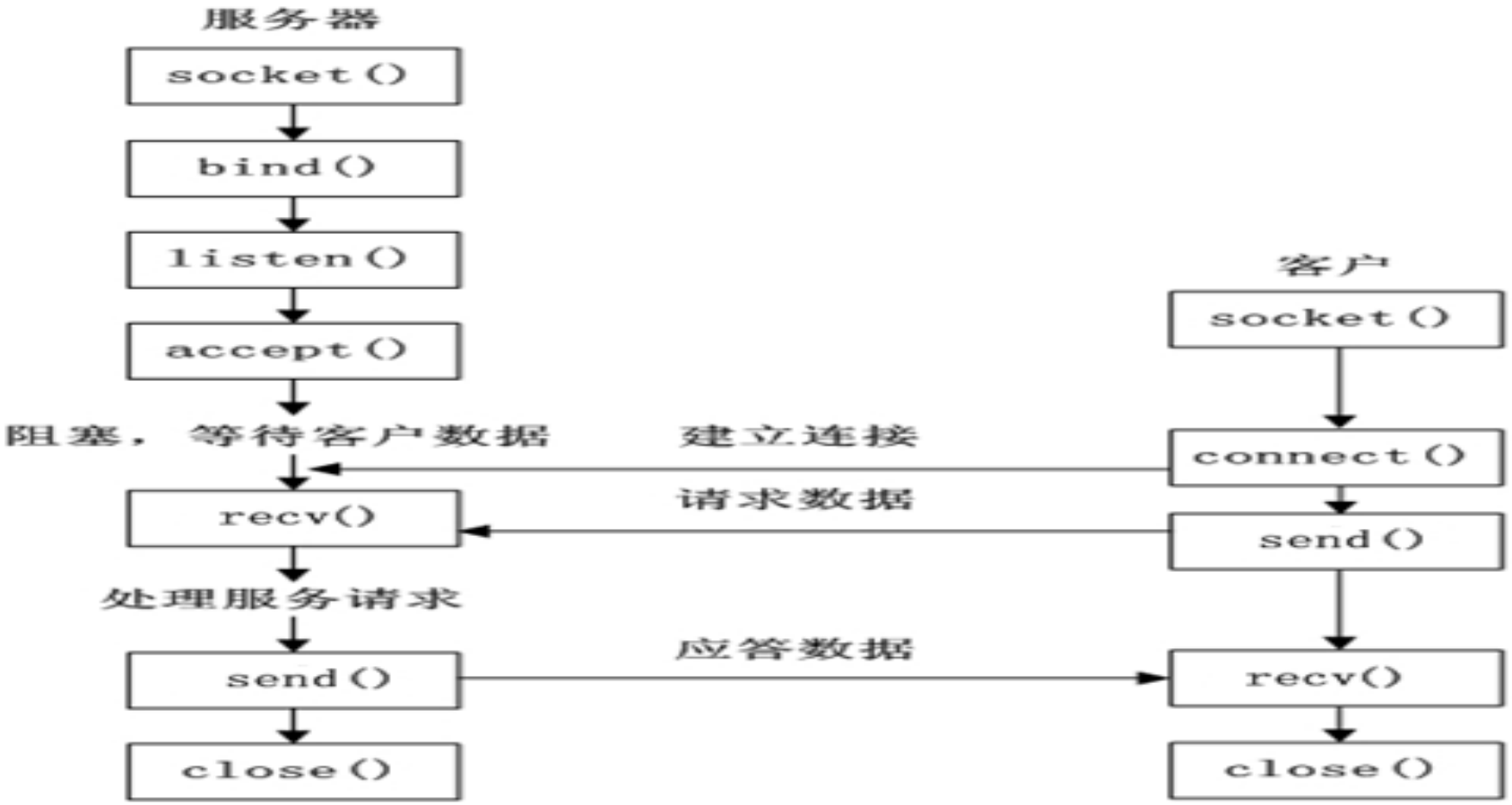
④ 步骤4的C语言实现方法

```
close(tsock);
```

1.5.2 UNIX/LINUX环境下TCP套接字通信模型与实现方法

基于前述TCP客户端与服务器算法的设计流程以及BSD UNIX套接字API系统函数的介绍易知，UNIX/LINUX环境下基于套接字的TCP通信模型可表示为下图1.6所示的形式：

第一章：基于套接字的TCP/IP网络通信原理与模型



由图1.6可知，在UNIX/LINUX环境下基于套接字的TCP通信模型中，服务器和客户端算法的实现流程可概略描述如下：

第一章：基于套接字的TCP/IP网络通信原理与模型

TCP服务器端算法的实现流程

（1）TCP服务器端算法的步骤描述

步骤1：调用socket()函数创建服务器端TCP主套接字；

步骤2：调用bind()函数将该TCP套接字绑定到本机的一个可用的端点地址；

步骤3：调用listen()函数将该TCP套接字设为被动模式，并设置等待队列的长度；

步骤4：调用accept()函数从该TCP套接字上接受一个新客户连接请求，并且在与该客户之间成功建立了TCP连接之后，为该TCP连接创建一个新的从套接字（由该新套接字来负责与客户之间进行实际的通信）；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤5：基于新创建的从套接字，调用recv()函数从套接字读取客户发送过来的数据并存入缓冲区中；

步骤6：基于新创建的从套接字，调用send()函数将缓冲区中的数据从套接字发送给该远程客户；

步骤7：与客户交互完毕，调用close()函数将从套接字关闭，释放所占用的系统资源。

步骤8：与所有客户交互完毕，调用close()函数将主套接字关闭，释放所占用的系统资源。

第一章：基于套接字的TCP/IP网络通信原理与模型

（2）TCP服务器端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int msock;                                  //声明主套接字描述符变量
msock = socket(AF_INET,SOCK_STREAM,0);  //调用socket()函数创建套接字
if (msock<0){                               //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
#define SERVER_PORT 10000          //定义端口号为10000
int ret;
struct sockaddr_in servaddr;       //声明端点地址结构体变量
memset(&servaddr,0,sizeof(struct sockaddr_in));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;       //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);  //给IP地址字段赋值
servaddr.sin_port=htons(SERVER_PORT);    //给端口号字段赋值
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用bind()函数将主套接字与端点地址绑定*/
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct
sockaddr_in));
if(ret<0){                              //调用bind()函数出错
    printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
    exit(-1);
}
```

③ 步骤3的C语言实现方法

```
#define QUEUE 20                        //定义等待队列长度为20
int ret;
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用listen()函数设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, QUEUE);
if(ret<0){                          //调用listen()函数出错
    printf("Listen Failed!\n");
    exit(-1);
}
```

④ 步骤4的C语言实现方法

```
int sscok;                          //声明从套接字描述符变量
struct sockaddr_in clientaddr;          //声明端点地址结构体变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
int len = sizeof(clientaddr);
memset(&clientaddr,0,sizeof(struct sockaddr_in));
/*以下语句用于调用accept()函数接受客户连接请求并创建从套接字*/
ssock = accept(msock, (struct sockaddr *)&clientaddr, &len);
if(ssock<0){                              //调用accept()函数出错
    printf("Accept Failed!\n");
    exit(-1);
}
```

⑤ 步骤5的C语言实现方法

```
#define BUFSIZE 4096                      //定义缓冲区大小为4M
char buf[BUFSIZE];                        //声明数据缓冲区变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
memset(buf,'\0',sizeof(buf));
int num;
/*以下语句用于调用recv()函数利用从套接字接收客户端发来的数据*/
num=recv(ssock,buf, sizeof(buf), 0);
if (num<0){
      printf("Recieve Data Failed!\n");
      exit(-1);
}
```

⑥ 步骤6的C语言实现方法

```
char *buffer = "HELLO!";                    //声明数据缓冲区变量
num=0;
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用send()函数利用从套接字发送数据给客户端*/
num= send(ssock,buffer,strlen(buffer),0);
if(num!=strlen(buffer)){                    //调用send()函数出错
    printf("Send Data Failed!\n");
    exit(-1);
}
```

⑦ 步骤7的C语言实现方法

```
close(ssock);
```

⑧ 步骤8的C语言实现方法

```
close(msock);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

TCP客户端算法的实现流程

（1）TCP客户端算法的步骤描述

步骤1：调用socket()函数创建客户端TCP套接字；

步骤2：找到期望与之通信的远程服务器端套接字的端点地址（即服务器端的IP地址和协议端口号）；然后，调用connect()函数向远程服务器端发起TCP连接建立请求；

步骤3：在与服务器端成功地建立了TCP连接之后，调用send()函数将缓冲区中的数据从套接字发送给该远程服务器端；

步骤4：调用recv()函数从套接字读取服务器端发送过来的数据并存入缓冲区中；

步骤5：与服务器端交互完毕，调用close()函数将套接字关闭，释放所占用的系统资源。

第一章：基于套接字的TCP/IP网络通信原理与模型

（2）TCP客户端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int tsock;                                  //声明套接字描述符变量
tsock = socket(AF_INET,SOCK_STREAM,0);  //调用socket()函数创建套接字
if (tsock<0){                               //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
#define SERVERIP "172.0.0.1"          //定义IP地址常量
#define SERVERPORT 10000              //定义端口号常量
struct sockaddr_in servaddr;          //声明端点地址结构体变量
memset(&servaddr,0,sizeof(struct sockaddr_in));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;          //给协议族字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);   //给IP地址字段赋值
servaddr.sin_port = htons(SERVERPORT);    //给端口号字段赋值
```

第一章：基于套接字的TCP/IP网络通信原理与模型

/*以下语句用于调用connect()函数向远程服务器发起TCP连接建立请求*/

```
int ret;
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                              //调用connect()函数出错
    printf("Connect Failed!\n");
    exit(-1);
}
```

③ 步骤3的C语言实现方法

```
char *buffer = "HELLO!";                //声明数据缓冲区变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
int num=0;
/*以下语句用于调用send()函数利用从套接字发送数据给服务器端
*/
num= send(tsock,buffer,strlen(buffer),0);
if(num!=strlen(buffer)){                    //调用send()函数出错
    printf("Send Data Failed!\n");
    exit(-1);
}
```

④ 步骤4的C语言实现方法

```
#define BUFSIZE 4096                        //定义缓冲区大小为4M
char buf[BUFSIZE];                          //声明数据缓冲区变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
memset(buf,'\0',sizeof(buf));
num=0;
/*以下语句用于调用recv()函数利用从套接字接收服务器端发来的数据*/
num=recv(tsock,buf, sizeof(buf), 0);
if (num<0){
      printf("Recieve Data Failed!\n");
      exit(-1);
}
```

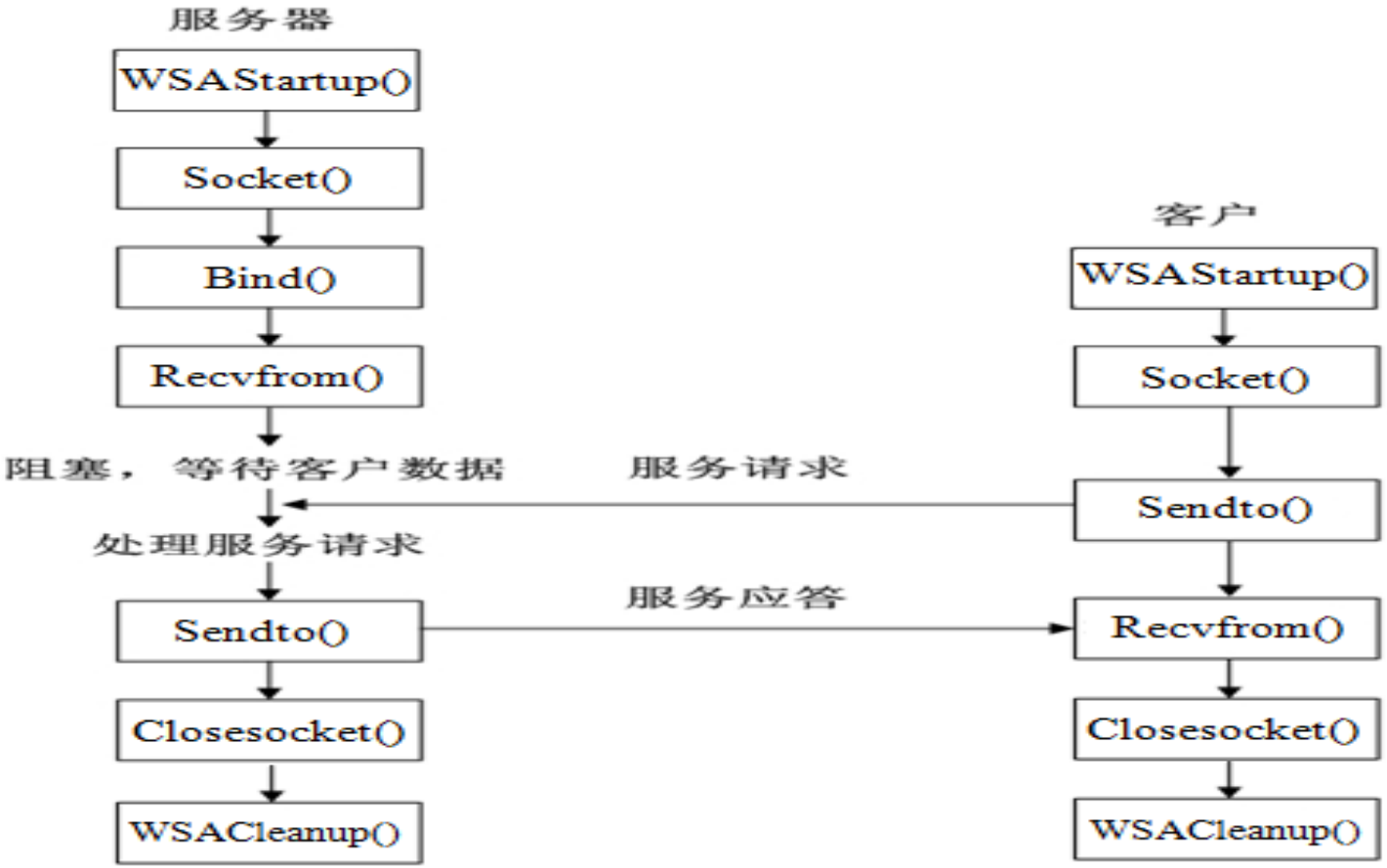
⑤ 步骤5的C语言实现方法

```
close(tsock);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

1.5.3 Windows环境下UDP套接字通信模型与实现方法

Windows环境下基于套接字的UDP通信模型可表示为下图1.7所示的形式：



第一章：基于套接字的TCP/IP网络通信原理与模型

由图1.7可知，在Windows环境下基于套接字的UDP通信模型中，服务器和客户端算法的实现流程可概略描述如下：

1. UDP服务器端算法的实现流程

（1）UDP服务器端算法的步骤描述：

步骤1：使用WSAStartup()函数初始化Winsock DLL；

步骤2：调用socket()函数创建服务器端UDP套接字；

步骤3：调用bind()函数将该UDP套接字绑定到本机的一个可用的端点地址；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤4：调用recvfrom()函数从该UDP套接字接收来自远程客户端的数据并存入到缓冲区中，同时获得远程客户端的套接字端点地址并保存；

步骤5：基于保存的远程客户端的套接字端点地址，调用sendto()函数将缓冲区中的数据从该UDP套接字发送给该远程客户端；

步骤6：与客户交互完毕，调用closesocket()函数将该UDP套接字关闭，释放所占用的系统资源；

步骤7：最后，调用WSACleanup()函数结束Winsock Socket API。

第一章：基于套接字的TCP/IP网络通信原理与模型

（2）UDP服务器端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int ret;
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
    printf("Couldn't Find a Useable Winsock.dll!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
SOCKET msock;                          //声明套接字描述符变量
msock = socket(AF_INET, SOCK_DGRAM, 0); //调用socket()函数创建套接字
if(msock == INVALID_SOCKET){           //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

③ 步骤3的C语言实现方法

```
#define SERVER_PORT 10000              //定义端口号为10000
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
int ret;
struct sockaddr_in servaddr;                    //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));    /*ZeroMemory()函数的功能与memset()函数类似*/
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;                    //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);  //给IP地址字段赋值
servaddr.sin_port=htons(SERVER_PORT);    //给端口号字段赋值
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用bind()函数将套接字与端点地址绑定*/
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct
sockaddr_in));
if(ret<0){                              //调用bind()函数出错
    printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
    exit(-1);
}
```

④ 步骤4的C语言实现方法

```
#define BUFSIZE 4096                    //定义数据缓冲区大小为4M
char buf[BUFSIZE];                      //声明数据缓冲区变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
int num=0;
struct sockaddr_in clientaddr;          //声明端点地址结构体变量
int len = sizeof(clientaddr);
ZeroMemory(&clientaddr,sizeof(clientaddr));
ZeroMemory(buf,sizeof(buf));
/*以下语句用于调用recvfrom()从套接字接收客户端发来的数据*/
num=recvfrom(msock,buf,sizeof(buf),0,(struct
sockaddr*)&clientaddr,&len);
if(num<0){                              //调用recvfrom()函数出错
    printf("Receive Data Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

⑤ 步骤5的C语言实现方法

```
num=0;
/*以下语句用于调用sendto()函数从套接字发送数据给客户端*/
num=sendto(msock, buf, strlen(buf),0,(struct
sockaddr*)&clientaddr, len);
if(num != strlen(buf)){                    //调用sendto()函数出错
      printf("Send Data Failed!\n");
      exit(-1);
}
```

⑥ 步骤6的C语言实现方法

```
closesocket(msock);
```

⑦ 步骤7的C语言实现方法

```
 WSACleanup();
```

第一章：基于套接字的TCP/IP网络通信原理与模型

2. UDP客户端算法的实现流程

（1）UDP客户端算法的步骤描述

步骤1：使用WSAStartup()函数初始化Winsock DLL；

步骤2：调用socket()函数创建客户端无连接套接字；

步骤3：找到期望与之通信的远程服务器的IP地址和协议端口号；然后再调用sendto()函数将缓冲区中的数据从套接字发送给远程服务器；

步骤4：调用recvfrom()函数从套接字接收来自远程服务器端的数据并存入缓冲区中；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤5：与服务器交互完毕，调用close()函数将套接字关闭，释放所占用的系统资源；

步骤6：最后，调用WSACleanup()函数结束Winsock Socket API。

② UDP客户端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int ret;
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
    printf("Couldn't Find a Useable Winsock.dll!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
SOCKET tsock;                          //声明套接字描述符变量
tsock = socket(AF_INET, SOCK_DGRAM, 0);  //调用socket()函数创建套接字
if (tsock == INVALID_SOCKET){          //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

③ 步骤3的C语言实现方法

```
#define SERVERIP "172.0.0.1"           //定义IP地址常量
#define SERVER_PORT 10000              //定义端口号为10000
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
char *buffer = "HELLO!";                    //声明数据缓冲区变量
struct sockaddr_in servaddr;                //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;                //给协议族字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);     //给IP地址字段赋值
servaddr.sin_port = htons(SERVERPORT);      //给端口号字段赋值
int num=0;
int len = sizeof(struct sockaddr_in);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用sendto()函数从套接字发送数据给服务器端*/
num=sendto(tsock, buffer, strlen(buffer),0,(struct
sockaddr*)&servaddr, len);
if(num != strlen(buf)){                    //调用sendto()函数出错
    printf("Send Data Failed!\n");
    exit(-1);
}
```

④ 步骤4的C语言实现方法

```
#define BUFSIZE 4096                  //定义数据缓冲区大小为4M
char buf[BUFSIZE];                   //声明数据缓冲区变量
int num=0;
struct sockaddr_in clientaddr;          //声明端点地址结构体变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
ZeroMemory(&clientaddr,sizeof(clientaddr));
ZeroMemory(buf,sizeof(buf));
/*以下语句用于调用recvfrom()函数从套接字接收服务器端发来的数据*/
num=recvfrom(tsock,buf,sizeof(buf),0,(struct sockaddr*)&clientaddr,&len);
if(num<0){                          //调用recvfrom()函数出错
    printf("Receive Data Failed!\n");
    exit(-1);
}
```

⑤ 步骤5的C语言实现方法

```
closesocket(tsock);
```

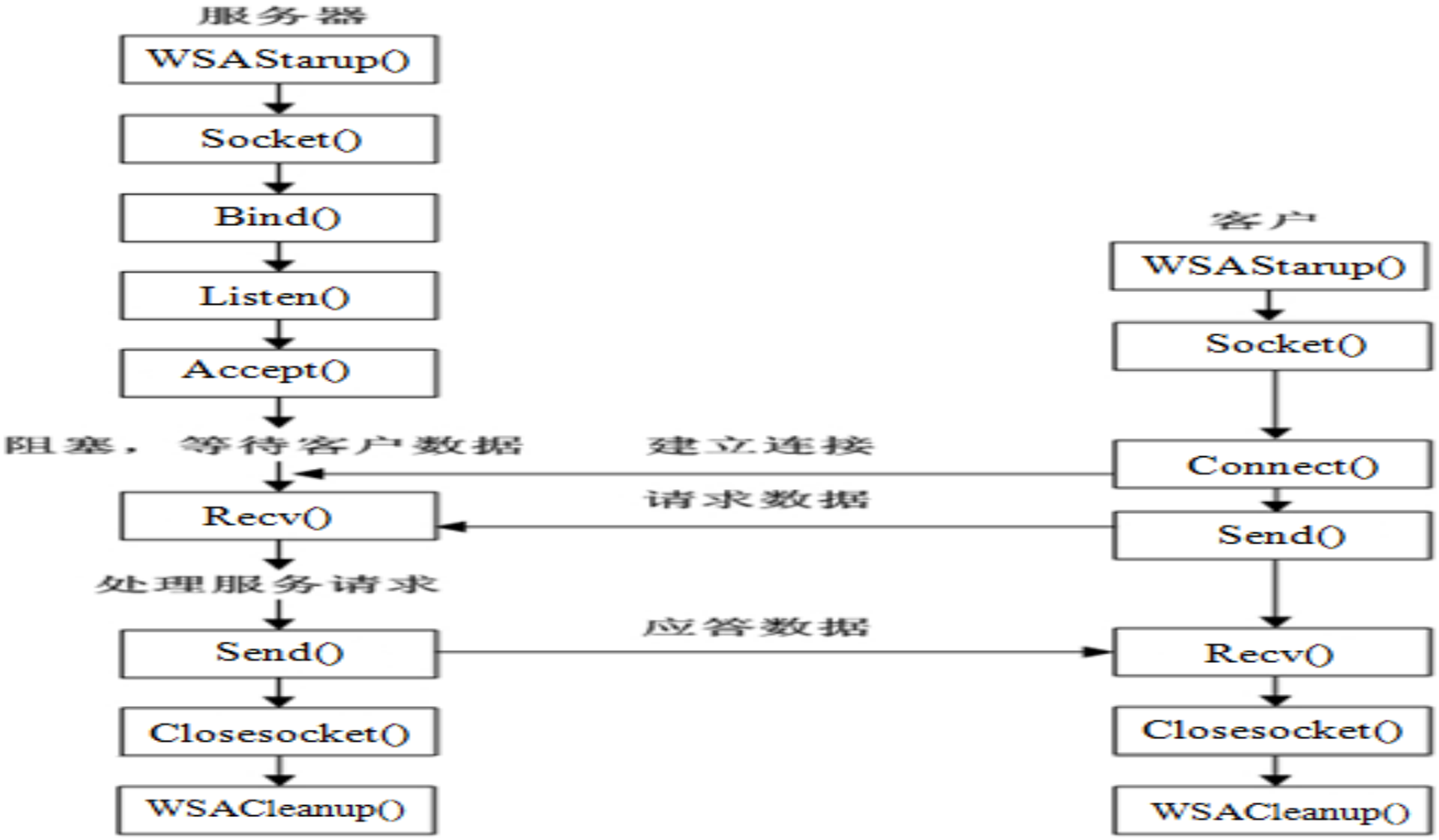
⑥ 步骤6的C语言实现方法

```
WSACleanup();
```

第一章：基于套接字的TCP/IP网络通信原理与模型

1.5.4 Windows环境下TCP套接字通信模型与实现方法

Windows环境下基于套接字的TCP通信模型可表示为下图1.8所示的形式。



第一章：基于套接字的TCP/IP网络通信原理与模型

由图1.8可知，在Windows环境下基于套接字的TCP通信模型中，服务器和客户端算法的实现流程可概略描述如下：

1. TCP服务器端算法的实现流程

（1）TCP服务器端算法的步骤描述

步骤1：使用WSAStartup()函数初始化Winsock DLL；

步骤2：调用socket()函数创建服务器端TCP主套接字；

步骤3：调用bind()函数将TCP主套接字绑定到本机的一个可用的端点地址；

步骤4：调用listen()函数将该TCP主套接字设为被动模式，并设置等待队列长度；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤5：调用accept()函数从该TCP主套接字上接受一个新的客户TCP连接请求，并在与该客户之间成功建立了TCP连接之后，为该TCP连接创建一个新的从套接字（由该新的从套接字来负责与客户之间进行实际的通信）；

步骤6：基于新创建的从套接字，调用recv()函数利用该从套接字读取客户端发送过来的数据并存入缓冲区中；

步骤7：基于新创建的从套接字，调用send()函数将缓冲区中的数据利用该从套接字发送给该远程客户端；

步骤8：与客户交互完毕，调用closesocket()函数将该从套接字关闭，释放所占用的系统资源；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤9：最后，当与所有客户交互完毕之后，调用closesocket()函数将TCP主套接字关闭，释放所占用的系统资源，然后，再调用WSACleanup()函数来结束Winsock Socket API。

（2）TCP服务器端算法的C语言实现方法

① 步骤1的C语言实现方法

```
int ret;
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
    printf("Couldn't Find a Useable Winsock.dll!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
SOCKET msock;                          //声明主套接字描述符变量
msock = socket(AF_INET,SOCK_STREAM,0);  //调用socket()函数创建套接字
if (msock == INVALID_SOCKET){          //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

③ 步骤3的C语言实现方法

```
#define SERVER_PORT 10000              //定义端口号为10000
int ret;
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
struct sockaddr_in servaddr;                    //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;                    //给协议族字段赋值
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);  //给IP地址字段赋值
servaddr.sin_port=htons(SERVER_PORT);    //给端口号字段赋值
/*以下语句用于调用bind()函数将主套接字与端点地址绑定*/
ret=bind(msock, (struct sockaddr *)&servaddr, sizeof(struct sockaddr_in));
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
if(ret<0){                              //调用bind()函数出错
    printf("Server Bind Port: %d Failed!\n", SERVER_PORT);
    exit(-1);
}
```

④ 步骤4的C语言实现方法

```
#define QUEUE 20                        //定义等待队列长度为20
int ret;
/*以下语句用于调用listen()函数设置等待队列长度和设套接字为被动模式*/
ret=listen(msock, QUEUE);
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
if(ret<0){                          //调用listen()函数出错
    printf("Listen Failed!\n");
    exit(-1);
}
```

⑤ 步骤5的C语言实现方法

```
SOCKET sscok;                       //声明从套接字描述符变量
struct sockaddr_in clientaddr;      //声明端点地址结构体变量
int len = sizeof(clientaddr);
ZeroMemory(&clientaddr,sizeof(clientaddr));
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用accept()接受客户连接请求并创建从套接字*/
ssock = accept(msock, (struct sockaddr *)&clientaddr, &len);
if(ssock == INVALID_SOCKET){       //调用accept()函数出错
    printf("Accept Failed!\n");
    exit(-1);
}
```

⑥ 步骤6的C语言实现方法

```
#define BUFSIZE 4096                    //定义缓冲区大小为4M
char buf[BUFSIZE];                     //声明数据缓冲区变量
ZeroMemory(buf,sizeof(buf));
int num;
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用recv()函数利用从套接字接收客户端发来的数据*/
num=recv(ssock,buf, sizeof(buf), 0);
if (num<0){
    printf("Recieve Data Failed!\n");
    exit(-1);
}
```

⑦ 步骤7的C语言实现方法

```
char *buffer = "HELLO!";                    //声明数据缓冲区变量
num=0;
/*以下语句用于调用send()函数利用从套接字发送数据给客户端*/
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
num= send(ssock,buffer,strlen(buffer),0);
if(num!=strlen(buffer)){                    //调用send()函数出错
    printf("Send Data Failed!\n");
    exit(-1);
}
```

⑧ 步骤8的C语言实现方法

```
closesocket(ssock);
```

⑨ 步骤9的C语言实现方法

```
closesocket(msock);
WSACleanup();
```

第一章：基于套接字的TCP/IP网络通信原理与模型

2. TCP客户端算法的实现流程

（1）TCP客户端算法的步骤描述

步骤1：使用WSAStartup()函数初始化Winsock DLL；

步骤2：调用socket()函数创建客户端TCP套接字；

步骤3：找到期望与之通信的远程服务器端套接字的端点地址（即远程服务器端的IP地址和协议端口号）；然后，再调用connect()函数向远程服务器端发起TCP连接建立请求；

步骤4：在与服务器端成功地建立了TCP连接之后，调用send()函数将缓冲区中的数据从该TCP套接字发送给该远程服务器端；

第一章：基于套接字的TCP/IP网络通信原理与模型

步骤5：调用recv()函数从该TCP套接字读取服务器端发送过来的数据并存入缓冲区中；

步骤6：与服务器端交互完毕，调用closesocket()函数将该TCP套接字关闭并释放所占用的系统资源。

步骤7：最后，调用WSACleanup()函数结束Winsock Socket API。

（2）TCP客户端算法的C语言实现方法

第一章：基于套接字的TCP/IP网络通信原理与模型

① 步骤1的C语言实现方法

```
int ret;
WORD sockVersion = MAKEWORD(2,2);
WSADATA wsaData;
ret=WSAStartup(sockVersion, &wsaData);
if (ret != 0){
    printf("Couldn't Find a Useable Winsock.dll!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

② 步骤2的C语言实现方法

```
SOCKET tsock;                          //声明套接字描述符变量
tsock = socket(AF_INET,SOCK_STREAM,0);  //调用socket()函数创建套接字
if (tsock == INVALID_SOCKET){          //调用socket()函数出错
    printf("Create Socket Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

③ 步骤3的C语言实现方法

```
#define SERVERIP "172.0.0.1"          //定义IP地址常量
#define SERVERPORT 10000              //定义端口号常量
struct sockaddr_in servaddr;          //声明端点地址结构体变量
ZeroMemory(&servaddr,sizeof(servaddr));
/*以下3条语句用于给端点地址结构体变量servaddr赋值*/
servaddr.sin_family=AF_INET;          //给协议族字段赋值
inet_aton(SERVERIP,&servaddr.sin_addr);   //给IP地址字段赋值
servaddr.sin_port = htons(SERVERPORT);    //给端口号字段赋值
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
/*以下语句用于调用connect()函数向远程服务器发起TCP连接建立请求*/
int ret;
ret=connect(tsock,(struct sockaddr *)&servaddr,sizeof(struct sockaddr));
if(ret<0){                              //调用connect()函数出错
    printf("Connect Failed!\n");
    exit(-1);
}
```

第一章：基于套接字的TCP/IP网络通信原理与模型

④ 步骤4的C语言实现方法

```
char *buffer = "HELLO!";                    //声明数据缓冲区变量
int num=0;
/*以下语句用于调用send()利用从套接字发送数据给服务器端*/
num= send(tsock,buffer,strlen(buffer),0);
if(num!=strlen(buffer)){                    //调用send()函数出错
    printf("Send Data Failed!\n");
    exit(-1);
}
```

⑤ 步骤5的C语言实现方法

```
#define BUFSIZE 4096                        //定义缓冲区大小为4M
char buf[BUFSIZE];                          //声明数据缓冲区变量
```

第一章：基于套接字的TCP/IP网络通信原理与模型

```
ZeroMemory(buf,sizeof(buf));
num=0;
/*以下语句用于调用recv()利用从套接字接收服务器端发来的数据*/
num=recv(tsock,buf, sizeof(buf), 0);
if (num<0){
      printf("Recieve Data Failed!\n");
      exit(-1);
}
```

⑥ 步骤6的C语言实现方法

```
closesocket(tsock);
```

⑦ 步骤7的C语言实现方法

```
WSACleanup();
```

第一章：基于套接字的TCP/IP网络通信原理与模型

1.6本章小结

本章主要对TCP/IP参考模型及其通信原理、TCP/IP网络通信中的客户/服务器模型、以及TCP/IP网络通信中的客户端和服务器端算法的设计流程等内容分别进行了详细介绍。通过本章的学习，需要了解基于套接字的TCP/IP网络通信原理；需要熟习基于套接字的TCP/IP网络通信模型与原理；需要掌握TCP/IP网络通信中的TCP服务器端与客户端算法的设计流程与C语言实现方法、以及UDP服务器端与客户端算法的设计流程与C语言实现方法。

本章作业

第1-10题

谢谢！