互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第1章 简介

1.1 概述

要编写通过计算机网络通信的程序，首先要确定这些程序相互通信所用的协议（protocol）。在深入设计一个协议的细节之前，应该从高层次决断通信由哪个程序发起以及响应在何时产生。举例来说，一般认为Web服务器程序是一个长时间运行的程序（即所谓的*守护程序*，daemon），它只在响应来自网络的请求时才发送网络消息。协议的另一端是Web客户程序，如某种浏览器，与服务器进程的通信总是由客户进程发起。大多数网络应用就是按照划分成*客户*（client）和*服务器*（server）[①]来组织的。在设计网络应用[②]时，确定总是由客户发起请求往往能够简化协议和程序[③]本身。当然一些较为复杂的网络应用还需要*异步回调*（asynchronous callback）通信，也就是由服务器向客户发起请求消息。然而坚持采纳图1-1所示的基本客户/服务器模型的网络应用毕竟要普遍得多。

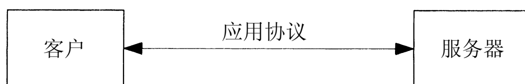

图1-1　网络应用：客户和服务器

通常客户每次只与一个服务器通信，不过以使用Web浏览器为例，我们也许在10分钟内就可以与许多不同的Web服务器通信。从服务器的角度来看，一个服务器同时与多个客户通信并不稀奇，见图1-2。本书后面将介绍若干种让一个服务器同时处理多个客户请求的方法。

3

可认为客户与服务器之间是通过某个网络协议通信的，但实际上，这样的通信通常涉及多个网络协议层。本书的焦点是TCP/IP协议族，也称为网际协议族。举例来说，Web客户与服务

① 本书英文原文通篇频繁使用client（客户）和server（服务器）这两个术语。实际上它们的具体含义随上下文而变化，有时指静态的源程序或可执行程序（客户程序和服务器程序），有时指动态进程（客户进程和服务器进程），有时指运行进程的主机（客户主机和服务器主机）。在不致引起混淆的前提下，我们简单地称客户进程为客户，称服务器进程为服务器。——译者注

② 应用（application）这个术语的具体含义随上下文而变化，有时指程序（应用程序），有时指进程（应用进程），有时作为名词性修饰词译为应用。本书有时把同处应用层的客户和服务器对也用应用表示，我们称之为应用系统、网络应用或应用。——译者注

③ Unix系统中程序（program）和进程（process）是在系统调用exec上衔接的。exec既可以由shell隐式调用（直接输入命令行执行程序属于这种情况），也可以在用户程序中显式调用。显式exec调用执行的程序在本书中称为新程序，以示与exec调用所在程序的区别。exec调用前后两个程序实际上在同一个进程环境下执行，不过往往使用新程序的名字来称呼这个进程。exec调用往往跟在某个fork调用之后，这样新程序将在新的进程环境中执行。客户程序和迭代服务器程序运行时通常只有一个进程，并发服务器程序运行时除主进程外，通常还为每个客户派生一个进程。程序和进程的密切关系使得两者有时相互渗透使用，不易区分。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

器之间使用TCP（Transmission Control Protocol，传输控制协议）通信。TCP又转而使用IP（Internet Protocol，网际协议）通信，IP再通过某种形式的数据链路层通信。如果客户与服务器处于同一个以太网，就有图1-3所示的通信层次。

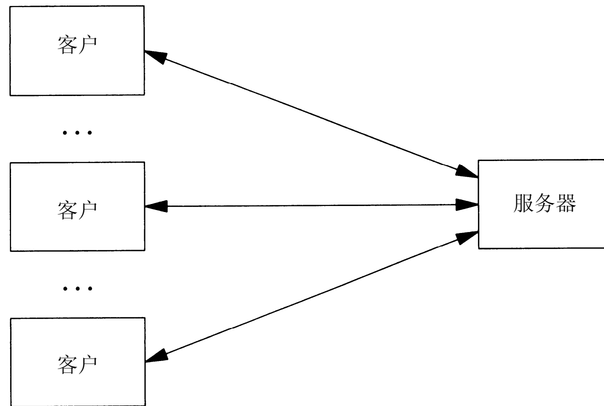

图1-2 一个服务器同时处理多个客户的请求

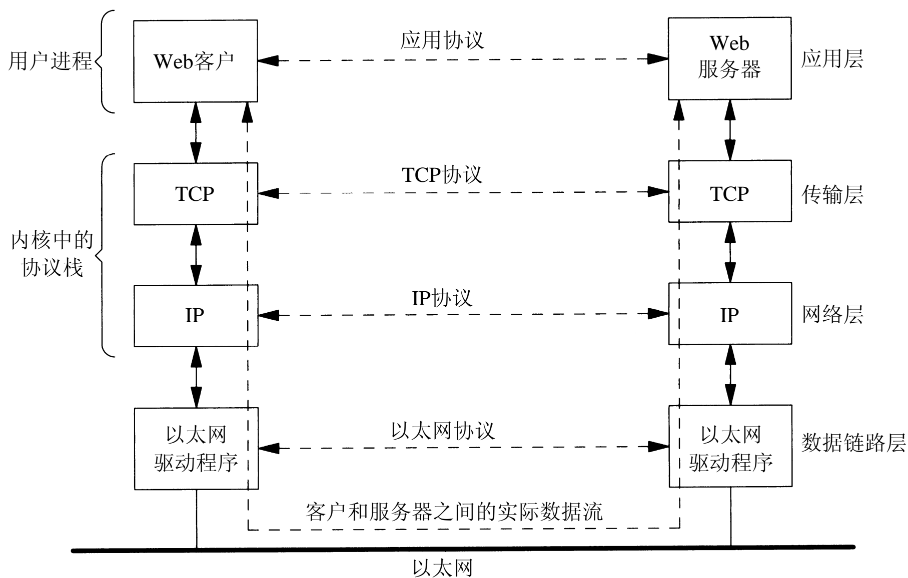

图1-3 客户与服务器使用TCP在同一个以太网中通信

尽管客户与服务器之间使用某个应用协议通信，传输层却使用TCP通信。注意，客户与服务器之间的信息流在其中一端是向下通过协议栈的，跨越网络后，在另一端则是向上通过协议栈的。另外注意，客户和服务器通常是用户进程，而TCP和IP协议通常是内核中协议栈的一部分。我们在图1-3右边标出了4个层。 4

本书讨论的协议不限于TCP和IP。有些客户和服务器改用UDP（User Datagram Protocol，用户数据报协议）而不是TCP，第2章将详细介绍这两个协议。此外，本书使用术语“IP”来称谓的那个协议，自20世纪80年代早期以来一直在使用，其实其正式名称是IPv4（IP version 4，IP

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

版本4）。IPv4的一个新版本IPv6（IP version 6，IP版本6）是在20世纪90年代中期开发出来的，将来会取代IPv4。本书既讨论使用IPv4的网络应用程序的开发，也讨论使用IPv6的网络应用程序的开发。附录A会给出IPv4和IPv6的一个比较，同时介绍正文中将讨论的其他协议。

同一网络应用的客户和服务器无需如图1-3所示处于同一个局域网（local area network，LAN）。例如，图1-4展示了处于不同局域网中的客户和服务器，而这两个局域网是使用路由器（router）连接到广域网（wide area network，WAN）的。

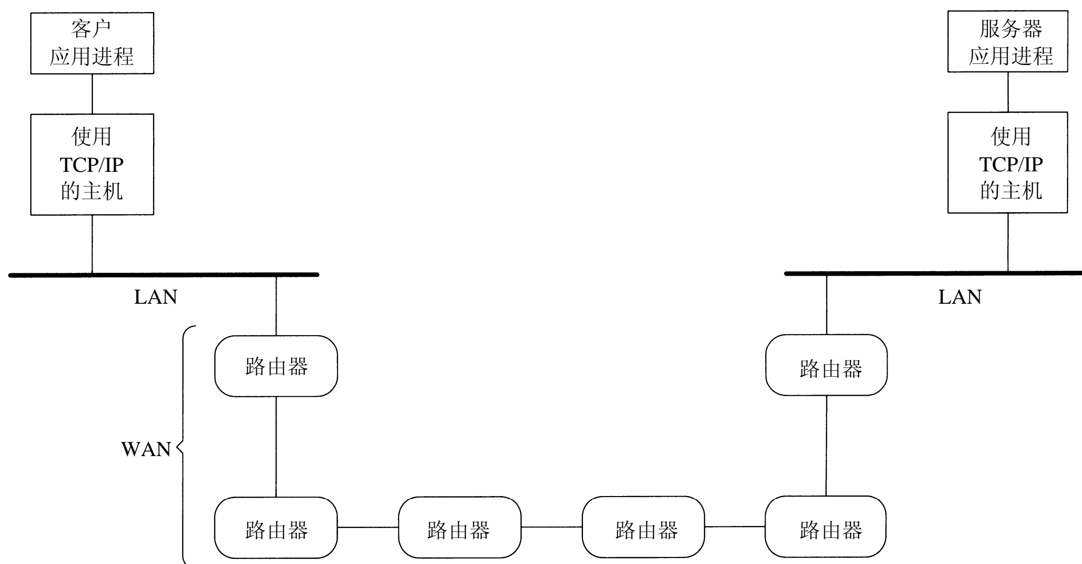

图1-4　处于不同局域网的客户主机和服务器主机通过广域网连接

路由器是广域网的架构设备。当今最大的广域网是因特网[①]（Internet）。许多公司也构建自己的广域网，而这些私用的广域网既可以连接到因特网，也可以不连接到因特网。

本章其余部分将概述多个主题，这些主题在后续章节中还会具体介绍。我们从一个尽管简单却完整的TCP客户程序开始，它展示了全书都会遇到的许多函数调用和概念。这个客户程序只能在IPv4上运行，不过我们会给出让它在IPv6上运行所需进行的修改。更好的办法是编写独立于协议的客户和服务器程序，这在第11章中会讨论。本章同时展示一个与该TCP客户程序配合工作的完整的TCP服务器程序。

① internet一词有多种含义。一是网际网（internet），采用TCP/IP协议族通信的任何网络都是网际网，因特网就是一个网际网。二是因特网（Internet），它是一个专用名词，特指从ARPANET发展而来的连接全球各个ISP的大型网际网。三是作为名词性修饰词，这时应根据情况分别译成“因特网”、“网际网”或“网际”。例如，Internet Protocol译成“网际协议”（注意：“Internet Protocol”是“internet protocol”一词名词专用化的结果）；Internet Society则译成“因特网学会”。应注意区分因特网和网际网这两个概念：因特网只有一个，为了确保其中任何一个节点（主机或路由器）都能寻址到，其寻址规则和地址分配方案是全球统一的；不属于因特网的网际网却可以为其中的节点任意分配地址，譬如说把因特网中的多播地址（224.0.0.0/4）分配用于单播目的也没有问题，因为地址属性（单播、多播、广播、回馈、私用等）是额外配置到TCP/IP协议族上的，并非TCP/IP协议族的本质特征，尽管实际上TCP/IP的各个实现几乎一律采用因特网的寻址规则。虽然国内权威机构已经为“Internet”一词正过中文名（因特网），许多文献仍然沿用“互联网”这个不确切的名称。互联网的说法是相对内联网（intranet）而言的，后者特指使用因特网私用地址寻址各个节点的网际网，因而只是比较特殊的网际网。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

5

为了简化代码，我们对本书中要调用的大多数系统函数定义了各自的包裹函数。多数情况下我们可以使用这些包裹函数来检查错误，输出适当的消息，以及在出错时终止程序的运行。我们还给出了本书中大多数例子所用的测试网络、主机、路由器以及它们的主机名、IP地址和操作系统。

如今讨论Unix时经常使用POSIX一词，它是一种被多数厂商采纳的标准。我们将介绍POSIX的历史以及它对本书所讲述的API的影响，并介绍该领域的其他主要标准。

1.2 一个简单的时间获取客户程序

让我们考虑一个具体的例子，引入将在本书中遇到的许多概念和说法。图1-5所示的是TCP当前时间查询客户程序的一个实现。该客户与其服务器建立一个TCP连接后，服务器以直观可读格式简单地送回当前时间和日期。

intro/daytimetcpcli.c

```
 1 #include    "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd, n;
 6     char    recvline[MAXLINE + 1];
 7     struct sockaddr_in  servaddr;

 8     if (argc != 2)
 9         err_quit("usage: a.out <IPaddress>");

10     if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
11         err_sys("socket error");

12     bzero(&servaddr, sizeof(servaddr));
13     servaddr.sin_family = AF_INET;
14     servaddr.sin_port   = htons(13);   /* daytime server */
15     if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
16         err_quit("inet_pton error for %s", argv[1]);

17     if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
18         err_sys("connect error");

19     while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
20         recvline[n] = 0;        /* null terminate */
21         if (fputs(recvline, stdout) == EOF)
22             err_sys("fputs error");
23     }
24     if (n < 0)
25         err_sys("read error");

26     exit(0);
27 }
```

intro/daytimetcpcli.c

图1-5 TCP时间获取客户程序

6

这就是本书用于展示所有源代码的格式。每个非空行都被编排行号。如稍后所示，代码

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

正文讲解部分一开始标注该段代码起始与结束的行号。有的段落会以一个简短的、描述性的醒目标题起头，对所讲解代码段进行概要说明。

每个源代码段起始与结束处的水平线标出了该代码段所在的源代码文件名，对于本例就是intro目录下的daytimetcpcli.c文件（intro/daytimetcpcli.c）。本书所有例子的源代码都可免费获得（见前言），在此标注它们的文件名便于读者找到其源文件。在阅读本书期间，编译、运行特别是修改这些程序是学习网络编程概念的好方法。

整本书中我们随时会插入缩进的小字号段落（如此处所示）来说明实现的细节和历史上的观点。

如果编译该程序生成默认的a.out可执行文件后执行它，我们会得到如下结果：

```
solaris % a.out 206.168.112.96                我们的输入
Mon May 26 20:58:40 2003                      程序的输出
```

当我们展示交互的输入和输出时，输入总是采用加粗的等宽字体，而计算机的输出总是采用不加粗的等宽字体。注释用宋体字加在右边。作为shell提示一部分的系统名字（本例中为solaris）指明在哪个主机上执行该命令。图1-16展示了用于运行本书中大多数例子的各个系统，它们的主机名本身通常就说明了各自的操作系统。

在这个短短27行的程序中有许多细节值得考虑。这里我们简短地提一下，目的是让初次遇到网络程序的读者有所准备，本书后面会更详细地说明这些内容。

包含头文件

1 包含我们自己编写的名为unp.h的头文件，见D.1节。该头文件包含了大部分网络程序都需要的许多系统头文件，并定义了所用到的各种常值[①]（如MAXLINE）。

命令行参数

2~3 这是main函数的定义，其形式参数就是命令行参数。本书中的代码假设使用ANSI C编译器（也称为ISO C编译器）编写。

创建TCP套接字

10~11 socket函数创建一个网际（AF_INET）字节流（SOCK_STREAM）套接字，它是TCP套接字的花哨名字。该函数返回一个小整数描述符，以后的所有函数调用（如随后的connect和read）就用该描述符来标识这个套接字。

7

if语句包含3个操作：调用socket函数，把返回值赋给变量sockfd，再测试所赋的这个值是否小于0。虽然我们可以把该语句分割成两条C语句：

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
```

但是把这两行合并成一行却是常见的C语言习惯用法。按照C语言的优先规则（小于运算符的优先级高于赋值运算符），函数调用和赋值语句外边的那对括号是必需的。作为一种编码风格，作者总是在这样的两个左括号间加一个空格，提示比较运算的左侧同时也是一个赋值运算。（这种风格借鉴自Minix源代码［Tenenbaum 1987］。）该程序稍后的while语句也使用相同的样式。

---

① 严格地说，C语言中用#define伪命令定义的对象称为常数，用const限定词定义并初始化的对象称为常量（相对于变量而言）。常数的值在编译时确定，常量的值则在运行时初始化后确定（不过此后只能作为右值使用）。本书绝大多数恒定值是用#define定义的常数。不过“常数”这一称谓容易让人狭义地理解成仅仅是数而已，因此本书统一使用“常值”指代其值恒定不变的对象。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

后面我们将遇到术语套接字（socket[①]）的许多不同用法。首先，我们正在使用的API称为套接字API（sockets API）。上一段中名为socket的函数就是套接字API的一部分。上一段中我们还提到了“TCP套接字”，它是“TCP端点”（TCP endpoint）的同义词。

如果socket函数调用失败，我们就调用自己的err_sys函数放弃程序运行。err_sys函数输出我们作为参数提供的出错消息以及所发生的系统错误的描述（例如出自socket函数的可能错误之一“Protocol not supported”（协议不受支持）），然后终止进程。这个函数和以err_开头的其他若干个函数都是我们自行编写的，它们的调用将贯穿全书，D.3节会描述这些函数。

指定服务器的IP地址和端口

12~16 我们把服务器的IP地址和端口号填入一个网际套接字地址结构（一个名为servaddr的sockaddr_in结构变量）。使用bzero把整个结构清零后，置地址族为AF_INET，端口号为13（这是时间获取服务器的众所周知端口，支持该服务的任何TCP/IP主机都使用这个端口号，见图2-18），IP地址为第一个命令行参数的值（argv[1]）。网际套接字地址结构中IP地址和端口号这两个成员必须使用特定格式，为此我们调用库函数htons（“主机到网络短整数”）去转换二进制端口号，又调用库函数inet_pton（“呈现形式到数值”）去把ASCII命令行参数（例如运行本例子所用的206.168.112.96）转换为合适的格式。

> bzero不是一个ANSI C函数。它起源于早期的Berkeley网络编程代码。不过我们在整本书中使用它而不用ANSI C的memset函数，因为bzero（带2个参数）比memset（带3个参数）更好记忆。几乎所有支持套接字API的厂商都提供bzero，如果没有，那么可以使用unp.h头文件中提供的该函数的宏定义。
>
> 事实上，在TCPv3一书首次印刷时，作者在10处出现memset函数的地方犯了错，互换了第二和第三个参数。C编译器发现不了这个错误，因为这两个参数的类型是相同的。（其实第二个参数是int类型，第三个参数是size_t，通常定义为unsigned int类型，然而分别指定给这两个参数的值为0和16，它们对于两个参数的类型同样可以接受。）对memset的这些调用仍然正常，不过没做任何事，因为待初始化的字节数被指定成了0。程序之所以仍然工作是因为只有少数套接字函数要求网际套接字地址结构的最后8个字节置0。无论如何，这确实是一个错误，且是一个通过使用bzero函数可以避免的错误，因为如果使用函数原型，C编译器总能发现bzero的两个参数被互换的错误。
>
> 此处也许是你第一次遇到inet_pton函数。它是一个支持IPv6（详见附录A）的新函数。以前的代码使用inet_addr函数来把ASCII点分十进制数串变换成正确的格式，不过它有不少局限，而这些局限在inet_pton中都得以纠正。如果你的系统尚未支持该函数，那你可以使用我们在3.7节中提供的它的一个实现。

8

建立与服务器的连接

17~18 connect函数应用于一个TCP套接字时，将与由它的第二个参数指向的套接字地址结构

① socket一词译者认为译成“套接口”更为准确，其理由如下。首先，作为网络编程API之一的套接口（sockets，注意这种用法总是采用复数形式，如sockets API、sockets library等）跟XTI一样，是应用层到传输层或其他协议层的访问接口。其次，具体使用的套接口是与Unix管道的某一端类似的东西，我们既可以往这个“口”写数据，也可以从这个“口”读数据。最后，套接口函数使用套接口描述字（discriptor）访问具体的套接口，如果把套接口描述字的简称sockfd译成“套接字”倒比较合适。从这个意义上看，一个套接口可对应多个套接字，因为Unix的描述字既可以复制，也可以继承；反过来，一个套接字对应且只对应一个套接口。但是，鉴于现在socket广泛被接受的译法是“套接字”，所以本书亦采用了“套接字”的译法。相应地，descriptor也采用了“描述符”的译法，而未坚持译为“描述字”。——编者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

指定的服务器建立一个TCP连接。该套接字地址结构的长度也必须作为该函数的第三个参数指定，对于网际套接字地址结构，我们总是使用C语言的sizeof操作符由编译器来计算这个长度。

在头文件unp.h中，我们使用#define把SA定义为struct sockaddr，也就是通用套接字地址结构。每当一个套接字函数需要一个指向某个套接字地址结构的指针时，这个指针必须强制类型转换成一个指向通用套接字地址结构的指针。这是因为套接字函数早于ANSI C标准，20世纪80年代早期开发这些函数时，ANSI C的void *指针类型还不可用。问题是“struct sockaddr”长达15个字符，往往造成源代码行超出屏幕（或者书页，若是排印在书上）的右边缘，因此我们把它缩减成SA。我们将在解释图3-3时详细讨论通用套接字地址结构。

读入并输出服务器的应答

19~25 我们使用read函数读取服务器的应答，并用标准的I/O函数fputs输出结果。[①]使用TCP时必须小心，因为TCP是一个没有记录边界的字节流协议。服务器的应答通常是如下格式的26字节字符串：

```
Mon May 26 20:58:40 2003\r\n
```

其中，\r是ASCII回车符，\n是ASCII换行符。使用字节流协议的情况下，这26个字节可以有多种返回方式：既可以是包含所有26个字节的单个TCP分节[②]，也可以是每个分

---

① 为求简洁明确，本书以后尽量采用直接把函数名或C语言关键词用作动词的译法。例如，本句的这种译法是“我们read服务器的应答，并fputs结果。”；又如：“如果connect成功，那就break出循环。”的意思是：“如果connect函数调用成功（表示连接成功），那就执行C语言的break语句跳出循环。”

② 计算机网络各层对等实体间交换的单位信息称为*协议数据单元*（protocol data unit，PDU），分节（segment）就是对应于TCP传输层的PDU。按照协议与服务之间的关系，除了最低层（物理层）外，每层的PDU通过由紧邻下层提供给本层的服务接口，作为下层的*服务数据单元*（service data unit，SDU）传递给下层，并由下层间接完成本层的PDU交换。如果本层的PDU大小超过紧邻下层的最大SDU限制，那么本层还要事先把PDU划分成若干个合适的片段让下层分开载送，再在相反方向把这些片段重组成PDU。同一层内SDU作为PDU的*净荷*（payload）字段出现，因此可以说上层PDU由本层PDU（通过其SDU字段）承载。每层的PDU除用于承载紧邻上层的PDU（即承载数据）外，也用于承载本层协议内部通信所需的控制信息。由于本书涉及PDU种类较多，为避免混淆，我们在本章末汇总简要说明。

应用层实体（如客户或服务器进程）间交换的PDU称为*应用数据*（application data），其中在TCP应用进程之间交换的是没有长度限制的单个双向字节流，在UDP应用进程之间交换的是其长度不超过UDP发送缓冲区大小的单个*记录*（record），在SCTP应用进程之间交换的是没有总长度限制的单个或多个双向记录流。传输层实体（例如对应某个端口的传输层协议代码的一次运行）间交换的PDU称为*消息*（message），其中TCP的PDU特称为*分节*（segment）。消息或分节的长度是有限的。在TCP传输层中，发送端TCP把来自应用进程的字节流数据（即由应用进程通过一次次输出操作写出到发送端TCP套接字中的数据）按顺序经分割后封装在各个分节中传送给接收端TCP，其中每个分节所封装的数据既可能是发送端应用进程单次输出操作的结果，也可能是连续数次输出操作的结果，而且每个分节所封装的单次输出操作的结果或者首尾两次输出操作的结果既可能是完整的，也可能是不完整的，具体取决于可在连接建立阶段由对端通告的最大分节大小（maximum segment size，MSS）以及外出接口的最大传输单元（maximum transmission unit，MTU）或外出路径的路径MTU（如果网络层具有路径MTU发现功能，如IPv6）。分节除了用于承载应用数据外，也用于建立连接（SYN分节）、终止连接（FIN分节）、中止连接（RST分节）、确认数据接收（ACK分节）、刷送待发数据（PSH分节）和携带紧急数据指针（URG分节），而且这些功能（包括承载数据）可以灵活组合。UDP传输层相当简单，发送端UDP就把来自应用进程的单个记录整个封装在UDP消息中传送给接收端UDP。SCTP引入了称为*块*（chunk）的数据单元，SCTP消息就由一个公共首部加上一个或多个块构成：公共首部类似UDP消息的首部，仅仅给出源目的端口号和整个SCTP消息的校验和；块则既可以承载数据（称为DATA块），也可以承载控制信息（计有SACK块、INIT块、INIT ACK块、COOKIE ECHO块、COOKIE ACK块、SHUTDOWN块、SHUTDOWN ACK块、SHUTDOWN COMPLETE块、ABORT块、ERROR块、HEARTBEAT块和HEARTBEAT ACK块，总称为控制块）。发送端SCTP把来自应用进程的（一个或多个）记录流数据按照流内

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

节只含1个字节的26个TCP分节，还可以是总共26个字节的任何其他组合。通常服务器返回包含所有26个字节的单个分节，但是如果数据量很大，我们就不能确保一次read调用能返回服务器的整个应答。因此从TCP套接字读取数据时，我们总是需要把read编写在某个循环中，当read返回0（表明对端关闭连接）或负值（表明发生错误）时终止循环。

本例中，服务器关闭连接表征记录的结束。HTTP（Hypertext Transfer Protocol，超文本传送协议）的1.0版本也采用这种技术。还可以用其他技术标记记录结束。例如，SMTP（Simple Mail Transfer Protocol，简单邮件传送协议）使用由ASCII回车符后跟换行符构成的2字节序列标记记录的结束；Sun远程过程调用（Remote Procedure Call，RPC）以及域名系统（Domain Name System，DNS）在使用TCP承载应用数据时，在每个要发送的记录之前放置一个二进制的计数值，给出这个记录的长度。这里的重要概念是TCP本身并不提供记录结束标志：如果应用程序需要确定记录的边界，它就要自己去实现，已有一些常用的方法可供选择。

终止程序

26 exit终止程序运行。Unix在一个进程终止时总是关闭该进程所有打开的描述符，我们的TCP套接字就此被关闭。

刚才已提过，本书后面会对刚才讲述的所有概念深入进行探讨。

9

1.3 协议无关性

图1-5中的程序是与IPv4协议相关的：我们分配并初始化一个sockaddr_in类型的结构，把该结构的协议族成员设置为AF_INET，并指定socket函数的第一个参数为AF_INET。

顺序和记录边界封装在各个DATA块中，并在DATA块首部记上各自的流ID。一个记录通常对应一个DATA块；对于过长的记录，发送端SCTP既可以像UDP那样拒绝发送，也可以把它们拆分到多个DATA块中以便发送，接收端SCTP收取后把它们组合成单个记录上传。作为传输层PDU的SCTP消息既可以只包含单个块（DATA块或控制块），也可以在接口MTU或路径MTU的限制下包含多个块（称为块的捆绑，控制块在前，DATA块在后），不过INIT块、INIT ACK块和SHUTDOWN COMPLETE块不能跟任何其他块捆绑。SCTP收发两端均独立处理捆绑在同一个消息中的各个块，鉴于此，我们可以直接把块作为传输层PDU看待，本书也往往这么使用。

网络层实体间交换的PDU称为IP数据报（IP datagram），其长度有限：IPv4数据报最大65 535字节，IPv6数据报最大65 575字节。发送端IP把来自传输层的消息（或TCP分节）整个封装在IP数据报中传送。链路层实体间交换的PDU称为帧（frame），其长度取决于具体的接口。IP数据报由IP首部和所承载的传输层数据（即网络层的SDU）构成。过长的IP数据报无法封装在单个帧中，需要先对其SDU进行分片（fragmentation），再把分成的各个片段（fragment）冠以新的IP首部封装到多个帧中。在一个IP数据报从源端到目的端的传送过程中，分片操作既可能发生在源端，也可能发生在途中，而其逆操作即重组（reassembly）一般只发生在目的端；SCTP为了传送过长的记录采取了类似的分片和重组措施。TCP/IP协议族为提高效率会尽可能避免IP的分片/重组操作：TCP根据MSS和MTU限定每个分节的大小以及SCTP根据MTU分片/重组过长记录都是这个目的（SCTP的块捆绑则是为了在避免IP分片/重组操作的前提下提高块传输效率）；另外，IPv6禁止在途中的分片操作（基于其路径MTU发现功能），IPv4也尽量避免这种操作。不论是否分片，都由IP作为链路层的SDU传入链路层，并由链路层封装在帧中的数据称为分组（packet，俗称包）。可见一个分组既可能是一个完整的IP数据报，也可能是某个IP数据报的SDU的一个片段被冠以新的IP首部后的结果。另外，本书中讨论的MSS是应用层（TCP）与传输层之间的接口属性，MTU则是网络层和链路层之间的接口属性。

上述讨论参见RFC 1122、RFC 793、RFC 768、RFC 3286、RFC 2960和本书2.11节、7.9节。另外需注意的是，SCTP目前只是处于提案标准（proposed standard）阶段，尚未进入能够被多数厂商采纳并实现的草案标准（draft standard）阶段，更没有像TCP和UDP那样历经考验而成为因特网标准（分配STD号）。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

为了让图1-5中的程序能够在IPv6上运行，我们必须修改这段代码。图1-6所示的是一个能够在IPv6上运行的版本，其中改动之处用加粗的等宽字体突出显示。

intro/daytimetcpcliv6.c

```
 1 #include    "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd, n;
 6     char    recvline[MAXLINE + 1];
 7     struct sockaddr_in6 servaddr;

 8     if (argc != 2)
 9         err_quit("usage: a.out <IPaddress>");

10     if ( (sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
11         err_sys("socket error");

12     bzero(&servaddr, sizeof(servaddr));
13     servaddr.sin6_family = AF_INET6;
14     servaddr.sin6_port   = htons(13);  /* daytime server */
15     if (inet_pton(AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
16         err_quit("inet_pton error for %s", argv[1]);

17     if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
18         err_sys("connect error");

19     while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
20         recvline[n] = 0;    /* null terminate */
21         if (fputs(recvline, stdout) == EOF)
22             err_sys("fputs error");
23     }
24     if (n < 0)
25         err_sys("read error");

26     exit(0);
27 }
```

intro/daytimetcpcliv6.c

图1-6　适合于IPv6的图1-5所示程序的修改版

我们只修改了程序的5行代码，得到的却是另一个与协议相关的程序：这回是与IPv6协议相关的。更好的做法是编写协议无关的程序。图11-11将给出本客户程序的协议无关版本，它使用了getaddrinfo函数（由tcp_connect函数调用）。

10

这两个程序的另一个不足之处是：用户必须以点分十进制数格式给出服务器的IP地址（如适合于IPv4版本的206.168.112.219）。人们更习惯于用名字（如www.unpbook.com）来代替数字。我们将在第11章中讨论主机名与IP地址之间以及服务名与端口之间的转换函数。我们特意推迟讨论这些函数，在第11章之前继续使用IP地址和端口号，目的是了解我们必须填写和查看的套接字地址结构的细节，避免被另一个函数集的细节把网络编程的讨论搞复杂了。

1.4　错误处理：包裹函数

任何现实世界的程序都必须检查每个函数调用是否返回错误。在图1-5所示的程序中，我们

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

检查socket、inet_pton、connect、read和fputs函数是否返回错误，当发生错误时，就调用我们自己的err_quit或err_sys函数输出一个出错消息并终止程序的运行。我们发现绝大多数情况下这正是我们想做的事。个别情况下，当这些函数返回错误时，我们想做的事并非简单地终止程序的运行，如图5-12所示，我们必须检查系统调用是否被中断了。

既然发生错误时终止程序的运行是普遍的情况，我们可以通过定义包裹函数（wrapper function）来缩短程序。每个包裹函数完成实际的函数调用，检查返回值，并在发生错误时终止进程。我们约定包裹函数名是实际函数名的首字母大写形式。例如，在语句

```
sockfd = Socket(AF_INET, SOCK_STREAM, 0);
```

中，函数Socket是函数socket的包裹函数，如图1-7所示。

lib/wrapsock.c

```
236 int
237 Socket(int family, int type, int protocol)
238 {
239     int     n;

240     if ( (n = socket(family, type, protocol)) < 0)
241         err_sys("socket error");
242     return(n);
243 }
```

lib/wrapsock.c

图1-7 socket函数的包裹函数

在本书中只要你遇到一个首字母大写的函数名，它就是我们定义的某个包裹函数。它调用的实际函数的名字与包裹函数名相同，不过以对应的小写字母开头。

然而在讲解本书中提供的源代码时，我们总是指称被调用的最低级别的函数（如socket），而不是包裹函数（如Socket）。

11

这些包裹函数不见得多节省代码量，但当我们在第26章中讨论线程时，将会发现线程函数遇到错误时并不设置标准Unix的errno变量，而是把errno的值作为函数返回值返回调用者。这意味着每次调用以pthread_开头的某个函数时，我们必须分配一个变量来存放函数返回值，以便在调用err_sys前把errno变量设置成该值。为避免引入花括号把代码弄得很混乱，我们可以使用C语言的逗号操作符，把errno的赋值与err_sys的调用组合成一条语句，如下所示：

```
int    n;

if ( (n = pthread_mutex_lock(&ndone_mutex)) != 0)
    errno = n, err_sys("pthread_mutex_lock error");
```

我们也可以为此定义一个新的错误处理函数，它取系统的错误号作为一个参数，不过通过定义如图1-8所示的包裹函数，我们可以让以上这段代码更为易读：

```
Pthread_mutex_lock(&ndone_mutex);
```

要是仔细推敲C代码的编写，我们可以用宏来替代函数，从而稍微提高运行时效率，不过包裹函数很少是程序性能的瓶颈所在。

选择首字母大写一个函数名作为其包裹函数名是一种折中的方法。其他方法也考虑过，譬如给函数名加一个“e”前缀（如［Kernighan and Pike 1984］一书第182页所示），给函数名加一个“_e”后缀，等等。这些方法都能明显地提示调用了其他函数，但我们的这种风格看来是最少分散注意力的。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这种技术还有助于检查那些错误返回值通常被忽略的函数是否出错，例如close和listen。

lib/wrappthread.c
```
72 void
73 Pthread_mutex_lock(pthread_mutex_t *mptr)
74 {
75     int     n;

76     if ( (n = pthread_mutex_lock(mptr)) == 0)
77         return;
78     errno = n;
79     err_sys("pthread_mutex_lock error");
80 }
```
lib/wrappthread.c

图1-8　pthread_mutex_lock的包裹函数

本书后面的例子中，除非必须检查某个确定的错误是否发生，并以不同于终止进程的其他某种方式处理它，否则就使用这些包裹函数。书中不提供所有包裹函数的源代码，不过它们是可以免费获得的（见前言）。

12

Unix errno值

只要一个Unix函数（例如某个套接字函数）中有错误发生，全局变量errno就被置为一个指明该错误类型的正值，函数本身则通常返回–1。err_sys查看errno变量的值并输出相应的出错消息，例如当errno值等于ETIMEDOUT时，将输出“Connection timed out”（连接超时）。

errno的值只在函数发生错误时设置。如果函数不返回错误，errno的值就没有定义。errno的所有正数错误值都是常值，具有以“E”开头的全大写字母名字，并通常在<sys/errno.h>头文件中定义。值0不表示任何错误。

在全局变量中存放errno值对于共享所有全局变量的多个线程并不适合。我们将在第26章中讲述解决这一问题的方法。

全书中我们将使用诸如“connect函数返回ECONNREFUSED”这样的句子简明表达以下意思：该函数返回一个错误（通常函数返回值为–1），同时errno被置为指定的常值。

1.5　一个简单的时间获取服务器程序

我们可以编写一个简单的TCP时间获取服务器程序，它和1.2节中的客户程序一道工作。图1-9给出了这个服务器程序，它使用了上一节中讲过的包裹函数。

创建TCP套接字

10　TCP套接字的创建与客户程序相同。

把服务器的众所周知端口捆绑到套接字

11~15　通过填写一个网际套接字地址结构并调用bind函数，服务器的众所周知端口（对于时间获取服务是13）被捆绑到所创建的套接字。我们指定IP地址为INADDR_ANY，这样要是服务器主机有多个网络接口，服务器进程就可以在任意网络接口上接受客户连接。以后我们将了解怎样限定服务器进程只在单个网络接口上接受客户连接。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

intro/daytimetcpsrv.c

```
 1 #include     "unp.h"
 2 #include     <time.h>

 3 int
 4 main(int argc, char **argv)
 5 {
 6     int      listenfd, connfd;
 7     struct sockaddr_in  servaddr;
 8     char     buff[MAXLINE];
 9     time_t   ticks;

10     listenfd = Socket(AF_INET, SOCK_STREAM, 0);

11     bzero(&servaddr, sizeof(servaddr));
12     servaddr.sin_family = AF_INET;
13     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
14     servaddr.sin_port = htons(13);  /* daytime server */

15     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));

16     Listen(listenfd, LISTENQ);

17     for ( ; ; ) {
18         connfd = Accept(listenfd, (SA *) NULL, NULL);

19         ticks = time(NULL);
20         snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
21         Write(connfd, buff, strlen(buff));

22         Close(connfd);
23     }
24 }
```

intro/daytimetcpsrv.c

图1-9 TCP时间获取服务器程序

把套接字转换成监听套接字

16 调用listen函数把该套接字转换成一个监听套接字，这样来自客户的外来连接就可在该套接字上由内核接受。socket、bind和listen这3个调用步骤是任何TCP服务器准备所谓的监听描述符（listening descriptor，本例中为listenfd）的正常步骤。

常值LISTENQ在我们的unp.h头文件中定义。它指定系统内核允许在这个监听描述符上排队的最大客户连接数。我们将在4.5节详细说明客户连接的排队。

接受客户连接，发送应答

17~21 通常情况下，服务器进程在accept调用中被投入睡眠，等待某个客户连接的到达并被内核接受。TCP连接使用所谓的三路握手（three-way handshake）来建立连接。握手完毕时accept返回，其返回值是一个称为已连接描述符（connected descriptor）的新描述符（本例中为connfd）。该描述符用于与新近连接的那个客户通信。accept为每个连接到本服务器的客户返回一个新描述符。

本书全文采用的无限循环采用以下风格：

```
for ( ; ; ) {
    . . .
}
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

13~14

当前时间和日期是由库函数time返回的，它实际上返回的是自Unix纪元即1970年1月1日0点0分0秒（国际标准时间）以来的秒数。下一个库函数ctime把该整数值转换成直观可读的时间格式，例如：

```
Mon May 26 20:58:40 2003
```

snprintf函数在这个字符串末尾添加一个回车符和一个回行符，随后write函数把结果字符串写给客户。

> 如果你尚不习惯改用snprintf代替较早的sprintf函数，那么现在是学习的时候了。调用sprintf无法检查目的缓冲区是否溢出。相反，snprintf要求其第二个参数指定目的缓冲区的大小，因此可确保该缓冲区不溢出。
>
> snprintf相对较晚才加到ANSI C标准中，在称为ISO C99的版本中引入。不过几乎所有厂商都把它作为标准C函数库的一部分提供，而且另有许多免费可得的版本可用。我们贯穿全书使用snprintf，也推荐你出于可靠性考虑在自己的程序中改用它来代替sprintf。
>
> 值得注意的是，许多网络入侵是由黑客通过发送数据，导致服务器对sprintf的调用使其缓冲区溢出而发生的。必须小心使用的函数还有gets、strcat和strcpy，通常应分别改为调用fgets、strncat和strncpy。更好的替代函数是后来才引入的strlcat和strlcpy，它们确保结果是正确终止的字符串。编写安全的网络程序的更多技巧参见[Garfinkel, Schwartz, and Spafford 2003] 的第23章。

终止连接

22　服务器通过调用close关闭与客户的连接。该调用引发正常的TCP连接终止序列：每个方向上发送一个FIN，每个FIN又由各自的对端确认。2.6节将详细讲述TCP的三路握手和用于终止一个TCP连接的4个TCP分组。

与上节查看客户程序一样，本节查看服务器程序也非常简略，具体细节留待本书以后论述。有以下几点需要注意。

- 与其客户程序一样，这一服务器程序也与IPv4协议相关。我们将在图11-13中给出使用getaddrinfo函数实现的一个协议无关的版本。
- 本服务器一次只能处理一个客户。如果多个客户连接差不多同时到达，系统内核在某个最大数目的限制下把它们排入队列，然后每次返回一个给accept函数。本服务器只需调用time和ctime这两个库函数，运行速度很快。然而如果服务器需用较多时间（譬如说几秒钟或一分钟）服务每个客户，那么我们必须以某种方式重叠对各个客户的服务。图1-9中所示的服务器称为迭代服务器（iterative server），因为对于每个客户它都迭代执行一次。同时能处理多个客户的并发服务器（concurrent server）有多种编写技术。最简单的技术是调用Unix的fork函数（4.7节），为每个客户创建一个子进程。其他技术包括使用线程代替fork（26.4节），或在服务器启动时预先fork一定数量的子进程（30.6节）。 15
- 如果从shell命令行启动本例这样的一个服务器，我们也许想要它运行很长时间，因为服务器往往在系统工作期间一直运行。这要求我们往服务器程序中添加代码，以便它能够作为一个Unix守护进程（daemon）——能在后台运行且不跟任何终端关联的进程——运行。我们将在13.4节讨论守护进程。

1.6　本书中客户/服务器程序示例索引表

贯穿全书的用于阐述网络编程中使用的各种技术的两个客户/服务器程序示例如下：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- 时间获取客户/服务器程序（开始于图1-5、图1-6和图1-9）；
- 回射客户/服务器程序（开始于第5章）。

为了提供本书所涵盖不同主题的路线图，我们用下面4个表格汇总了将要开发的程序，并给出了它们的源代码所在的起始图号。图1-10列出了本书开发的时间获取客户程序的不同版本，其中有两个版本前面已讲过。图1-11列出了时间获取服务器程序的不同版本。图1-12列出了回射客户程序的不同版本，图1-13列出了回射服务器程序的不同版本。

图　号	说　明
1-5	TCP/IPv4，协议相关
1-6	TCP/IPv6，协议相关
11-4	TCP/IPv4，协议相关，调用gethostbyname和getservbyname
11-11	TCP，协议无关，调用getaddrinfo和tcp_connect
11-16	UDP，协议无关，调用getaddrinfo和udp_client
16-11	TCP，使用非阻塞connect
31-8	TCP，协议相关，用TPI取代套接字
E-1	TCP，协议相关，产生SIGPIPE
E-5	TCP，协议相关，输出套接字接收缓冲区的大小和MSS
E-11	TCP，协议相关，允许主机名（gethostbyname）或者IP地址
E-12	TCP，协议无关，允许主机名（gethostbyname）

图1-10 本书开发的时间获取客户程序的不同版本

16

图　号	说　明
1-9	TCP/IPv4，协议相关
11-13	TCP，协议无关，调用getaddrinfo和tcp_listen
11-14	TCP，协议无关，调用getaddrinfo和tcp_listen
11-19	UDP，协议无关，调用getaddrinfo和udp_server
13-5	TCP，协议无关，作为孤立的守护进程运行
13-12	TCP，协议无关，从inetd守护进程派生

图1-11 本书开发的时间获取服务器程序的不同版本

图　号	说　明
5-4	TCP/IPv4，协议相关
6-9	TCP，使用select
6-13	TCP，使用select并操纵缓冲区
8-7	UDP/IPv4，协议相关
8-9	UDP，验证服务器的地址
8-17	UDP，调用connect获取异步错误
14-2	UDP，使用SIGALRM信号在读服务器的应答时启动超时
14-4	UDP，使用select函数在读服务器的应答时启动超时
14-5	UDP，使用SO_RCVTIMEO套接字选项在读服务器的应答时启动超时
15-4	Unix域字节流，协议相关
15-6	Unix域数据报，协议相关

图1-12 本书开发的回射客户程序的不同版本

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

图　号	说　明
16-3	TCP，使用非阻塞I/O
16-10	TCP，使用两个进程（fork）
16-21	TCP，建立连接，然后发送RST
14-15	TCP，使用/dev/poll达成多路复用
14-18	TCP，使用kqueue达成多路复用
20-5	UDP，具有竞争状态的广播
20-6	UDP，具有竞争状态的广播
20-7	UDP，通过使用pselect消除了竞争状态的广播
20-9	UDP，通过使用sigsetjmp和siglongjmp消除了竞争状态的广播
20-10	UDP，通过在信号处理函数中使用IPC消除了竞争状态的广播
22-6	UDP，使用超时、重传和序列号实现可靠性
24-14	（第2版）UDP，使用带外数据对服务器心搏测试①
26-2	TCP，使用两个线程
27-6	TCP/IPv4，指定一条源路径
27-13	UDP/IPv6，指定一条源路径

图1-12　（续）

图　号	说　明
5-2	TCP/IPv4，协议相关
5-12	TCP/IPv4，协议相关，收拾终止了的子进程
6-21	TCP/IPv4，协议相关，使用select，单个进程处理所有客户
6-25	TCP/IPv4，协议相关，使用poll，单个进程处理所有客户
8-3	UDP/IPv4，协议相关
8-24	TCP和UDP/IPv4，协议相关，使用select
14-14	TCP，使用标准I/O函数库
15-3	Unix域字节流，协议相关
15-5	Unix域数据报，协议相关
15-15	Unix域字节流，带有从客户端传递凭证
22-4	UDP，接收目的地址和收取接口信息，截取数据报
22-15	UDP，捆绑所有接口地址
25-4	UDP，使用信号驱动的I/O
26-3	TCP，每个客户一个线程
26-4	TCP，每个客户一个线程，可移植的参数传递
27-6	TCP/IPv4，输出接收到的源路径
27-14	UDP/IPv6，输出并反转接收到的源路径
28-31	UDP，使用icmpd接收异步错误
E-15	UDP，捆绑所有接口地址

图1-13　本书开发的回射服务器程序的不同版本

1.7　OSI模型

描述一个网络中各个协议层的常用方法是使用国际标准化组织（International Organization

①　此处保留了本书第2版的内容。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

for Standardization，ISO）的计算机通信开放系统互连（open systems interconnection，OSI）模型。这是一个七层模型，如图1-14所示。图中同时给出了它与网际协议族的近似映射。

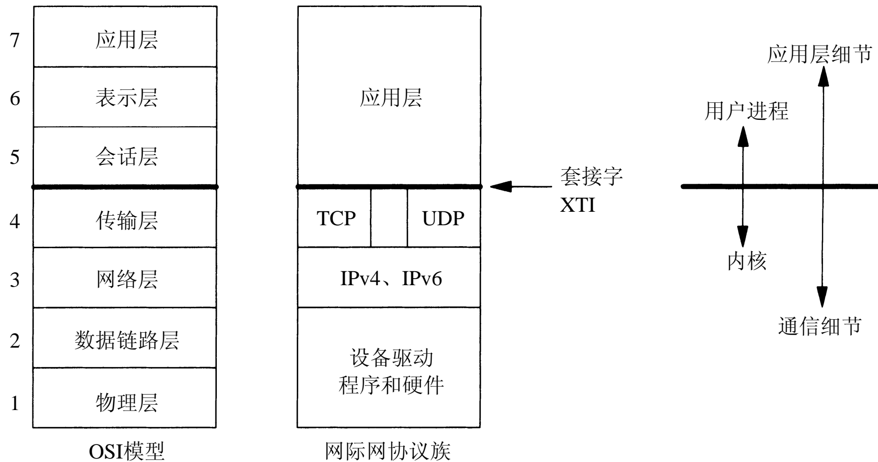

图1-14　OSI模型和网际协议族中的各层

我们认为OSI模型的底下两层是随系统提供的设备驱动程序和网络硬件。通常情况下，除需知道数据链路的某些特性外（如将在2.11节论述的1500字节以太网的MTU大小），我们不必关心这两层的具体情况。

网络层由IPv4和IPv6这两个协议处理，我们将在附录A中讲述它们。可以选择的传输层有TCP或UDP，我们将在第2章中讲述它们。图1-14中TCP与UDP之间留有间隙，表明网络应用绕过传输层直接使用IPv4或IPv6是可能的。这就是所谓的*原始套接字*（raw socket），我们将在第28章中讨论。

OSI模型的顶上三层被合并成一层，称为应用层。这就是Web客户（浏览器）、Telnet客户、Web服务器、FTP服务器和其他我们在使用的网络应用所在的层。对于网际协议，OSI模型的顶上三层协议几乎没有区别。

本书讲述的套接字编程接口是从顶上三层（网际协议的应用层）进入传输层的接口。本书的焦点是：如何使用套接字编写使用TCP或UDP的网络应用程序。我们已提到原始套接字，在第29章中我们将看到，甚至可以彻底绕过IP层直接读写数据链路层的帧。

为什么套接字提供的是从OSI模型的顶上三层进入传输层的接口？这样设计有两个理由，如图1-14右侧所注。理由之一是顶上三层处理具体网络应用（如FTP、Telnet或HTTP）的所有细节，却对通信细节了解很少；底下四层对具体网络应用了解不多，却处理所有的通信细节：发送数据，等待确认，给无序到达的数据排序，计算并验证校验和，等等。理由之二是顶上三层通常构成所谓的*用户进程*（user process），底下四层却通常作为操作系统内核的一部分提供。Unix与其他现代操作系统都提供分隔用户进程与内核的机制。由此可见，第4层和第5层之间的接口是构建API的自然位置。

17～19

1.8　BSD网络支持历史

套接字API起源于1983年发行的4.2BSD操作系统。图1-15展示了各种BSD发行版本的发展

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

史，并注明了TCP/IP的主要发展历程。1990年面世的4.3BSD Reno发行版本随着OSI协议进入BSD内核而对套接字API做了少量的改动。

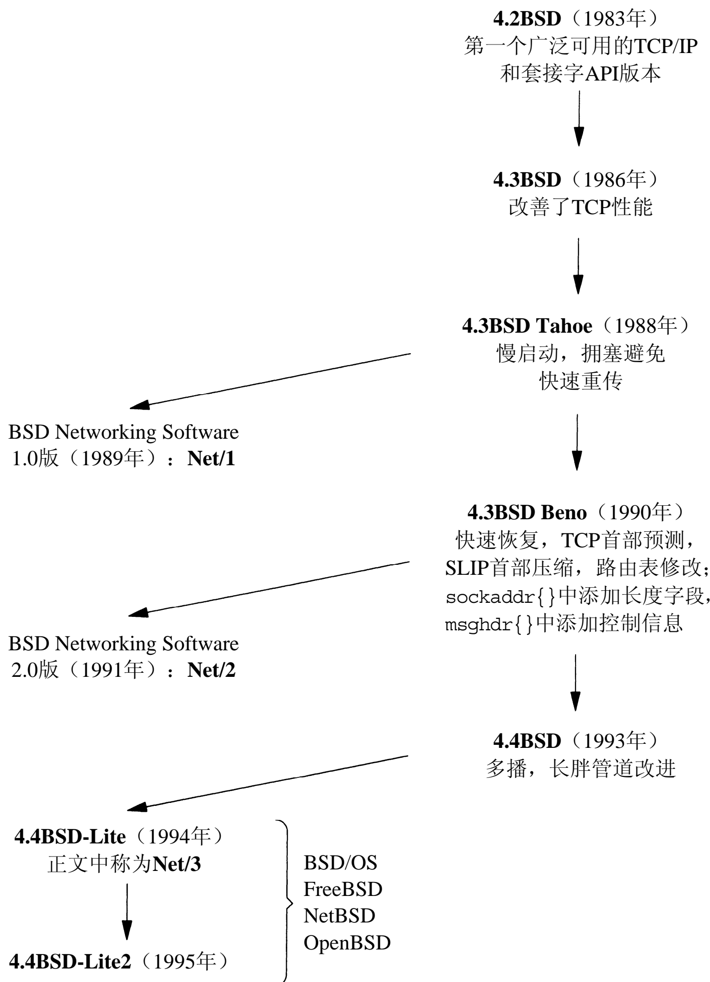

图1-15　各种BSD版本的历史

图1-15中从4.2BSD往下到4.4BSD的通路展示了源自Berkeley计算机系统研究组（Computer Systems Research Group，CSRG）的各个版本，它们要求获取者已拥有Unix的源代码许可权。然而其中的所有网络支持代码，不论是内核支持（如TCP/IP协议栈、Unix域协议栈及套接字API）还是应用程序（如Telnet和FTP客户和服务器程序）都是独立于源自AT&T的Unix代码开发的。因此从1989年起，Berkeley开始提供第一个BSD网络支持版本，它包含所有的网络支持代码以及不受Unix源代码许可权约束的其他各种BSD系统软件。这些包含网络支持代码的版本是可公开获取的，最终因特网上任何人都可通过匿名FTP获取。

源自Berkeley的最终版本是1994年的4.4BSD-Lite和1995年的4.4BSD-Lite2。我们指出这两个版本是其他多个系统（包括BSD/OS、FreeBSD、NetBSD和OpenBSD）的基础，这些系统大多数仍然处于活跃的开发和完善之中。有关各种BSD版本和各种Unix系统历史的详情参见［Mckusick et al.1996］的第1章。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

许多Unix系统从某个版本的BSD网络支持代码（包括套接字API）开始提供网络支持，我们称这些实现为源自Berkeley的实现（Berkeley-derived implementation）。许多商业版本的Unix是基于System V版本4（System V Release 4，SVR4）的，其中有一些系统使用源自Berkeley的网络支持代码（如UnixWare 2.x），其他SVR4系统的网络支持代码却是独立起源的（如Solaris 2.x）。我们还要注意，Linux这种流行的可免费获得的Unix实现并不适合归属源自Berkeley的系列，因为它的网络支持代码和套接字API都是从头开始开发的。

20 ~ 21

1.9 测试用网络及主机

图1-16展示了本书示例所用的各个网络和主机。对于每个主机，我们都标出了它的操作系统和硬件类型（因为有些操作系统可运行在不止一种硬件上）。各个框内的名字就是出现在本书中的各个主机名。

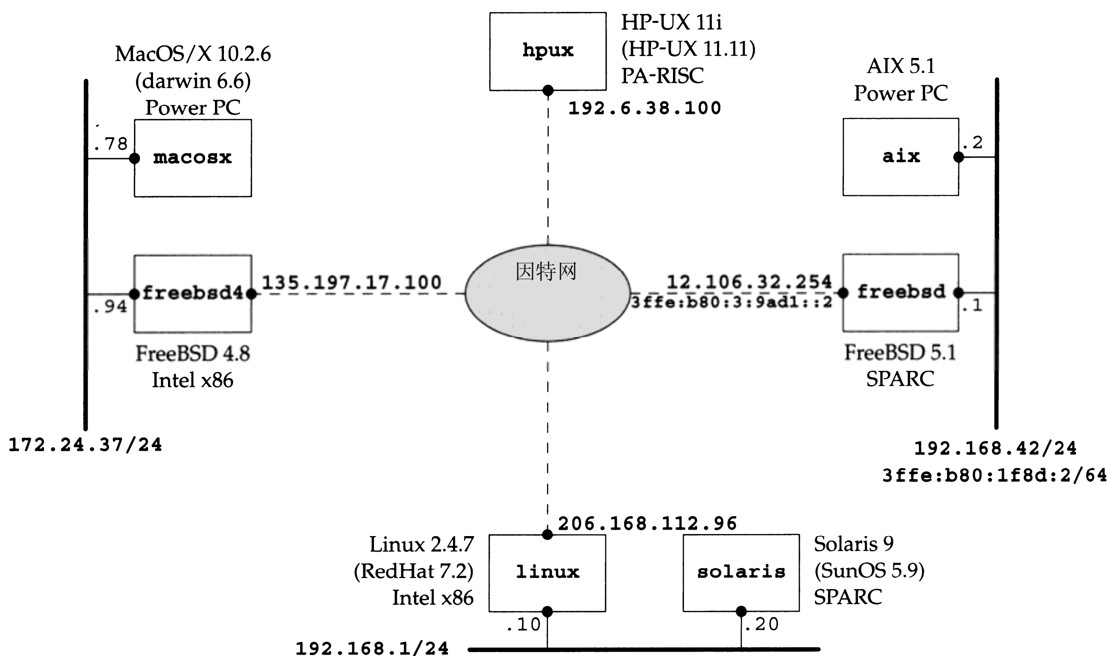

图1-16 本书示例所用的网络和主机

图1-16所示的拓扑适合本书的例子，不过机器大范围地散布在因特网上，物理拓扑实际上变得不太重要。事实上虚拟专用网络（virtual private network，VPN）或安全shell（secure shell，SSH）连接提供这些机器之间的连通性，而无需顾及这些主机的物理位置。

图中“/24”（和/64）指出从地址的最左位开始用于标识网络和子网的连续位数。A.4节将说明现今用于指定子网边界的/*n*记法。

> Sun操作系统的真实名字是SunOS 5.x，而不是Solaris 2.x，但是大家习惯称它为Solaris，实际上这是操作系统和与之捆绑的其他软件的合称。

22

网络拓扑的发现

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

图1-16展示了本书的全部示例所用主机的网络拓扑，但是为了在你自己的网络上运行这些例子和完成习题，你可能需要了解自己的网络拓扑。尽管目前还没有关于网络配置和管理的现行Unix标准，但大多数Unix系统都提供了可用于发现某些网络细节的两个基本命令：netstat和ifconfig。通过阅读所用系统上这些命令的手册页面[①]，你可以获悉有关它们的输出信息的详情。要留意的是，有些厂商把这些命令存放在诸如/sbin或/usr/sbin这样的管理目录中，而不是通常的/usr/bin目录，而这些管理目录可能不在通常的shell搜索路径中（由PATH环境变量指定）。

(1) netstat -i提供网络接口的信息。我们还指定-n标志以输出数值地址，而不是试图把它们反向解析成名字。下面的例子给出了接口及其名字和统计信息：

```
linux % netstat -ni
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500   049211085      0      0      040540958      0      0      0 BMRU
lo    16436   098613572      0      0      098613572      0      0      0 LRU
```

其中环回（loopback）接口称为lo，以太网接口称为eth0。下面的例子给出了支持IPv6的一个主机的类似信息：

```
freebsd % netstat -ni
Name    Mtu Network       Address            Ipkts Ierrs    Opkts Oerrs  Coll
hme0   1500 <Link#1>      08:00:20:a7:68:6b 29100435    35 46561488     0     0
hme0   1500 12.106.32/24  12.106.32.254      28746630     - 46617260     -     -
hme0   1500 fe80:1::a00:20ff:fea7:686b/64
                          fe80:1::a00:20ff:fea7:686b
                                                    0     -        0     -     -
hme0   1500 3ffe:b80:1f8d:1::1/64
                          3ffe:b80:1f8d:1::1        0     -        0     -     -
hme1   1500 <Link#2>      08:00:20:a7:68:6b     51092     0    31537     0     0
hme1   1500 fe80:2::a00:20ff:fea7:686b/64
                          fe80:2::a00:20ff:fea7:686b
                                                    0     -       90     -     -
hme1   1500 192.168.42    192.168.42.1          43584     -    24173     -     -
hme1   1500 3ffe:b80:1f8d:2::1/64
                          3ffe:b80:1f8d:2::1       78     -        8     -     -
lo0   16384 <Link#6>                            10198     0    10198     0     0
lo0   16384 ::1/128       ::1                      10     -       10     -     -
lo0   16384 fe80:6::1/64  fe80:6::1                 0     -        0     -     -
lo0   16384 127           127.0.0.1             10167     -    10167     -     -
gif0   1280 <Link#8>                                6     0        5     0     0
gif0   1280 3ffe:b80:3:9ad1::2/128
                          3ffe:b80:3:9ad1::2        0     -        0     -     -
gif0   1280 fe80:8::a00:20ff:fea7:686b/64
                          fe80:8::a00:20ff:fea7:686b
                                                    0     -        0     -     -
```

23

注意：为了对齐输出字段，我们对较长的代码行做了回行处理。

(2) netstat -r展示路由表，也是另一种确定接口的方法。我们通常指定-n标志以输出数值地址。它还给出默认路由器的IP地址。

```
freebsd % netstat -nr
Routing tables
```

① 手册页面（manual page或man page）是所有Unix系统都提供的使用man命令查看到的有关命令、函数和文件等的帮助信息。某个条目的手册页面就是以该条目为命令行参数执行man的输出。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
Internet:
Destination       Gateway             Flags    Refs     Use Netif    Expire
default           12.106.32.1         USGc       10    6877  hme0
12.106.32/24      link#1              UC          3       0  hme0
12.106.32.1       00:b0:8e:92:2c:00   UHLW        9       7  hme0      1187
12.106.32.253     08:00:20:b8:f7:e0   UHLW        0       1  hme0       140
12.106.32.254     08:00:20:a7:68:6b   UHLW        0       2   lo0
127.0.0.1         127.0.0.1           UH          1   10167   lo0
192.168.42        link#2              UC          2       0  hme1
192.168.42.1      08:00:20:a7:68:6b   UHLW        0      11   lo0
192.168.42.2      00:04:ac:17:bf:38   UHLW        2   24108  hme1       210

Internet6:
Destination                          Gateway              Flags    Netif Expire
::/96                                ::1                  UGRSc     lo0 =>
default                              3ffe:b80:3:9ad1::1   UGSc     gif0
::1                                  ::1                  UH        lo0
::ffff:0.0.0.0/96                    ::1                  UGRSc     lo0
3ffe:b80:3:9ad1::1                   3ffe:b80:3:9ad1::2   UH       gif0
3ffe:b80:3:9ad1::2                   link#8               UHL       lo0
3ffe:b80:1f8d::/48                   lo0                  USc       lo0
3ffe:b80:1f8d:1::/64                 link#1               UC       hme0
3ffe:b80:1f8d:1::1                   08:00:20:a7:68:6b    UHL       lo0
3ffe:b80:1f8d:2::/64                 link#2               UC       hme1
3ffe:b80:1f8d:2::1                   08:00:20:a7:68:6b    UHL       lo0
3ffe:b80:1f8d:2:204:acff:fe17:bf38   00:04:ac:17:bf:38    UHLW     hme1
fe80::/10                            ::1                  UGRSc     lo0
fe80::%hme0/64                       link#1               UC       hme0
fe80::a00:20ff:fea7:686b%hme0        08:00:20:a7:68:6b    UHL       lo0
fe80::%hme1/64                       link#2               UC       hme1
fe80::a00:20ff:fea7:686b%hme1        08:00:20:a7:68:6b    UHL       lo0
fe80::%lo0/64                        fe80::1%lo0          Uc        lo0
fe80::1%lo0                          link#6               UHL       lo0
fe80::%gif0/64                       link#8               UC       gif0
fe80::a00:20ff:fea7:686b%gif0        link#8               UHL       lo0
ff01::/32                            ::1                  U         lo0
ff02::/16                            ::1                  UGRS      lo0
ff02::%hme0/32                       link#1               UC       hme0
ff02::%hme1/32                       link#2               UC       hme1
ff02::%lo0/32                        ::1                  UC        lo0
ff02::%gif0/32                       link#8               UC       gif0
```

24

(3) 有了各个网络接口的名字，执行ifconfig就可获得每个接口的详细信息。

```
linux % ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:C0:9F:06:B0:E1
          inet addr:206.168.112.96  Bcast:206.168.112.127  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49214397 errors:0 dropped:0 overruns:0 frame:0
          TX packets:40543799 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:1098069974 (1047.2 Mb)  TX bytes:3360546472 (3204.8 Mb)
          Interrupt:11 Base address:0x6000
```

该命令给出了指定接口的IP地址、子网掩码和广播地址。其中的MULTICAST标志通常指明该接口所在主机支持多播。有些ifconfig的实现还提供-a标志，用于输出所有已配置接口的信息。

(4) 找出本地网络中众多主机的IP地址的方法之一是，针对从上一步找到的本地接口的广播

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

地址执行ping命令。

```
linux % ping -b 206.168.112.127
WARNING: pinging broadcast address
PING 206.168.112.127 (206.168.112.127) from 206.168.112.96 : 56(84) bytes of data.
64 bytes from 206.168.112.96: icmp_seq=0 ttl=255 time=241 usec
64 bytes from 206.168.112.40: icmp_seq=0 ttl=255 time=2.566 msec (DUP!)
64 bytes from 206.168.112.118: icmp_seq=0 ttl=255 time=2.973 msec (DUP!)
64 bytes from 206.168.112.14: icmp_seq=0 ttl=255 time=3.089 msec (DUP!)
64 bytes from 206.168.112.126: icmp_seq=0 ttl=255 time=3.200 msec (DUP!)
64 bytes from 206.168.112.71: icmp_seq=0 ttl=255 time=3.311 msec (DUP!)
64 bytes from 206.168.112.31: icmp_seq=0 ttl=64 time=3.541 msec (DUP!)
64 bytes from 206.168.112.7: icmp_seq=0 ttl=255 time=3.636 msec (DUP!)
...
```

1.10　Unix 标准

在编写本书时，最引人注目的Unix标准化活动是由Austin公共标准修订组（The Austin Common Standards Revision Group，CSRG）主持的。他们的努力结果是涵盖1 700多个编程接口的约4 000页内容的规范［Josey 2002］。这些规范既具有IEEE POSIX名字，也具有开放团体的技术标准（The Open Group's Technical Standard）名字。其结果是同一个Unix标准有多个名字来指称：ISO/IEC 9945:2002、IEEE Std 1003.1-2001和单一Unix规范第3版（Single Unix Specification Version 3）都指同一个标准。本书中除了像本节这样需要讨论各种较早期标准各自特性的章节外，我们简单地称这个Unix标准为POSIX规范（The POSIX Specification）。

获取这个统一标准的最简易方法是定购其CD-ROM拷贝或通过Web免费访问。这两种方法的起始点都是http://www.UNIX.org/version3。

25

1.10.1　POSIX 的背景

POSIX（可移植操作系统接口）是Portable Operating System Interface的首字母缩写。它并不是单个标准，而是由电气与电子工程师学会（the Institute for Electrical and Electronics Engineers, Inc.）即IEEE开发的一系列标准。POSIX标准已被国际标准化组织即ISO和国际电工委员会（the International Electrotechnical Commission）即IEC采纳为国际标准（这两个组织合称为ISO/IEC）。下面是POSIX标准的发展简史。

- 第一个POSIX标准是IEEE Std 1003.1-1988（317页）。它详述了进入类Unix内核的C语言接口，涵盖了下述领域：进程原语（fork、exec、信号和定时器）、进程环境（用户ID和进程组）、文件与目录（所有I/O函数）、终端I/O、系统数据库（口令文件和用户组文件）以及tar和cpio归档格式。

  第一个POSIX标准在1986年是称为“IEEE-IX”的试用版。POSIX这个名字是由Richard Stallman建议使用的。

- 第二个POSIX标准是IEEE Std 1003.1-1990（356页），也称为ISO/IEC 9945-1: 1990。从1988版本到1990版本只做了少量的修改。新添的副标题为“Part 1: System Application Program Interface (API) [C Language]”，表明本标准为C语言API。
- 下一个标准是两卷本的IEEE Std 1003.2-1992（约1300页）。它的副标题为“Part 2: Shell and Utilities”。这一部分定义了shell（基于System V的Bourne Shell）和大约100个实用程

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

序（通常从shell启动执行的程序，如awk、basename、vi和yacc等等）。本书称这个标准为POSIX.2。

- 再下一个标准是IEEE Std 1003.1b-1993（590页），先前称为IEEE P1003.4。这是对1003.1-1990标准的更新，添加了由P1003.4工作组开发的实时扩展。1003.1b-1993相比1990年版标准新增的条目包括：文件同步、异步I/O、信号量、存储管理（mmap和共享内存）、执行调度、时钟与定时器以及消息队列。
- 更下一个标准是IEEE Std 1003.1 1996年版[IEEE 1996]（743页），也称为ISO/IEC 9945-1: 1996，它包括1003.1-1990（基本API）、1003.1b-1993（实时扩展）、1003.1c-1995（pthreads）和1003.1i-1995（对1003.1b的技术性修订）。该标准增添了3章关于线程的内容，并另有关于线程同步（互斥锁和条件变量）、线程调度和同步调度的各节。本书称这个标准为POSIX.1。该标准还有一个前言，其中声明ISO/IEC 9945由下面3个部分构成。
  - Part 1: System API (C language)——第1部分：系统API（C语言）。
  - Part 2: Shell and utilities——第2部分：Shell和实用程序。
  - Part 3: System administration——第3部分：系统管理（正在开发中）。

  第1部分和第2部分就是我们所说的POSIX.1和POSIX.2。

> 743页中有超过四分之一的篇幅是一个标题为“Rationale and Notes”（理由与注解）的附录。该附录含有历史性信息和某些特性被加入或删除的理由。这些理由通常跟正式标准一样有教益。

26

- 最后一个标准是在2000年被认可[①]的IEEE Std 1003.1g: Protocol-independent interfaces (PII)。在单一Unix规范第3版（The Single Unix Specification Version 3）面世之前，这是与本书涵盖的主题最为相关的POSIX产品。它是联网API标准，它定义了两个API，并称它们为详尽网络接口（Detailed Network Interface，DNI）。
  - DNI/Socket，基于4.4BSD的套接字API。
  - DNI/XTI，基于X/Open的XPG4规范。

这个标准的工作作为P1003.12工作组（后来改名为P1003.1g）起始于20世纪80年代后期。本书称这个标准为POSIX.1g。

关于各种POSIX标准的当前状况可以访问http://www.pasc.org/standing/sd11.html。

1.10.2 开放团体的背景

开放团体（The Open Group）是由1984年成立的X/Open公司（X/Open Company）和1988年成立的开放软件基金会（Open Software Foundation，OSF）于1996年合并成的组织。它是厂商、工业界最终用户、政府和学术机构共同参加的国际组织。下面是开放团体制定的标准的简要背景。

- X/Open公司于1989年出版了*X/Open Portability Guide*（X/Open移植性指南，XPG）第3期，即XPG3。
- XPG第4期即XPG4出版于1992年，其第2版出版于1994年。这个最新版本也称为“Spec 1 170”，其中魔数1170是系统接口数（926个）、头文件数（70个）和命令数（174个）

① 这里被认可标准（approved standard）意思是成为正式标准前的特定阶段。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

的总和。这组规范的最终名字是X/Open Single Unix Specification（X/Open单一Unix规范），也称为“Unix 95”。

- 单一Unix规范第2版于1997年3月发行。符合这个规范的产品称为“Unix 98”。本书就称这个规范为“Unix 98”。Unix 98的接口数目从1170个增长到1434个，而用于工作站的接口数则达到3 030个，因为它包含公共桌面环境（Common Desktop Environment，CDE），而公共桌面环境又需要X Windows系统和Motif用户接口。本规范的详情参见http://www.UNIX.org/version2和［Josey 1997］。Unix 98为套接字API和XTI API定义了网络支持服务。这个规范与POSIX.1g几乎相同。

不幸的是，X/Open称它们的网络标准为XNS：X/Open Networking Services。定义Unix 98套接字和XTI的文档的这一版本称为“XNS Issue 5”（XNS第5期）。在网络界，XNS已是Xerox Network Systems体系结构的简称。所以，我们避免使用XNS，而称这个X/Open文档为Unix 98网络API标准。

27

1.10.3 标准的统一

如本节开头所提，伴随Austin CSRG发布单一Unix规范第3版，POSIX和开放团体都继续发展，达成统一的标准。CSRG促成50多家公司就单一标准达成一致意见，这在Unix发展史上确实是一件划时代之大事。如今大多数Unix系统都符合POSIX.1和POSIX.2的某个版本，不少系统符合单一Unix规范第3版。

历史上多数Unix系统或者源自Berkeley，或者源自System V，不过这些差别在慢慢消失，因为大多数厂商已开始采纳这些标准。然而在系统管理的处理上两者仍然存在较大差别，这个领域目前还没有标准可循。

本书的焦点是单一Unix规范第3版，其中又以套接字API为主。只要可能，我们就使用标准函数。

1.10.4 因特网工程任务攻坚组

因特网工程任务攻坚组（Internet Engineering Task Force，IETF）是一个由关心因特网体系结构的发展及其顺利运作的网络设计者、操作员、厂商和研究人员联合组成的开放的国际团体。它向任何感兴趣的个人开放。

因特网标准处理过程在RFC 2026［Bradner 1996］中说明。因特网标准一般处理协议问题而不是编程API，不过仍有两个RFC（RFC 3493［Gilligan et al. 2003］和RFC 3542［Stevens et al. 2003］）说明了IPv6的套接字API。它们是信息性的RFC，并不是标准，制定它们的目的是加速部署由多家从事IPv6工作较早的厂商所开发的可移植网络应用程序。尽管标准主体趋于花费很长的时间，其中许多API却已经在单一Unix规范第3版中标准化了。

1.11 64位体系结构

20世纪90年代中期到末期开始出现向64位体系结构和64位软件发展的趋势。其原因之一是在每个进程内部可以由此使用更长的编址长度（即64位指针），从而可以寻址很大的内存空间（超过2^{32}字节）。现有32位Unix系统上共同的编程模型称为ILP32模型，表示整数（I）、长整数（L）和指针（P）都占用32位。64位Unix系统上变得最为流行的模型称为LP64模型，表示只有长整数（L）和指针（P）占用64位。图1-17对这两种模型进行了比较。

28

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

从编程角度看，LP64模型意味着我们不能假设一个指针能存放在一个整数中。我们还必须考虑LP64模型对现有API的影响。

数据类型	ILP32模型	LP64模型
char	8	8
short	16	16
int	32	32
long	32	64
指针	32	64

图1-17　ILP32和LP64模型保存不同数据类型所占用的位数的比较

ANSI C创造了size_t数据类型，它用于作为malloc的唯一参数（待分配的字节数），或者作为read和write的第三个参数（待读或写的字节数）。在32位系统中size_t是一个32位值，但是在64位系统中它必须是一个64位值，以便发挥更大寻址模型的优势。这意味着64位系统中也许含有一个把size_t定义为unsigned long的typedef指令。联网API存在如下问题：POSIX.1g的某些草案规定，存放套接字地址结构大小的函数参数具有size_t数据类型（如bind和connect的第三个参数）。某些XTI结构也含有数据类型为long的成员（如t_info和t_opthdr结构）。如果这些规定不加修改，当Unix系统从ILP32模型转变为LP64模型时，size_t和long都将从32位值变为64位值。这两个例子实际上并不需要使用64位的数据类型：套接字地址结构的长度最多也就几百个字节，给XTI的结构成员使用long数据类型则是个错误。

处理这些情况的办法是使用专门设计的数据类型。套接字API对套接字地址结构的长度使用socklen_t数据类型，XTI则使用t_scalar_t和t_uscalar_t数据类型。不把这些值由32位改为64位的理由是易于为那些已在32位系统中编译的应用程序提供在新的64位系统中的二进制代码兼容性。

1.12　小结

图1-5展示了一个尽管简单但却完整的TCP客户程序，它从某个指定的服务器读取当前时间和日期；而图1-9则展示了其服务器程序的一个完整版本。这两个例子引入了许多本书其他部分将要扩展的概念和术语。

我们的客户程序与IPv4协议相关，我们于是把它修改成使用IPv6，但这样做却只是给了我们另外一个协议相关的程序。我们将在第11章中开发一些可用来编写协议无关代码的函数，这在因特网开始使用IPv6后会变得非常重要。

纵贯本书，我们将使用1.4节中介绍的包裹函数来缩短代码，同时又保证测试每个函数调用，检查是否返回错误。我们的包裹函数都以一个大写字母开头。

29

单一Unix规范第3版有多个名称，我们简单地称之为POSIX规范。它是两个长期发展的标准团体各自努力的汇合，由Austin CSRG最终团结起来。

对Unix网络支持历史感兴趣的读者可参阅叙述Unix历史的［Salus 1994］和叙述TCP/IP及因特网历史的［Salus 1995］。

习题

1.1　按1.9节末尾的步骤找出你自己的网络拓扑的信息。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.2　获取本书示例的源代码（见前言），编译并测试图1-5所示的TCP时间获取客户程序。运行这个程序若干次，每次以不同IP地址作为命令行参数。

1.3　把图1-5中的`socket`的第一参数改为9999。编译并运行这个程序。结果如何？找出对应于所输出出错的`errno`值。你如何可以找到关于这个错误的更多信息？

1.4　修改图1-5中的`while`循环，加入一个计数器，累计`read`返回大于零值的次数。在终止前输出这个计数器值。编译并运行你的新客户程序。

1.5　按下述步骤修改图1-9中的程序。首先，把赋于`sin_port`的端口号从13改为9999。然后，把`write`的单一调用改为循环调用，每次写出结果字符串的一个字节。编译修改后的服务器程序并在后台启动执行。接着修改前一道习题中的客户程序（它在终止前输出计数器值），把赋于`sin_port`的端口号从13改为9999。启动这个客户程序，指定运行修改后的服务器程序的主机的IP地址作为命令行参数。客户程序计数器的输出值是多少？如果可能，在不同主机上运行这个客户与服务器程序。

30