

精选习题答案

第1章

1.3 在Solaris上我们得到

```
solaris % daytimetcpccli 127.0.0.1
socket error: Protocol not supported
```

要找出有关这个错误的详细信息，我们首先在<sys/errno.h>头文件中使用grep查找字符串Protocol not supported。

```
solaris % grep 'Protocol not supported' /usr/include/sys/errno.h
#define EPROTONOSUPPORT 120      /* Protocol not supported */
```

这就是由socket返回的errno值。我们然后查看手册页面：

```
solaris % man socket
```

大多数手册页面在将近结束处形如“Errors”的标题下给出这个错误的额外信息，不过有些简洁。

1.4 我们把第一个声明改成：

```
int      sockfd, n, counter = 0;
```

再作为while循环的第一个语句加上如下行：

```
counter++;
```

最后在结束之前加上如下行：

```
printf("counter = %d\n", counter);
```

所显示的值总是1。

1.5 我们声明一个名为i的int变量，再把write调用改为：

```
for (i = 0; i < strlen(buff); i++)
    Write(connfd, &buff[i], 1);
```

其结果随客户主机和服务器主机而定。如果客户和服务器运行在同一个主机上，那么计数器值通常是1，意味着尽管服务器调用了26次write，所写出数据也仅由客户的一次read返回。然而如果客户运行在Solaris 2.5.1上而服务器运行在BSD/OS 3.0上，那么计数器值通常是2。如果监视以太网上的分组，我们发现第一个字符自成一个分组发送，剩余25个字符包含在下一个分组内发送。（我们在7.9节就Nagle算法的讨论解释了如此行为的原因。）相反，如果客户运行在BSD/OS 3.0上而服务器运行在Solaris 2.5.1上，那么计数器值是26。如果监视以太网上的分组，我们发现每个字符自成一个分组发送。

本例子的目的在于强调不同的TCP对数据做不同的处理，我们的应用程序必须做好作为字节流读入这些数据的准备，直到遇上数据流末尾。

第 2 章

- 2.1 访问 <http://www.iana.org/numbers.htm>，找到名为“IP Version Number”的注册处，我们看到版本0是保留的，版本1~3未曾分配，版本5是网际网流协议（Internet Stream Protocol）。
- 2.2 所有RFC都可以通过电子邮件、匿名FTP或Web免费获取。起始点之一是 <http://www.ietf.org>。目录 <ftp://ftp.isi.edu/in-notes> 是一个存放RFC的位置。可以从取得当前RFC索引开始，它通常是文件 `rfc-index.txt`（也可以取得它的HTML版本 <http://www.rfc-editor.org/rfc-index.html>）。使用某种形式的编辑器搜索RFC索引（参见上一个习题的解答）查找“Stream”一词，我们发现RFC 1819定义了网际网流协议的版本2。无论何时查找可能是由某个RFC涵盖的信息，别忘了先搜索RFC索引。
- 2.3 对于IPv4这个默认值产生576字节的IP数据报（其中IPv4首部占用20字节，TCP首部占用20字节，剩下536字节的TCP净荷），这是IPv4的最小重组缓冲区大小。
- 2.4 本例子中执行主动关闭操作的是服务器而不是客户。
- 2.5 令牌环网上的主机不能发送超过1460字节的数据，因为它接收到的MSS是1460。以太网上的主机可以发送最多4096字节的数据，但是为了避免分片，它不会超过外出接口（即以太网）的MTU。TCP净荷不能超过由对端宣告的MSS，但是净荷小于这个数量的TCP分节总是可以发送的。
- 2.6 Assigned Numbers网页（<http://www.iana.org/numbers.htm>）中的“Protocol Numbers”注册处给出OSPF的协议号为89。
- 2.7 选择性确认只是表明由选择性确认消息反映的序列号所涵盖的数据已被接收，而累积确认表明由累积确认消息中的序列号指示的所有以前的数据都已被接收。如果从发送缓冲区中基于选择性确认释放数据，那么系统只能释放确切被确认的数据，而不能释放之前或之后的任何数据。

914

第 3 章

- 3.1 C中函数不能改变按值传递的参数的值。要让被调用的函数修改由调用者传入的某个值，调用者必须传递指向这个待修改值的一个指针。
- 3.2 指针必须按所读或所写的字节数增长，但是C不允许void指针如此增长（因为C编译器不知道void指针指向的数据类型）。

第 4 章

- 4.1 看一下除INADDR_ANY（它的各位全为0）和INADDR_NONE（它的各位全为1）外以INADDR_打头的各个常值的定义。譬如说D类多播地址INADDR_MAX_LOCAL_GROUP的定义是0xe00000ff，其注释是“224.0.0.255”，它显然是按主机字节序定义的。
- 4.2 下面是在connect调用之后新添加的若干行：

```
len = sizeof(cliaddr);
Getsockname(sockfd, (SA *) &cliaddr, &len);
printf("local addr: %s\n",
      Sock_ntop((SA *) &cliaddr, len));
```

这要求声明len为socklen_t变量并声明cliaddr为struct sockaddr_in变量。注意

getsockname的值-结果参数（len）必须在调用之前初始化成由第二个参数所指向变量的大小。涉及值-结果参数的最常见编程错误就是忘记了这样的初始化。

- 4.3 子进程调用close时引用计数从2递减为1，因此不会向客户发送FIN。以后当父进程调用close时引用计数递减为0，于是发送FIN。
- 4.4 accept返回EINVAL，因为它的第一个参数不是一个监听套接字描述符。
- 4.5 不调用bind的话，listen调用赋予监听套接字一个临时端口。

第5章

- 5.1 TIME_WAIT状态的持续时间应该在1分钟到4分钟之间，前提是MSL在30秒钟到2分钟之间。
- 5.2 把一个二进制文件作为客户的标准输入时我们的客户/服务器程序并不工作。假设前3个字节为二进制数1、二进制数0和一个换行符。图5-5中fgets调用最多读入MAXLINE-1个字符，除非碰到换行符或已到达文件尾而提前返回。在本例子中它将读入前3个字符，然后以一个空字节结束待返回的字符串。然而图5-5中strlen调用返回的是1，因为它只计到第一个空字节。客户于是只把第一个字节发送给服务器，导致服务器阻塞在readline调用上，等待一个换行符。客户也阻塞在等待服务器的应答上。这就是所谓的死锁（deadlock）：两个进程都阻塞在等待因对方原因而永远不会到达的事件上。这里的问题是fgets以一个空字节表征所返回数据的结尾，因此它读入的数据不能含有任何空字节。
- 5.3 Telnet把输入行转换成NVT ASCII（TCPv1的26.4节），意味着以CR（回车符）后跟LF（换行符）的双字节序列终止每一行。而我们的客户程序只加一个换行符。尽管如此，我们仍然可以使用Telnet客户与我们的服务器通信，因为我们的服务器回射每个字符，包括每个换行符之前的回车符。
- 5.4 连接终止序列的最后两个分节并不发送。我们杀掉服务器子进程之后（在客户输入“another line”之前），客户向服务器发送数据导致服务器TCP响应以一个RST。这个RST使得连接中止，并防止连接的服务器端（执行主动关闭的那一端）经历TIME_WAIT状态。
- 5.5 没有什么变化，因为在服务器主机上新启动的服务器进程创建一个监听套接字就等待新的连接请求的到达。我们在步骤3发送的是需进入某个ESTABLISHED状态TCP连接的数据分节，而新启动的服务器在其监听套接字上绝看不到这些数据分节，因此服务器主机的TCP对它们的响应仍然是RST。
- 5.6 图E-1给出了这个程序。在Solaris上运行它产生如下输出：

```
solaris % tsigpipe 192.168.1.10
SIGPIPE received
write error: Broken pipe
```

第一个2秒钟的sleep用于让daytime服务器发送应答并关闭它的连接所在端。第一个write导致发送一个数据分节到服务器，服务器则响应以RST（因为daytime服务器已经完全关闭了它的套接字）。注意TCP允许我们继续写出到一个已收到FIN的套接字。第二个sleep让客户接收到服务器的RST，于是第二个write引发SIGPIPE信号。既然信号处理函数返回主控制流，write于是返回一个EPIPE的错误。

```

1 #include      "unp.h"
2 void
3 sig_pipe(int signo)
4 {
5     printf("SIGPIPE received\n");
6     return;
7 }

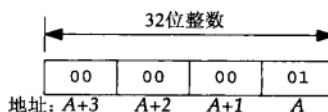
8 int
9 main(int argc, char **argv)
10 {
11     int      sockfd;
12     struct sockaddr_in servaddr;
13
14     if (argc != 2)
15         err_quit("usage: tcpcli <IPaddress>");
16
17     sockfd = Socket(AF_INET, SOCK_STREAM, 0);
18
19     bzero(&servaddr, sizeof(servaddr));
20     servaddr.sin_family = AF_INET;
21     servaddr.sin_port = htons(13); /* daytime server */
22     Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
23
24     Signal(SIGPIPE, sig_pipe);
25
26     Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
27     sleep(2);
28     Write(sockfd, "hello", 5);
29     sleep(2);
30     Write(sockfd, "world", 5);
31
32     exit(0);
33 }

```

tcpcliserv/tsigpipe.c

图E-1 产生SIGPIPE

- 5.7 假设服务器主机支持弱端系统模型（weak end system model，在8.8节讲解），那么一切正常。也就是说即使目的IP地址是右端数据链路的IP地址，服务器主机也会接受到达左端数据链路的外来IP数据报（本例子中它承载一个TCP分节）。我们可以如下测试这一点：在主机linux（图1-16）上运行服务器，然后在主机solaris上启动客户，不过给客户指定的是服务器主机的另一个IP地址（206.168.112.96）。连接建立之后如果在服务器主机上运行netstat，我们将看到该连接的本地IP地址是来自客户SYN的目的IP地址，而不是SYN到达数据链路的IP地址（这跟我们在4.4节提及的一样）。
- 5.8 我们的客户运行在小端字节序的Intel系统上，那儿32位整数值1按图E-2所示格式存放。



图E-2 32位整数值1的小端字节序格式表示

这4个字节按A、A+1、A+2和A+3的顺序通过套接字发送，然后以如图E-3所示的大端

字节序格式存放。

01	00	00	00
A	A+1	A+2	A+3

图E-3 来自图E-2的32位整数以大端字节序格式表示

值0x01000000就是16777216。类似地，由客户发送的整数2将被服务器解释成0x02000000即33554432。这两个整数的和是50331648即0x03000000。服务器把这个大端字节序值发送给客户后，客户把它解释成整数3。

然而32位整数值-22在小端字节序系统上如图E-4表示，采用的是负数的二进制补码表示。

ff	ff	ff	ea
A+3	A+2	A+1	A

图E-4 32位整数值-22的小端字节序格式表示

它在大端字节序服务器主机上被解释成0xeaffffff即-352321537。类似地，-77的小端字节序表示是0xffffffffb3，但是在大端字节序服务器主机上却表示成0xb3ffffffff即-1275068417。服务器上的这两个整数相加的结果是0x9effffe即-1627389954。这个大端字节序的值通过套接字发送给客户后以小端字节序解释的值是0xfeffff9e即-16777314，它就是我们的例子所显示的值。

- 5.9 技术路线是正确的(把二进制值转换成网络字节序表示)，但是不能使用htonl和ntohl这两个函数。尽管这两个函数中的l曾经表意“long”(长整数)，它们却只是操作在32位整数上(3.4节)。64位系统上一个长整数可能占据64位，这两个函数就不能正确工作了。有人也许定义hton64和ntoh64这两个函数来解决本问题，但是它们在使用32位表示长整数的系统上又不能工作。
- 5.10 第一种情形下服务器将永远阻塞在图5-20的readn中，因为客户发送的是2个32位值，但是服务器等待的却是2个64位值。这两个主机之间对换客户和服务器将导致客户发送2个64位值，但是服务器只读入第一个64位，并把它解释成2个32位值。第二个64位值仍然在服务器的套接字接收缓冲区中。服务器往回写出1个32位值，但是客户却仍然永远阻塞在图5-19的readn中，等待读入1个64位值。
- 5.11 IP路由功能查看目的IP地址(服务器主机的IP地址)，搜索路由表确定外出接口和下一跳(ICPv1第9章)。外出接口的主IP地址用作源IP地址，前提是该套接字尚未绑定某个本地IP地址。

918

第6章

- 6.1 这个整数数组包含在一个结构中，而C是允许结构跨等号赋值的。
- 6.2 如果select告诉我们某个套接字可写，该套接字的发送缓冲区就有8192字节的可用空间，但是当我们以8193字节的缓冲区长度对这个阻塞式套接字调用write时，write将会阻塞，等待最后1个字节的可用空间。对阻塞式套接字的读操作只要有数据总会返回一个不足计数(short count)，然而对阻塞式套接字的写操作将一直阻塞到所有数据都能被内核接受为止。可见当使用select测试某个套接字的可写条件时，我们必须

把该套接字预先设置成非阻塞以避免阻塞。

- 6.3 如果两个描述符都可读，那么只执行第一个测试，它测试的是套接字描述符。不过这么做并没有导致客户程序不能工作，它只是降低了效率而已。这就是说，如果select返回值表明两个描述符均可读，那么第一个if语句为真，导致客户从套接字readline并fputs到标准输出。下一个if语句却被跳过（就因为我们在这个if关键词之前所冠的else关键词），不过select接着再次被调用，它马上发现标准输入可读，于是立即返回。这里的关键概念是清除“标准输入可读”条件的不是select的返回，而是从标准输入真正地读入。
- 6.4 使用getrlimit函数取得RLIMIT_NOFILE资源的当前值，然后调用setrlimit把当前软限制 (rlim_cur) 设置成硬限制 (rlim_max)。举例来说，Solaris 2.5上描述符数目的软限制是64，但是任何进程都可以把它增长到默认的硬限制1024。getrlimit和setrlimit不属于POSIX.1，但是在Unix 98中却是必需的。
- 6.5 服务器应用进程持续向客户发送数据，客户TCP确认后扔掉它们。
- 6.6 以参数SHUT_RDWR或SHUT_WR调用shutdown总是发送FIN，而close只在调用时描述符引用计数为1的条件下才发送FIN。
- 6.7 read返回一个错误，我们的Read包裹函数于是终止服务器。服务器不应该如此脆弱。注意我们在图6-26中处理了这种情况，尽管即便这样的代码还是不够健壮。考虑客户和服务器之间丧失连接的情况，服务器某个响应的发送尝试最终发生超时，返回的错误可能是ETIMEDOUT。
- 通常服务器不应该因为这样的原因而中止。它应该登记错误，关闭出错套接字，并继续服务其他客户。对于像这样由单个进程来应付所有客户的服务器来说，简单中止的错误处理方式是难以接受的。然而如果服务器是由一个子进程来应付仅仅一个客户，那么中止某个子进程并不会影响父进程（我们假定它处理所有的新连接并派生子进程）和服务其他客户的任何其他子进程。

919

第7章

- 7.2 图E-5给出了本习题的解答之一。我们去掉了用于显示由服务器返回的数据串的那些语句，因为本例子用不着这些值。

首先声明这个程序没有“唯一正确”的输出，其结果随系统而变化。有些系统（尤如Solaris 2.5.1及更早版本）总是返回0值的套接字缓冲区大小，使得我们无法查看该值在连接前后有什么事发生。

至于MSS，在connect之前显示的值是实现的默认值（通常是536或512），在connect之后显示的值则取决于可能有的来自对端的MSS选项。举例来说，本地以太网上connect之后的值可能是1460。然而connect某个远程网络上的一个服务器主机之后显示的MSS值可能类似默认值，除非你的系统支持路径MTU发现功能。如果可能的话，在程序运行期间运行一个像tcpdump（C.5节）这样的工具，以查看来自对端的SYN分节中的真正MSS选项。

至于套接字接收缓冲区的大小，许多实现在连接建立之后把它向上舍入成MSS的倍数。查看连接建立之后套接字接收缓冲区大小的另一个方法是使用像tcpdump这样的工具监视分组，观察TCP的通告窗口（advertised window）。

```

1 #include "unp.h"
2 #include <netinet/tcp.h>      /* for TCP_MAXSEG */
3 int
4 main(int argc, char **argv)
5 {
6     int sockfd, rcvbuf, mss;
7     socklen_t len;
8     struct sockaddr_in servaddr;
9
10    if (argc != 2)
11        err_quit("usage: rcvbuf <IPaddress>");
12
13    sockfd = Socket(AF_INET, SOCK_STREAM, 0);
14
15    len = sizeof(rcvbuf);
16    Getsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &rcvbuf, &len);
17    len = sizeof(mss);
18    Getsockopt(sockfd, IPPROTO_TCP, TCP_MAXSEG, &mss, &len);
19    printf("defaults: SO_RCVBUF = %d, MSS = %d\n", rcvbuf, mss);
20
21    bzero(&servaddr, sizeof(servaddr));
22    servaddr.sin_family = AF_INET;
23    servaddr.sin_port = htons(13); /* daytime server */
24    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
25
26    Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
27
28 }

```

sockopt/rcvbuf.c

图E-5 在连接建立前后显示套接字接收缓冲区大小和MSS值

7.3 分配一个名为linger的linger结构并如下初始化它：

```

str_cli(stdin, sockfd);

linger.l_onoff = 1;
linger.l_linger = 0;
Setsockopt(sockfd, SOL_SOCKET, SO_LINGER, &linger, sizeof(linger));
exit(0);

```

这应该使得客户TCP以一个RST而不是正常的4分节交换终止连接。服务器子进程的readline调用返回ECONNRESET错误，所显示的消息如下：

```
readline error: Connection reset by peer
```

尽管执行主动关闭的是客户，它也不应该经历TIME_WAIT状态。

7.4 第一个客户调用setsockopt、bind和connect。如果第二个客户在第一个客户调用bind和connect之间调用bind，那么它将返回EADDRINUSE错误。然而一旦第一个客户已连接到对端，第二个客户的bind就正常工作，因为第一个客户的套接字当时处于已连接状态。处理这种竞争状态的唯一办法是让第二个客户在bind调用返回EADDRINUSE

错误的情况下再尝试调用bind多次，而不是一返回该错误就放弃。

7.5 我们在支持多播的一个主机（MacOS X 10.2.6）上运行这个程序^①。

```
macosx % sock -s 9999 &
[1]      29297                                以通配地址启动第一个服务器
macosx % sock -s 172.24.37.78 9999           不使用-A尝试启动第二个服务器
can't bind local address: Address already in use
macosx % sock -s -A 172.24.37.78 9999 &       使用-A再次尝试：成功
[2]      29699
macosx % sock -s -A 127.0.0.1 9999 &          使用-A启动第三个服务器：成功
[3]      29700
macosx % netstat -na | grep 9999
tcp4      0  0   127.0.0.1.9999      *.*      LISTEN
tcp4      0  0   172.24.37.78.9999    *.*      LISTEN
tcp4      0  0   *.9999            *.*      LISTEN
```

920
l
921

7.6 我们首先在支持多播但不支持SO_REUSEPORT选项的一个主机（Solaris 9）上尝试^②。

① 本书第2版解答如下。我们在不支持多播的一个主机（UnixWare 2.1.2）上运行这个程序。

```
unixware % sock -s 9999 &                  以通配地址启动第一个服务器
[1]      29697
unixware % sock -s 206.62.226.37 9999        不使用-A尝试启动第二个服务器
can't bind local address: Address already in use
unixware % sock -s -A 206.62.226.37 9999 &    使用-A再次尝试，成功
[2]      29699
unixware % sock -s -A 127.0.0.1 9999 &        使用-A启动第三个服务器，成功
[3]      29700
unixware % netstat -na | grep 9999
tcp      0  0   127.0.0.1.9999      *.*      LISTEN
tcp      0  0   206.62.226.37.9999    *.*      LISTEN
tcp      0  0   *.9999            *.*      LISTEN
```

——译者注

② 本书第2版解答如下。我们首先在不支持多播的一个主机（UnixWare 2.1.2）上尝试。

```
unixware % sock -s -u -A 206.62.226.37 8888 &  第一个服务器启动
[4]      29707
unixware % sock -s -u -A 206.62.226.37 8888
can't bind local address: Address already in use  不能启动第二个服务器
```

我们给这两个运行实例都指定了SO_REUSEADDR选项，但是它不起作用。

我们接着在支持多播但不支持SO_REUSEPORT选项的一个主机（Solaris 2.6）上尝试。

```
solaris26 % sock -s -u 8888 &                第一个服务器启动
[1]      1135
solaris26 % sock -s -u 8888
can't bind local address: Address already in use
solaris26 % sock -s -u -A 8888 &              使用-A启动第二个服务器：成功
solaris26 % netstat -na | grep 8888
*.*.8888      Idle
*.*.8888      Idle
```

我们看到重复的捆绑

这个系统上第一个bind不必指定SO_REUSEADDR，但是第二个起必须指定。

最后我们在既支持多播又支持SO_REUSEPORT选项的BSD/OS 3.0上进行尝试。我们首先给一前一后两个服务器尝试SO_REUSEADDR选项，但是它不起作用。

```
bsdi % sock -u -s -A 7777 &
[1]      17610
bsdi % sock -u -s -A 7777
can't bind local address: Address already in use
```

接着只给第二个服务器而不给第一个服务器尝试SO_REUSEPORT选项。这也不起作用，因为完全重复的捆绑要求共享同一捆绑的所有套接字都使用该选项。

```
bsdi % sock -u -s -T 8888 &
[1]      17612
bsdi % sock -u -s -T 8888
can't bind local address: Address already in use
```

最后给两个服务器都指定SO_REUSEPORT选项，这是起作用的。

```
bsdi % sock -u -s -T 9999 &
[1]      17614
bsdi % sock -u -s -T 9999 &
[2]      17615
bsdi % netstat -na | grep 9999
udp      0  0   *.9999            *.*      LISTEN
udp      0  0   *.9999            *.*      LISTEN
```

——译者注

```
solaris % sock -s -u 8888 &                               第一个服务器启动
[1]          24051
solaris % sock -s -u 8888
can't bind local address: Address already in use
solaris % sock -s -u -A 8888 &                         使用-A启动第二个服务器; 成功
solaris % netstat -na | grep 8888                         我们看到重复的捆绑
*.8888
*.8888
Idle
Idle
```

这个系统上第一个bind不必指定SO_REUSEADDR，但是第二个起必须指定。

最后我们在既支持多播又支持SO_REUSEPORT选项的MacOS X 10.2.6上进行尝试。我们首先给一前一后两个服务器尝试SO_REUSEADDR选项，但是它不起作用。

```
macosx % sock -u -s -A 7777 &
[1]          17610
macosx % sock -u -s -A 7777
can't bind local address: Address already in use
```

接着只给第二个服务器而不给第一个服务器尝试SO_REUSEPORT选项。这也不起作用，因为完全重复的捆绑要求共享同一捆绑的所有套接字都使用该选项。

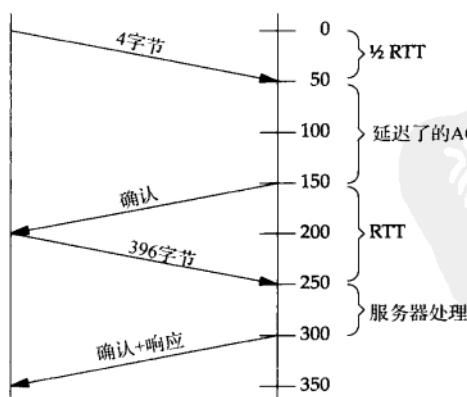
```
macosx % sock -u -s 8888 &
[1]          17612
macosx % sock -u -s -T 8888
can't bind local address: Address already in use
```

最后给两个服务器都指定SO_REUSEPORT选项，这是起作用的。

```
macosx % sock -u -s -T 9999 &
[1]          17614
macosx % sock -u -s -T 9999 &
[2]          17615
macosx % netstat -na | grep 9999
udp4      0      0      *.9999      *.*
```

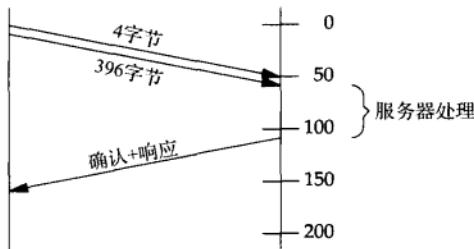
- 7.7 它不起任何作用，因为ping使用ICMP套接字，而SO_DEBUG套接字选项只影响TCP套接字。SO_DEBUG套接字选项的描述一直有些笼统，譬如说“这个选项开启相应协议层中的调试”，但是实现该选项的唯一协议层一直只是TCP。

- 7.8 图E-6给出了时间线。



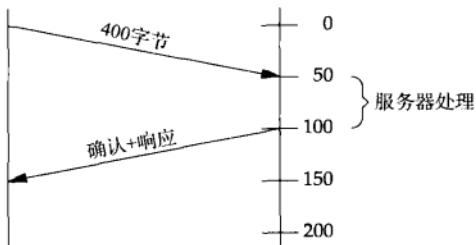
图E-6 Nagle算法与延滞ACK的交互情况

- 7.9 设置TCP_NODELAY套接字选项导致来自第二个write的数据被立即发送，即使该连接上还有一个尚未得到确认的小分组。这种情况如图E-7所示。本例子中总时间只稍微超过150 ms。



图E-7 通过设置TCP_NODELAY套接字选项避免Nagle算法

- 7.10 这种办法的优点是减少了分组的个数，如图E-8所示。



图E-8 使用writev代替设置TCP_NODELAY套接字选项

- 7.11 4.2.3.2节声称“延滞必须低于0.5秒钟，而且在完全大小分节流上至少每隔一个分节就应该有一个ACK”。源自Berkeley的实现延滞ACK最久200ms (TCPv2第821页)。
- 7.12 图5-2中的服务器父进程大部分时间花在阻塞于accept调用中，图5-3中的子进程则大部分时间花在阻塞于read调用中，它是由readline调用的。保持存活选项对于监听套接字不起作用，因此父进程不受客户主机崩溃影响。子进程的read将返回ETIMEDOUT错误，它在跨连接的最后一次数据交换之后约2小时发生。[923]
- 7.13 图5-5中的客户大部分时间花在阻塞于fgets调用中，fgets本身则阻塞在标准I/O函数库中对于标准输入某种类型的读操作中。当跨连接的最后一次数据交换之后约2小时保持存活定时器超时并且所有保持存活侦探分组都没有诱发来自服务器的响应时，套接字的待处理错误被设置成ETIMEDOUT。然而客户阻塞在对于标准输入的fgets调用中，因此看不到这个错误，直到对于套接字执行读或写操作。这就是我们在第6章把图5-5改成使用select的原因之一。
- 7.14 客户大部分时间花在阻塞于select调用中，一旦待处理错误被设置成ETIMEDOUT (如上一题的解答所述)，select就立即返回套接字的可读条件。
- 7.15 只交换2个而不是4个TCP分节。两个系统的定时器精确同步的可能性非常低；因此一端的保持存活定时器会比另一端略早一点超时。首先超时的那一端发送保持存活侦探分组，导致另一端确认这个分组。然而保持存活侦探分组的接收导致时钟略慢的主机

把保持存活定时器重置成2小时。

- 7.16 最初的套接字API并没有listen函数。相反，socket函数的第四个参数含有套接字选项，而SO_ACCEPTCON就是用来指定监听套接字的。加了listen函数后，这个选项还是保留着，不过现在只是由内核来设置（TCPv2第456页）。

第8章

- 8.1 是的。read返回4096字节的数据，recvfrom则返回2048字节（2个数据报中的第一个）。不管应用请求多大，recvfrom决不会返回多于1个数据报的数据。

- 8.2 如果协议使用可变长度套接字地址结构，clilen很可能太大。我们将在第15章看到这对于Unix域套接字地址结构是可以接受的，不过正确编写这个函数的方式是把由recvfrom返回的真正长度用作sendto的长度。

- 8.4 像这样运行ping是查看由运行ping的主机接收到的ICMP消息的简易方法。我们把分组发送频率由通常的每秒钟1次降低到每60秒钟1次以减少输出量。如果我们在主机aix上运行我们的UDP客户程序，所指定的服务器IP地址为192.168.42.1，同时运行ping程序，就会得到如下输出（注意即使指定-v选项，也并非所有ping客户程序都显示所接收到的ICMP错误）：

```
aix % ping -v -i 60 127.0.0.1
PING 127.0.0.1: (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=255 time=0 ms
36 bytes from 192.168.42.1: Destination Port Unreachable
Vr HL TOS Len ID Flg off TTL Pro cks Src Dst Data
4 5 00 0022 0007 0 0000 1e 11 c770 192.168.42.2 192.168.42.1
UDP: from port 40645, to port 9877 (decimal)
```

- 8.5 监听TCP套接字也许有一个套接字接收缓冲区大小，但是它绝不会接受数据。大多数实现并不预先给套接字发送缓冲区或接收缓冲区分配内存空间。使用SO_SNDBUF和SO_RCVBUF套接字选项指定的套接字缓冲区大小仅仅是给套接字设定的上限。

- 8.6 我们在多宿主机freebsd上指定-u选项（使用UDP）和-l选项（指定本地IP地址和端口）运行sock程序。

```
freebsd % sock -u -l 12.106.32.254.4444 192.168.42.2 8888
hello
```

本地IP地址在图1-16中是主机freebsd的因特网侧接口，但是数据报要到达目的地则必须从另一个接口出去。^①使用tcpdump监视网络表明源IP地址确实是客户绑定的那个地址，而不是外出接口的地址。

```
14:28:29.614846 12.106.32.254.4444 > 192.168.42.2.8888: udp 6
14:28:29.615225 192.168.42.2 > 12.106.32.254: icmp: 192.168.42.2
                                udp port 8888 unreachable
```

- 8.7 在客户程序中放一个printf调用会在每个数据报之间引入一个延迟，从而允许服务器接收更多的数据报。在服务器程序中放一个printf调用则会导致服务器丢失更多数据报。

- 8.8 IPv4数据报最大为65535字节，这由图A-1中16位的总长度字段限定。IPv4首部需要20

^① “Connection refused（连接被拒）” 错误的返回是因为sock程序调用connect，导致服务器主机返回ICMP端口不可达错误。——译者注

字节，UDP首部需要8字节，留给UDP用户数据最大65507字节。对于没有任何扩展首部的IPv6数据报而言（自然没有特大报支持，因为特大净荷长度是一个步跳选项，出现在步跳选项扩展首部中），扣除IPv6首部的净荷最大为65535字节（图A-2），再扣除8字节UDP首部，留给UDP用户数据最大65527字节。^①

图E-9给出了dg_cli函数的新版本。如果没有预先设置发送缓冲区大小，源自Berkeley的内核就给sendto调用返回EMSGSIZE错误，因为套接字发送缓冲区的默认大小通常不足以暂存最大的UDP数据报（先做完习题7.1）。然而如果我们运行如图E-9所示客户程序，先设置客户套接字缓冲区大小再发送和接收UDP数据报，那么服务器不返送任何数据报。我们可以运行tcpdump验证客户的数据报发送到了服务器上，但是如果在服务器程序中放一个printf调用，我们就发现它的recvfrom调用并没有返回这个数据报。问题出在服务器的UDP套接字接收缓冲区小于我们发送的数据报，因此该数据报被丢弃掉而不是被递送到套接字。在FreeBSD系统上我们可通过运行netstat -s命令，并查看接收这个大数据报前后“dropped due to full socket buffers”（因套接字缓冲区满而丢弃）计数器值的变化加以验证。最终的办法就是修改服务器程序，预先设置它的套接字发送缓冲区与接收缓冲区的大小。

```

1 #include    "unp.h"
2 #undef MAXLINE
3 #define MAXLINE 65507
4 void
5 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
6 {
7     int      size;
8     char    sendline[MAXLINE], recvline[MAXLINE + 1];
9     ssize_t n;
10    size = 70000;
11    Setssockopt(sockfd, SOL_SOCKET, SO_SNDBUF, &size, sizeof(size));
12    Setssockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size));
13    Sendto(sockfd, sendline, MAXLINE, 0, pservaddr, servlen);
14    n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
15    printf("received %d bytes\n", n);
16 }
```

udpcliserv/dgclibig.c

图E-9 写出最大大小的UDP/IPv4数据报

大多数网络上65535字节的IP数据报需要分片。回顾2.11节，我们知道IP层必须支持的重组缓冲区大小只有576字节，因此你可能会碰到接收不了本习题发送的最大大小数据报的主机。另外源自Berkeley的许多实现（包括4.4BSD-Lite 2）有一个正负号缺陷（bug），它导致UDP不能接受大于32767字节的数据报（TCPv2第770页第95行）。

第9章

9.1 总地说来，整体上接受短期请求偶尔需要长期会话的应用系统可以利用sctp_peeloff。举例来说，某个类似传统UDP的SCTP应用系统中，服务器通常有如短期事务处理那般响应客户的请求，不过偶尔被请求执行长期的音频数据传送。大多

^① 本书第2版Stevens先生可能据于早期IPv6规范得出UDP/IPv6用户数据最大为65 487字节（65535-40-8），第3版新作者尽管一直在强调最新的IPv6规范，在UDP/IPv6用户数据的计算上却仍然使用第2版不合时的推导。译者对此做了修正。——译者注

数情况下服务器发送少数几个短小的UDP消息就行了；然而一旦音频请求到达，长期会话就被激活，以发送音频信息。这种情形下可以使用该函数把音频流剥离出来给专门的线程或进程处理。

- 9.2 因为SCTP不支持半关闭状态，造成当客户调用close时，关联终止序列把来自服务器的任何已排队但尚未处理的未决数据冲刷掉以终止关联，达到关闭关联目的。
- 9.3 对于一到一式套接字客户必须首先调用sctp_connect显式建立关联，但是该函数没有额外指定数据的参数，因此无法在四路握手的第三个分组中随COOKIE ECHO消息携带数据。对于一到多式套接字客户不比先建立关联再发送数据，而是可以调用sctp_sendto同时完成两者，也就是说这样发送的数据随COOKIE ECHO消息被IP数据报载送到对端，这样的关联是隐式建立的。
- 9.4 本端准备与之建立关联的对端在关联建立阶段能够发送回数据的唯一可能情形是它在关联建立之前就准备好了数据。当两端都使用一到多式套接字几乎同时发送数据隐式建立关联时就会发生这种情形。这种关联建立称为INIT冲突，详见〔Stewart and Xie 2001〕第4章。
- 9.5 某些情况下并非所有绑定的地址都可以传递到对端端点。具体地说，如果某个应用进程绑定的IP地址中既有公用的又有私用的，那么可能只有公用地址可以与对端共享。另一个例子是IPv6的链路局部地址不一定能够与对端共享。

第 10 章

- 10.1 如果sctp_sendmsg调用返回错误，那就不会发送任何消息，客户进程于是阻塞在sctp_recvmsg调用中，等待永远不会返送回来的响应消息。解决该问题的办法显然是检查这个函数调用的返回值，如果发现消息发送出错，那就报告错误而不再接收。如果sctp_recvmsg调用返回错误，那就不会有响应消息达到，客户进程循环回去继续尝试发送消息，可能导致建立新的关联。避免这个问题的办法也是检查这个函数调用的返回值，根据情况或者报告错误并关闭套接字，从而让服务器也收到一个错误，或者若错误是暂时的则重新尝试sctp_recvmsg调用。
- 927 10.2 如果服务器在接收一个请求后退出，客户将被永远挂起，等着决不会到来的消息。客户检测这种情况的方法之一是开启关联事件。这样当服务器退出时客户将收到一个消息，告知客户该关联已经不复存在。客户可就此采取恢复手段，譬如说联系另外一个服务器。方法之二是客户启动一个定时器，过一段时间收不到响应消息就取消关联。
- 10.3 我们选择800字节是为了试图让每个SCTP块处于单个分组中。更好的方法也许是通过SCTP_MAXSEG套接字选项确定适合一个SCTP块的大小。
- 10.4 Nagle算法（由SCTP_NODELAY套接字选项控制，见7.10节）只会在选择较小的数据传送大小前提下导致问题。只要以迫使SCTP立即发送的大小写出数据就不会发生危害。然而如果给out_sz选择一个偏小的值，结果就会发生扭曲，暂缓发送某些数据以等待来自对端的SACK。因此如果使用较小的大小值，那么禁止Nagle算法（即开启SCTP_NODELAY套接字选项）可能比较可取。
- 10.5 如果应用进程在建立一个关联之后改动流的数目，那么这个关联的实际流数不会改变。这是因为流数的变更仅仅影响新的关联，而不影响现有关联。

10.6 一到多式套接字允许隐式设置关联。为了使用辅助数据更改某个关联的设置，我们首先需要使用sendmsg调用把这些数据提供给对端。因此请求更多的流要求使用辅助数据通过sendmsg进行隐式关联重新设置。

第 11 章

11.1 图E-10给出了调用gethostbyaddr的程序。

names/hostent2.c

```

1 #include    "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     char      *ptr, **pptr;
6     char      str[INET6_ADDRSTRLEN];
7     struct hostent *hptr;
8
9     while (--argc > 0) {
10         ptr = *++argv;
11         if ((hptr = gethostbyname(ptr)) == NULL) {
12             err_msg("gethostbyname error for host: %s: %s",
13                     ptr, hstrerror(h_errno));
14             continue;
15         }
16         printf("official hostname: %s\n", hptr->h_name);
17         for (pptr = hptr->h_aliases; *pptr != NULL; pptr++)
18             printf("    alias: %s\n", *pptr);
19
20         switch (hptr->h_addrtype) {
21         case AF_INET:
22 #ifdef AF_INET6
23         case AF_INET6:
24             pptr = hptr->h_addr_list;
25             for ( ; *pptr != NULL; pptr++) {
26                 printf("\taddress: %s\n",
27                         Inet_ntop(hptr->h_addrtype, *pptr, str, sizeof(str)));
28                 if ((hptr = gethostbyaddr(*pptr, hptr->h_length,
29                                         hptr->h_addrtype)) == NULL)
30                     printf("\t(gethostbyaddr failed)\n");
31                 else if (hptr->h_name != NULL)
32                     printf("\tname = %s\n", hptr->h_name);
33                 else
34                     printf("\t(no hostname returned by gethostbyaddr)\n");
35             }
36             break;
37         default:
38             err_ret("unknown address type");
39             break;
40     }
41     exit(0);
42 }
```

names/hostent2.c

图E-10 图11-3改成调用gethostbyaddr的结果

本程序针对只有一个IP地址的主机运行没有问题。如果针对拥有8个IP地址的一个主机运行图11-3中的程序，我们得到如下输出：

```
freebsd % hostent cnn.com
official hostname: cnn.com
    address: 64.236.16.20
    address: 64.236.16.52
    address: 64.236.16.84
    address: 64.236.16.116
    address: 64.236.24.4
    address: 64.236.24.12
    address: 64.236.24.20
    address: 64.236.24.28
```

但是如果我们针对同一个主机运行图E-10中的程序，那么它只输出其中一个IP地址：

```
freebsd % hostent2 cnn.com
official hostname: cnn.com
    address: 64.236.24.4
    name = www1.cnn.com
```

928
929

问题在于gethostbyname和gethostbyaddr这两个函数共享同一个hostent结构，就如11.18节开首部分所示。当我们的新程序在调用gethostbyname之后调用gethostbyaddr时，它重用了这个结构以及由它指向的存储区（即h_addr_list指针数组及由该数组所指向的数据），结果冲掉了由gethostbyname返回的其余7个IP地址。

- 11.2 如果你的系统不支持重入版本的gethostbyaddr（我们将在11.19节讲解），那么你必须在调用gethostbyaddr之前复制由gethostbyname返回的指针数组以及由该数组所指向的数据。
- 11.3 chargen服务器一直向客户发送数据，直到客户关闭连接为止（也就是说你中断客户为止）。
- 11.4 这是较新版本BIND的一个特性，不过POSIX没有规定这种处理方式，在可移植程序中不能依赖它。图E-11给出了图11-4中程序的修改后版本。对主机名字符串的测试顺序很重要。我们首先调用inet_pton，因为它是一个快速的全内存访问测试函数，用于判定主机名字符串是不是一个有效的点分十进制数IP地址。仅当这种测试失败时我们才调用gethostbyname，它往往牵涉某些网络资源，因而得花一段时间。
如果这个字符串是一个有效的点分十进制数IP地址，我们就自行构造指向这个IP地址的指针数组（addrs），它使得以后使用pptr的循环代码保持不变。
既然主机名字符串已被转换成套接字地址结构中的二进制数格式，我们于是进入使用pptr的循环。我们把图11-4中的memcpy调用改为memmove（两者功能相同，不过后者能够正确处理源目的内存区重叠的情形），这是因为如果主机名字符串是一个点分十进制数IP地址，那么调用memmove的源和目的内存区是相同的。
- 11.5 图E-12给出了这个程序。
我们使用由gethostbyname返回的h_addrtype值判定地址类型，并使用我们的sock_set_port和sock_set_addr这两个函数（见3.8节）在合适的套接字地址结构中设置端口和地址这两个字段。
本程序尽管能够工作，却存在两个局限。首先，我们必须处理所有差异，查看h_addrtype后再适当地设置sa和salen。更好的办法是由某个库函数不仅完成主机名

和服务名的查找，而且完成整个套接字地址结构的填写（例如11.6节的getaddrinfo）。其次，本程序只在支持IPv6的主机上能够编译。要在仅仅支持IPv4的主机上编译就得添加不少#define伪代码，从而把代码搞得复杂起来。

```
-----names/daytimetcpccli2.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     struct in_addr  **pptr, *addrs[2];  

9     struct hostent  *hp;  

10    struct servent *sp;  

11  

12    if (argc != 3)  

13        err_quit("usage: daytimetcpccli2 <hostname> <service>");  

14  

15    bzero(&servaddr, sizeof(servaddr));  

16    servaddr.sin_family = AF_INET;  

17  

18    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) == 1) {  

19        addrs[0] = &servaddr.sin_addr;  

20        addrs[1] = NULL;  

21        pptr = &addrs[0];  

22    } else if ( (hp = gethostbyname(argv[1])) != NULL) {  

23        pptr = (struct in_addr **) hp->h_addr_list;  

24    } else  

25        err_quit("hostname error for %s: %s", argv[1], hstrerror(h_errno));  

26  

27    if ( (n = atoi(argv[2])) > 0)  

28        servaddr.sin_port = htons(n);  

29    else if ( (sp = getservbyname(argv[2], "tcp")) != NULL)  

30        servaddr.sin_port = sp->s_port;  

31    else  

32        err_quit("getservbyname error for %s", argv[2]);  

33  

34    for ( ; *pptr != NULL; pptr++) {  

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

36  

37        memmove(&servaddr.sin_addr, *pptr, sizeof(struct in_addr));  

38        printf("trying %s\n", Sock_ntop((SA *) &servaddr, sizeof(servaddr)));  

39  

40        if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) == 0)  

41            break;           /* success */  

42        err_ret("connect error");  

43        close(sockfd);  

44    }  

45    if (*pptr == NULL)  

46        err_quit("unable to connect");  

47  

48    while ( (n = Read(sockfd, recvline, MAXLINE)) > 0) {  

49        recvline[n] = 0;      /* null terminate */  

50        Fputs(recvline, stdout);  

51    }  

52    exit(0);  

53 }
```

-----names/daytimetcpccli2.c-----

图E-11 允许点分十进制数IP地址或主机名，端口号或服务名的版本

names/daytimetccli3.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     struct sockaddr_in6 servaddr6;
9     struct sockaddr *sa;
10    socklen_t salen;
11    struct in_addr **pptr;
12    struct hostent *hp;
13    struct servent *sp;
14
15    if (argc != 3)
16        err_quit("usage: daytimetccli3 <hostname> <service>");
17
18    if ( (hp = gethostbyname(argv[1])) == NULL)
19        err_quit("hostname error for %s: %s", argv[1], hstrerror(h_errno));
20
21    if ( (sp = getservbyname(argv[2], "tcp")) == NULL)
22        err_quit("getservbyname error for %s", argv[2]);
23
24    pptr = (struct in_addr **) hp->h_addr_list;
25    for ( ; *pptr != NULL; pptr++) {
26        sockfd = Socket(hp->h_addrtype, SOCK_STREAM, 0);
27
28        if (hp->h_addrtype == AF_INET) {
29            sa = (SA *) &servaddr;
30            salen = sizeof(servaddr);
31        } else if (hp->h_addrtype == AF_INET6) {
32            sa = (SA *) &servaddr6;
33            salen = sizeof(servaddr6);
34        } else
35            err_quit("unknown addrtype %d", hp->h_addrtype);
36
37        bzero(sa, salen);
38        sa->sa_family = hp->h_addrtype;
39        sock_set_port(sa, salen, sp->s_port);
40        sock_set_addr(sa, salen, *pptr);
41
42        printf("trying %s\n", Sock_ntop(sa, salen));
43
44        if (connect(sockfd, sa, salen) == 0)
45            break;          /* success */
46        err_ret("connect error");
47        close(sockfd);
48    }
49    if (*pptr == NULL)
50        err_quit("unable to connect");
51
52    while ( (n = Read(sockfd, recvline, MAXLINE)) > 0) {
53        recvline[n] = 0; /* null terminate */
54        Fputs(recvline, stdout);
55    }
56    exit(0);
57 }

```

names/daytimetccli3.c

图E-12 图11-4中程序同时适用于IPv4和IPv6的修改版本

11.7 分配一个大缓冲区（比任何套接字地址结构都要大）并调用getsockname。它的第三个参数是一个值-结果参数，由它返回真正的协议地址大小。不过这种方法只适合具有固定长度套接字地址结构的协议（例如IPv4和IPv6），对于能够返回可变长度套接字地址结构的协议（例如Unix域套接字，第15章）却不能保证正确工作。

11.8 我们首先分配存放主机名和服务名的数组：

```
char host[NI_MAXHOST], serv[NI_MAXSERV];
```

然后在accept返回之后改为调用getnameinfo以取代sock_ntop：

```
if (getnameinfo(cliaddr, len, host, NI_MAXHOST, serv,
    NI_MAXSERV, NI_NUMERICHOST | NI_NUMERICSERV) == 0)
    printf("connection from %s.%s\n", host, serv);
```

既然这是服务器程序，我们指定NI_NUMERICHOST和NI_NUMERICSERV标志以避免查询DNS和查找/etc/services文件。

11.9 第二个服务器碰到的第一个问题是无法捆绑与第一个服务器相同的端口，这是因为没有设置SO_REUSEADDR套接字选项。最容易的解决办法是制作udp_server函数的一个副本，把它命名为udp_server_reuseaddr，由它设置这个套接字选项，再从服务器程序调用这个新函数。

11.10 当客户输出“Trying 206.62.226.35...”时，gethostbyname已经返回了IP地址。客户在此之前任何停顿是解析器用于查找主机名的时间。输出“Connected to bsdi.kohala.com”意味着connect已经返回。这两个输出行之间的任何停顿是connect用来建立连接的时间。

第 12 章

12.1 下面是相关的摘录片段（省掉了登录和列目录等内容）。主机freebsd上的FTP客户不论使用IPv4还是IPv6总是先尝试EPRT命令，若不工作则退回到PORT命令。

```
freebsd % ftp aix-4
Connected to aix-4.unpbook.com.
220 aix FTP server ...
...
230 Guest login ok, access restrictions apply.
ftp> debug
Debugging on (debug=1).
ftp> passive
Passive mode: off; fallback to active mode: off.
ftp> dir
--> EPRT |1|192.168.42.1|50484|
500 'EPRT |1|192.168.42.1|50484|': command not understood.
disabling epsv4 for this connection
--> PORT 192,168,42,1,197,52
200 PORT command successful.
--> LIST
150 Opening ASCII mode data connection for /bin/ls.
...
freebsd % ftp ftp.kame.net
Trying 2001:200:0:4819:203:47ff:fea5:3085...
Connected to orange.kame.com.
220 orange.kame.com FTP server ...
...
230 Guest login ok, access restrictions apply.
```

```

ftp> debug
Debugging on (debug=1).
ftp> passive
Passive mode: off; fallback to active mode: off.
ftp> dir
--> EPRT |2|3ffe:b80:3:9ad1::2|50480|
200 EPRT command successful.
--> LIST
150 Opening ASCII mode data connection for '/bin/ls'.

```

第 13 章

- 13.1 `daemon_init` 中关闭所有描述符的 `close` 调用也将关闭由 `tcp_listen` 建立的监听 TCP 套接字。既然作为守护进程编写的程序可能是从某个系统启动命令脚本执行的，因此我们不应该假设任何出错消息都能写到某个终端。所有出错消息都应该使用 `syslog` 登记，即使诸如命令行参数无效之类的启动出错消息也不例外。
- 13.2 TCP 版本的 `echo`、`discard` 和 `chargen` 服务器由 `inetd` 派生出来之后作为子进程运行，因为它们需要运行到客户终止连接为止。另外 2 个 TCP 服务器 `time` 和 `daytime` 并不需要 `inetd` 派生子进程，因为它们的服务极易实现（即取得当前时间和日期，把它格式化后写出，再关闭连接），于是由 `inetd` 直接处理。所有 5 个 UDP 服务的处理都不需要 `inetd` 派生子进程，因为每个服务对于引发它的任一客户数据报所作的响应只是最多产生一个数据报。因此这 5 个服务也由 `inetd` 直接处理。
- 13.3 这是一个众所周知的拒绝服务型攻击 ([CERT 1996a])。来自端口 7 的第一个数据报导致 `chargen` 服务器发送回一个数据报到端口 7，它被回射成发送到 `chargen` 服务器的下一个数据报，这样一直循环下去。`FreeBSD` 上实现的解决办法是拒绝源端口和目的端口都是内部服务的外来数据报。另一个常用的解决办法是在每个主机上通过 `inetd` 禁止这些内部服务，或者在一个组织机构接入因特网的路由器上这么做。
- 13.4 客户的 IP 地址和端口取自由 `accept` 填写的套接字地址结构。
`inetd` 对 UDP 套接字无能为力的原因是读入数据报的 `recvfrom` 是由通过 `exec` 激活的真正服务器而不是 `inetd` 本身执行的。
`inetd` 可以仅仅为了获取客户的 IP 地址和端口而指定 `MSG_PEEK` 标志 (14.7 节) 窥读数据报，被窥读的数据报保持原地不动，留待真正的服务器读入。

934

第 14 章

- 14.1 如果未曾建立过信号处理函数，那么第一个 `signal` 调用将返回 `SIG_DFL`，而重新设置信号处理函数的第二个 `signal` 调用只是把它设置回默认处置。
- 14.3 下面是修改后的 `for` 循环：

```

for ( ; ; ) {
    if ( (n = Recv(sockfd, recvline, MAXLINE, MSG_PEEK)) == 0)
        break;      /* server closed connection */

    Iocctl(sockfd, FIONREAD, &npend);
    printf("%d bytes from PEEK, %d bytes pending\n", n, npend);

    n = Read(sockfd, recvline, MAXLINE);
    recvline[n] = 0;      /* null terminate */
    Fputs(recvline, stdout);
}

```

- 14.4 数据仍然输出，因为掉出main函数末尾等同于从这个函数返回，而main函数又是由C启动例程如下调用的：

```
exit(main(argc, argv));
```

因此exit仍然被调用，标准I/O清扫例程也同样被调用。

第 15 章

- 15.1 unlink从文件系统中删除了路径名，此后客户调用connect就会失败。服务器的监听套接字不受影响，不过unlink之后没有客户能够成功connect到其上。

- 15.2 即使路径名仍然存在，客户也无法connect到服务器，这是因为connect成功要求当前有一个打开着的绑定了那个路径名的Unix域套接字（15.4节）。

- 15.3 当服务器通过调用sock_ntop显示客户的协议地址时，输出信息将是“datagram from (no pathname bound)”（数据报来自（无路径名绑定）），因为默认情况下客户的套接字上不绑定任何路径名。

解决办法之一是在udp_client和udp_connect中明确检查是否为一个Unix域套接字，若是则调用bind给它捆绑一个临时路径名。这么做把协议相关处理置于原本所属的库函数中，而不是置于我们的应用程序中。

- 15.4 尽管我们迫使服务器程序为它的26字节应答逐个字节调用write，客户程序中放置的sleep调用还是保证在调用read之前所有26个分节都接收到，使得单个read调用返回完整的应答。这个例子只是为了（再次）验证TCP是一个没有内在记录边界的字节流。[935]
要使用Unix域协议，我们以2个命令行参数/local（或/unix）和/tmp/daytime（或你想使用的任何其他临时路径名）启动客户和服务器。情况没有变化：每次运行客户程序由read返回的都是26个字节。

服务器为每个send指定MSG_EOR标志之后逐个发送的每个字节都被认为是一个逻辑记录，客户每次调用read所返回的也将是1个字节。这里碰巧的是源自Berkeley的实现默认支持MSG_EOR标志。不过这一点没有写在正式文档中，在生产性代码中不应该使用。我们这儿使用它作为表现字节流协议和面向记录协议之差异的一个例子。从实现角度看，每个输出操作都进入一个内存缓冲区（mbuf），MSG_EOR标志由内核随mbuf从发送套接字转移到接收套接字的接收缓冲区维持在mbuf中。调用read时MSG_EOR标志仍然依附在每个mbuf上，因此通用内核read例程（它支持MSG_EOR标志，因为一些协议使用它）独自返回每个字节。如果我们改用recvmsg取代read，它将在每次返回一个字节时还在msg_flags成员中返回MSG_EOR标志。这个特性并不适用于TCP，因为发送端TCP从来不看所发送mbuf中的MSG_EOR标志，而且即使它看了，它也无法在TCP首部中把这个标志传递给接收端TCP。（感谢Matt Thomas指出这个没有写在文档中的“特性”。）

- 15.5 图E-13给出了这个程序的实现。

```
1 #include "unp.h"
2 #define PORT      9999
3 #define ADDR      "127.0.0.1"
4 #define MAXBACKLOG 100
```

debug/backlog.c

图E-13 确定不同的backlog值对应的真正已排队连接数

936

```

5           /* globals */
6 struct sockaddr_in serv;
7 pid_t     pid;                                /* of child */
8 int      pipefd[2];
9 #define pfd pipefd[1]                      /* parent's end */
10 #define cfd pipefd[0]                     /* child's end */

11          /* function prototypes */
12 void    do_parent(void);
13 void    do_child(void);

14 int
15 main(int argc, char **argv)
16 {
17     if (argc != 1)
18         err_quit("usage: backlog");
19     Socketpair(AF_UNIX, SOCK_STREAM, 0, pipefd);
20     bzero(&serv, sizeof(serv));
21     serv.sin_family = AF_INET;
22     serv.sin_port = htons(PORT);
23     Inet_pton(AF_INET, ADDR, &serv.sin_addr);

24     if ( (pid = Fork()) == 0)
25         do_child();
26     else
27         do_parent();
28     exit(0);
29 }

30 void
31 parent_alarm(int signo)
32 {
33     return;      /* just interrupt blocked connect() */
34 }

35 void
36 do_parent(void)
37 {
38     int      backlog, j, k, junk, fd[MAXBACKLOG + 1];
39     Close(cfd);
40     Signal(SIGALRM, parent_alarm);

41     for (backlog = 0; backlog <= 14; backlog++) {
42         printf("backlog = %d: ", backlog);
43         Write(pfd, &backlog, sizeof(int)); /* tell child value */
44         Read(pfd, &junk, sizeof(int));   /* wait for child */

45         for (j = 1; j <= MAXBACKLOG; j++) {
46             fd[j] = Socket(AF_INET, SOCK_STREAM, 0);
47             alarm(2);
48             if (connect(fd[j], (SA *) &serv, sizeof(serv)) < 0) {
49                 if (errno != EINTR)
50                     err_sys("connect error, j = %d", j);
51             printf("%d connections completed\n", j-1);
52             for (k = 1; k <= j; k++)
53                 Close(fd[k]);
}

```

图E-13 (续)

```

54             break; /* next value of backlog */
55         }
56         alarm(0);
57     }
58     if (j > MAXBACKLOG)
59         printf("%d connections?\n", MAXBACKLOG);
60 }
61 backlog = -1; /* tell child we're all done */
62 Write(pfd, &backlog, sizeof(int));
63 }

64 void
65 do_child(void)
66 {
67     int listenfd, backlog, junk;
68     const int on = 1;
69     Close(pfd);

70     Read(cfd, &backlog, sizeof(int)); /* wait for parent */
71     while (backlog >= 0) {
72         listenfd = Socket(AF_INET, SOCK_STREAM, 0);
73         Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
74         Bind(listenfd, (SA *) &serv, sizeof(serv));
75         Listen(listenfd, backlog); /* start the listen */
76         Write(cfd, &junk, sizeof(int)); /* tell parent */
77         Read(cfd, &backlog, sizeof(int)); /* just wait for parent */
78         Close(listenfd); /* closes all queued connections, too */
79     }
80 }

```

937

debug/backlog.c

图E-13 (续)

第 16 章

- 16.1 套接字描述符是在父子进程之间共享的，因此它的引用计数为2。要是父进程调用close，那么这只是把该引用计数由2减为1，而且既然它仍然大于0，FIN就不发送。这就是使用shutdown函数的另一个理由：即使描述符的引用计数仍然大于0，FIN也被强迫发送出去。
- 16.2 父进程将继续写出到已经接收FIN的套接字。它发送给服务器的第一个分节将引发RST响应。此后那个write调用将导致内核像我们在5.12节讨论过的那样向父进程发送SIGPIPE信号。
- 16.3 当子进程调用getppid以向父进程发送SIGTERM信号时，所返回的进程ID将是1即init进程，它是所有孤儿进程的继父（也就是说它继承所有其父进程在子进程仍在运行时就终止的那些子进程）。子进程试图向init进程发送这个信号，但是没有足够的权限。然而如果这个客户程序有机会以超级用户特权运行，从而允许它向init发送信号，那么在发送该信号之前应该检测getppid的返回值。
- 16.4 如果去掉这两行，select就被调用。不过select调用将立即返回，因为连接建立之后套接字是可写的。这个测试加goto语句只是避免不必要的调用select。
- 16.5 如果服务器在accept调用返回之后立即发送数据，而当三路握手的第二个分节到达以在客户端完成连接的时候客户主机却比较忙（图2-5），那么来自服务器的数据可能在客户的connect调用返回之前到达。举例来说，SMTP服务器在未从中读之前就立即往

一个新建立的连接中写，以便给客户发送一个问候消息。

第 17 章

- 17.1 无关紧要，因为图17-2中union的前3个成员都是套接字地址结构。

第 18 章

- 18.1 `sdl_nlen`成员将是5, `sdl_alen`成员将是8。整个`sockaddr_dl`结构需要21个字节，在32位体系结构上则向上舍入成24个字节（TCPv2第89页）。
- 18.2 内核的响应绝不发送到这个套接字。`SO_USELOOPBACK`套接字选项确定内核是否把应答发送给发送进程，TCPv2第649~650页讨论了这一点。它的默认设置是开启，因为大多数进程需要这些应答。禁止该选项将防止内核把应答发送给发送进程。

第 20 章

- 20.1 如果你接收到许多应答，它们每次到达的先后顺序不应该都一样。不过发送主机本身的应答通常是第一个，因为其数据报的来往并不出现在真正的网络上。
- 20.2 FreeBSD上当信号处理函数往管道中写入一个空字节并返回之后，`select`返回`EINTR`。`select`再次被调用时返回管道的可读条件。

第 21 章

- 21.1 我们运行该程序得不到任何输出。为了防止进程偶尔收取并非期待的多播数据报，内核不把接收到的多播数据报送给未曾在其上执行过任何多播操作（譬如加入某个组）的目的地套接字。这里发送的那个UDP数据报的目的地址是224.0.0.1，它是所有具备多播能力的节点都必须参加的所有主机组。该UDP数据报作为一个多播以太网帧发送，因而所有具备多播能力的节点都接收到它，因为它们都属于这个组。然而内核丢弃了接收到的数据报，因为捆绑了`daytime`端口的那个进程（通常就是`inetd`）未曾设置任何多播选项。^①

^① 本书第2版解答如下。我们运行该程序的输出如下：

```
solaris % udpcli05 224.0.0.1
hi
from 206.62.226.34: Thu Jun 19 17:28:32 1997
from 206.62.226.43: Thu Jun 19 17:28:32 1997
from 206.62.226.42: Thu Jun 19 17:28:32 1997
from 206.62.226.40: Thu Jun 19 17:28:32 1997
from 206.62.226.35: Thu Jun 19 17:28:32 1997
```

5个给出响应的主机运行的操作系统有AIX、BSD/OS、Digital Unix和Linux。没有给出响应却具有多播能力的仅有节点是运行Solaris 2.5的主机和Cisco路由器。

这里发送的那个UDP数据报的目的地址是224.0.0.1，它是所有具备多播能力的节点都必须参加的所有主机组。该UDP数据报作为一个多播以太网帧发送，因而所有具备多播能力的节点都接收到它，因为它们都属于这个组。给出响应的主机都把接收到的数据报传递给UDP版本的`daytime`服务器（它通常是`inetd`的一部分），而不管其套接字是否已经加入所有主机组。然而Solaris的实现却要求目的地套接字必须加入所有主机组才能接收该数据报。本例子表明决不是设计来响应多播数据报的UDP程序也能接收到多播数据报。我们在第20章看到过这个`daytime`例子发生同样的事情：决不是设计来响应广播数据报的UDP程序也能接收广播数据报。——译者注

21.2 图E-14给出了调用bind捆绑多播地址和端口0的main函数简单修改版本。

不幸的是，我们尝试运行本程序的3个系统（FreeBSD 4.8、MacOS X和Linux 2.4.7）都允许如此bind，随后发送的UDP数据报具有多播源IP地址。^①

21.3 在支持多播的主机laix上这么做的输出如下：

```
aix % ping 224.0.0.1
PING 224.0.0.1: 56 data bytes
64 bytes from 192.168.42.2: icmp_seq=0 ttl=255 time=0 ms
64 bytes from 192.168.42.1: icmp_seq=0 ttl=64 time=1 ms (DUP!)
^C
---224.0.0.1 PING Statistics---
1 packets transmitted, 1 packets received, +1 duplicates, 0% packet loss
round-trip min/avg/max = 0/0/0 ms
```

图1-16中右侧以太网上两个主机都给出响应。^②

为了防护特定拒绝服务攻击，某些系统默认情况下对于广播或多播ping不给出响应。为了让主机freebsd给出响应，我们必须使用如下命令进行配置：

```
freebsd % sysctl net.inet.icmp.bmcastecho=1
```

21.5 值1073741824转换成浮点数并除以4294967296得到0.250。再乘以1000000得到250000，

940

^① 本书第2版接着解答如下。不幸的是，我们尝试运行本程序的3个系统（BSD/OS、Digital Unix和Solaris 2.5）都允许如此bind，随后发送的UDP数据报具有多播源IP地址。给出响应的那5个系统（跟上一道习题一样）都在应答中对换源IP地址和目的IP地址，结果所有5个应答都是多播数据报！接收了这些应答的具有多播能力的客户主机倒没对它们过度反应，因为这些应答的目的端口就是最初捆绑多播地址时由客户主机的内核选定的临时端口，当时该端口没有绑定在任何套接字上。对于多播UDP数据报，ICMP不产生端口不可达消息。——译者注

^② 本书第2版解答如下。在支持多播的主机solaris上这么做的输出如下：

```
solaris % ping 224.0.0.1
PING 224.0.0.1: 56 data bytes
64 bytes from solaris.kohala.com (206.62.226.33): icmp_seq=0. time=4. ms
64 bytes from linux.kohala.com(206.62.226.40): icmp_seq=0. time=9. ms
64 bytes from aix.kohala.com (206.62.226.43): icmp_seq=0. time=11. ms
64 bytes from bsdi.kohala.com (206.62.226.35): icmp_seq=0. time=13. ms
64 bytes from alpha.kohala.com (206.62.226.42): icmp_seq=0. time=15. ms
64 bytes from sunos5.kohala.com (206.62.226.36): icmp_seq=0. time=17. ms
64 bytes from bsdi2.kohala.com (206.62.226.34): icmp_seq=0. time=54. ms
64 bytes from gw.kohala.com (206.62.226.62): icmp_seq=0. time=75. ms
^?
---224.0.0.1 PING Statistics---
1 packets transmitted, 8 packets received, 8.00 times amplification
round-trip (ms) min/avg/max = 4/24/75
solaris % ping 224.0.0.2
PING 224.0.0.2: 56 data bytes
64 bytes from bsdi.kohala.com (206.62.226.35): icmp_seq=0. time=3. ms
64 bytes from gw.kohala.com (206.62.226.62): icmp_seq=0. time=24. ms
^?
---224.0.0.2 PING Statistics ---
1 packets transmitted, 2 packets received, 2.00 times amplification
round-trip (ms) min/avg/max = 3/13/24
```

对于所有主机组，本书第2版图1-16中顶部以太网上除没有多播能力的unixware外所有主机都给出响应（当然包括发送主机本身）。对于所有路由器组，我们期待bsdi给出响应，因为它是所在子网上的多播路由器，有一个通往MBone（B.2节）的隧道，且运行着mrouted。路由器gw也给出响应，不过它并不扮演多播路由器角色。

——译者注

它以微秒为单位就是1/4秒。

最大的整数小数部分是4294967295，它除以4294967296得到0.9999999976716935634。再乘以1000000并截成整数得到999999，它就是最大的微秒数。

```
mcast/udpcli06.c
1 #include "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int sockfd;
6     socklen_t salen;
7     struct sockaddr *cli, *serv;
8
9     if (argc != 2)
10        err_quit("usage: udpcli06 <IPaddress>");
11
12     sockfd = Udp_client(argv[1], "daytime", (void **) &serv, &salen);
13
14     cli = Malloc(salen);
15     memcpy(cli, serv, salen); /* copy socket address struct */
16     sock_set_port(cli, salen, 0); /* and set port to 0 */
17     Bind(sockfd, cli, salen);
18
19     dg_cli(stdin, sockfd, serv, salen);
20
21     exit(0);
22 }
```

mcast/udpcli06.c

图E-14 捆绑多播地址的UDP客户程序main函数

第22章

- 22.1 我们已经知道sock_ntop使用自己的静态缓冲区存放结果。如果我们在同一个printf中作为参数调用它两次，第二次调用就会覆盖第一次调用的结果。
- 22.2 是的，如果应答中包含0个字节的用户数据的话（也就是仅有一个hdr结构）。
- 22.3 由于select并不修改指定其时间限制的timeval结构，因此你必须记下第一个分组的发送时刻（它已由rtt_ts以微秒为单位返回）。当select返回套接字的可读条件时，记下当前时刻；如果需要再次调用recvmsg，那就给select计算新的超时值。
- 22.4 常用的技巧就如我们在22.6节所做的那样给每个接口地址创建一个套接字，然后就从请求到达的那个套接字发送相应的应答。
- 22.5 既不给出主机名参数也不设置AI_PASSIVE标志就调用getaddrinfo会导致它假定获取本地主机地址0::1（IPv6）或127.0.0.1（IPv4）的信息。回顾一下，我们知道在主机支持IPv6的前提下，getaddrinfo先于IPv4套接字地址结构返回IPv6套接字地址结构。如果主机同时支持这两个协议，那么udp_client中的socket调用尝试将以地址族为AF_INET6的首次尝试成功告终。

图E-15是这个程序的协议无关版本。

```
advio/udpserv04.c
1 #include "unpifi.h"
2 void mydg_echo(int, SA *, socklen_t);
3 int
```

advio/udpserv04.c

图E-15 22.6节中程序的协议无关版本

```

4 main(int argc, char **argv)
5 {
6     int      sockfd, family, port;
7     const int on = 1;
8     pid_t    pid;
9     socklen_t salen;
10    struct sockaddr *sa, *wild;
11    struct ifi_info *ifi, *ifihead;
12
13    if (argc == 2)
14        sockfd = Udp_client(NULL, argv[1], (void **) &sa, &salen);
15    else if (argc == 3)
16        sockfd = Udp_client(argv[1], argv[2], (void **) &sa, &salen);
17    else
18        err_quit("usage: udpserv04 [ <host> ] <service or port>");
19    family = sa->sa_family;
20    port = sock_get_port(sa, salen);
21    Close(sockfd); /* we just want family, port, salen */
22
23    for (ifihead = ifi = Get_ifi_info(family, 1);
24         ifi != NULL; ifi = ifi->ifi_next) {
25
26        /* bind unicast address */
27        sockfd = Socket(family, SOCK_DGRAM, 0);
28        Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
29
30        sock_set_port(ifi->ifi_addr, salen, port);
31        Bind(sockfd, ifi->ifi_addr, salen);
32        printf("bound %s\n", Sock_ntop(ifi->ifi_addr, salen));
33
34        if ((pid = Fork()) == 0) { /* child */
35            mydg_echo(sockfd, ifi->ifi_addr, salen);
36            exit(0); /* never executed */
37        }
38
39        if (ifi->ifi_flags & IFF_BROADCAST) {
40            /* try to bind broadcast address */
41            sockfd = Socket(family, SOCK_DGRAM, 0);
42            Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
43
44            sock_set_port(ifi->ifi_brdaddr, salen, port);
45            if (bind(sockfd, ifi->ifi_brdaddr, salen) < 0) {
46                if (errno == EADDRINUSE) {
47                    printf("EADDRINUSE: %s\n",
48                          Sock_ntop(ifi->ifi_brdaddr, salen));
49                    Close(sockfd);
50                    continue;
51                } else
52                    err_sys("bind error for %s",
53                          Sock_ntop(ifi->ifi_brdaddr, salen));
54            }
55            printf("bound %s\n", Sock_ntop(ifi->ifi_brdaddr, salen));
56
57            if ((pid = Fork()) == 0) { /* child */
58                mydg_echo(sockfd, ifi->ifi_brdaddr, salen);
59                exit(0); /* never executed */
60            }
61        }
62    }
63 }

```

图E-15 (续)

942

```

55     /* bind wildcard address */
56     sockfd = Socket(family, SOCK_DGRAM, 0);
57     Setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
58     wild = Malloc(salen);
59     memcpy(wild, sa, salen); /* copy family and port */
60     sock_set_wild(wild, salen);
61     Bind(sockfd, wild, salen);
62     printf("bound %s\n", Sock_ntop(wild, salen));
63     if ((pid = Fork()) == 0) /* child */
64         mydg_echo(sockfd, wild, salen);
65         exit(0); /* never executed */
66     }
67     exit(0);
68 }

69 void
70 mydg_echo(int sockfd, SA *myaddr, socklen_t salen)
71 {
72     int n;
73     char mesg[MAXLINE];
74     socklen_t len;
75     struct sockaddr *cli;
76     cli = Malloc(salen);
77     for ( ; ; ) {
78         len = salen;
79         n = Recvfrom(sockfd, mesg, MAXLINE, 0, cli, &len);
80         printf("child %d, datagram from %s", getpid(), Sock_ntop(cli, len));
81         printf(", to %s\n", Sock_ntop(myaddr, salen));
82         Sendto(sockfd, mesg, n, 0, cli, len);
83     }
84 }
```

advio/udpserv04.c

图E-15 (续)

第 24 章

24.1 是的。第一个例子中的2个字节是随单个紧急指针发送的，该指针指向的是b后面的字节。第二个例子（两个函数调用）中首先发送的是a以及指向它之后字节的紧急指针，接着以另外一个TCP分节发送的是b和指向它之后字节的另外一个紧急指针。

24.2 图E-16给出了使用poll的版本。

```

1 #include "unp.h"                                         oob/tcprecv03p.c
2 int
3 main(int argc, char **argv)
4 {
5     int listenfd, connfd, n, justreadoob = 0; .
6     char buff[100];
7     struct pollfd pollfd[1];
8     if (argc == 2)
```

图E-16 以poll代替select的图24-6中程序的修改版本

```

9      listenfd = Tcp_listen(NULL, argv[1], NULL);
10     else if (argc == 3)
11         listenfd = Tcp_listen(argv[1], argv[2], NULL);
12     else
13         err_quit("usage: tcprecv03p [ <host> ] <port#>");
14
15     connfd = Accept(listenfd, NULL, NULL);
16
17     pollfd[0].fd = connfd;
18     pollfd[0].events = POLLRDNORM;
19     for ( ; ; ) {
20         if (justreadoob == 0)
21             pollfd[0].events |= POLLRDBAND;
22
23         Poll(pollfd, 1, INFTIM);
24
25         if (pollfd[0].revents & POLLRDBAND) {
26             n = Recv(connfd, buff, sizeof(buff)-1, MSG_OOB);
27             buff[n] = 0; /* null terminate */
28             printf("read %d OOB byte: %s\n", n, buff);
29             justreadoob = 1;
30             pollfd[0].events &= ~POLLRDBAND; /* turn bit off */
31         }
32
33         if (pollfd[0].revents & POLLRDNORM) {
34             if ( (n = Read(connfd, buff, sizeof(buff)-1)) == 0) {
35                 printf("received EOF\n");
36                 exit(0);
37             }
38         }
39     }
40 }
```

943

—oob/tcprecv03p.c

图E-16 (续)

第 25 章

- 25.1 这样的改动引入了一个错误。问题在于`nqueue`是在处理数组元素`dg[iget]`之前递减的，导致信号处理函数有可能把新的数据报从套接字读入到这个数组元素。

第 26 章

- 26.1 使用`fork`的例子将会使用101个描述符，其中1个是监听套接字描述符，其余100个是已连接套接字描述符。不过101个进程（1个父进程，100个子进程）的每一个只打开着一个描述符（忽略任何其他描述符，例如服务器不是守护进程时的标准输入）。然而线程化的服务器是单个进程中101个描述符，每个线程（包括主线程）处理其中一个。
- 26.2 TCP连接终止序列的最后2个分节（服务器的FIN和客户对于该FIN的ACK）将不会交换。这使得连接的客户端一直处于`FIN_WAIT_2`状态（图2-4）。源自Berkeley的实现在客户端保持这种状态超过11分钟时就会超时断连（TCPv2第825~827页）。服务器还可能（最终）耗尽描述符。

944

- 26.3 这个消息应该在主线程已从套接字读入EOF而另一个线程却还在运行时显示。这么做的一个简单方法是声明名为done且初始化为0的另一个外部变量。线程copyto在返回之前把该变量设置成1。主线程检查该变量，如果其值为0就显示这个出错消息。既然设置该变量的线程只有一个，因而没有任何同步的必要。

第 27 章

- 27.1 没有变化，所有系统都是邻居，因此严格的源路径等同于宽松的源路径。
- 27.2 我们会在缓冲区的末尾放一个EOL（值为0的单个字节）。
- 27.3 ping创建的是一个原始套接字（第28章），因此能够获取使用recvfrom读入的每个数据报的完整IP首部，包括任何IP选项在内。
- 27.4 因为rlogind是由inetd激活的（13.5节），而描述符0正是通达客户的套接字。
- 27.5 问题在于setsockopt的第五个参数以指向长度的指针取代长度本身。这个缺陷可能是在开始使用ANSI C原型时修正的。
这个缺陷结果是无害的，因为正如我们所提，禁止IP_OPTIONS套接字选项既可以指定一个空指针作为第四个参数，也可以使用值为0的第五个（长度）参数（TCPv2第269页）。

第 28 章

- 28.1 IPv6首部中的版本字段和下一个首部字段是无法得到的。净荷长度字段或者作为某个输出函数的一个参数，或者作为来自某个输入函数的返回值总是可得到，但是如果需要特大净荷选项，那么真正的选项本身应用进程是得不到的。分片首部应用进程也得不到。
- 28.2 最终客户的套接字接收缓冲区会被填满，导致作为服务器的icmpd守护进程的write调用阻塞。我们不希望发生这种情况，因为它使得icmpd在任何套接字上都停止处理新的数据。最容易的解决办法是让icmpd把它跟客户的Unix域连接的本地端设置成非阻塞式。icmpd然后必须改为调用write以取代它的包裹函数write，并仅仅忽略EWOULDBLOCK错误。
- 28.3 源自Berkeley的内核默认允许在原始套接字上的广播（TCPv2第1057页）。SO_BROADCAST套接字选项只有UDP套接字才需指定。
- 28.4 我们的程序既不检查多播地址，也不设置IP_MULTICAST_IF套接字选项，因此内核可能通过搜索224.0.0.1的路由表项选定外出接口。我们也不设置IP_MULTICAST_TTL套接字选项，因此它默认成1，这是合理的。

945

第 29 章

- 29.1 这个标志表示跳转缓冲区已由sigsetjmp设置（图29-10）。尽管这个标志看似多余，但是在信号处理函数建立之后和调用sigsetjmp之前，SIGALRM信号被递交的机会还是存在的。即使程序本身不会导致产生该信号，它也可能以其他方式产生，譬如使用kill命令。

第 30 章

- 30.1 父进程保持监听套接字打开着是为以后需要fork额外的子进程而做准备（这是对于现行代码的一种改进）。

30.2 是的，数据报套接字能够取代字节流套接字用于传递描述符。在使用数据报套接字情况下，当某个子进程过早终止时，父进程在流管道的拥有端接收不到EOF，不过父进程可以使用SIGCHLD信号达到这个目的。这种能够使用SIGCHLD的情形与28.7节中的icmpd守护进程情形相比的一个差别是：后者的客户和服务器之间不存在父子关系，因此流管道上的EOF是服务器检测某个客户已消失的唯一办法。

第31章

31.1 我们假定流关闭时协议的默认处理就是顺序释放，这对TCP来说是正确的。

946



参 考 文 献

所有RFC都可以通过电子邮件、匿名FTP或WWW免费获取，起始点之一是<http://www.ietf.org>。目录<ftp://ftp.isi.edu/in-notes>是一个存放RFC的位置。这里不给出RFC的URL。

标记为“Internet Draft（因特网草案）”的条目是因特网工程任务攻坚组（Internet Engineering Task Force, IETF）正在进展中的著作。这些草案在出版后6个月就完成使命。因此它们或者在本书付印之后就有新版本出现，或者已经作为RFC出版。它们跟RFC一样，也可以免费从因特网上获取。访问因特网草案的起始点之一也是<http://www.ietf.org>。这里给出每个因特网草案的URL中的文件名部分，因为文件名含有版本号。

要是能够找到本参考文献所引用的论文或报告的电子文件，其URL就同时列出。需留意的是这些URL可能随时间而变动，因此读者应该经常访问本书的主页（<http://www.unpbook.com/>），检查本书的最新勘误表。<http://citeseer.nj.nec.com/cs>是一个相当棒的论文在线数据库。通过这个数据库输入一篇论文或报告的标题不仅可以找到引用它的其他论文，还能给出已知的在线版本。

- Albitz, P. and Liu, C. 2001, *DNS and Bind, Fourth Edition*. O'Reilly & Associates, Sebastopol, CA.
Allman, M., Floyd, S., and Partridge, C. 2002. "Increasing TCP's Initial Window," RFC 3390.
Allman, M., Ostermann, S., and Metz, C.W. 1998. "FTP Extensions for IPv6 and NATs," RFC 2428.
Allman, M., Paxson, V., and Stevens, W.R. 1999. "TCP Congestion Control," RFC 2581.
Almquist, P. 1992. "Type of Service in the Internet Protocol Suite," RFC 1349 (obsoleted by RFC 2474).

如何使用IPv4首部的服务类型字段。已被RFC 2474 [Nicholset al. 1998] 和RFC 3168 [Ramakrishnan, Floyd, and Black 2001] 淘汰。

- Baker, F. 1995. "Requirements for IP Version 4 Routers," RFC 1812.
Borman, D.A. 1997a. "Re: Frequency of RST Terminated Connections," end2end-interest mailing list (<http://www.unpbook.com/borman.97jan30.txt>).
Borman, D.A. 1997b. "Re: SYN/RST cookies," tcp-impl mailing list (<http://www.unpbook.com/borman.97jun06.txt>).
Borman, D.A. Deering, S.E., and Hinden,R. 1999. "IPv6 Jumbograms," RFC 2675.
Braden, R.T., 1989. "Requirements for Internet Hosts-Communication Layers," RFC 1122.

主机要求RFC（Host Requirements RFC）的前半部分。这部分的内容包括链路层、IPv4、ICMPv4、IGMPv4、ARP、TCP及UDP。

- Braden, R.T. 1992. "TIME-WAIT Assassination Hazards in TCP", RFC 1337.
Braden, R.T. Borman, D.A., and Partridge, C. 1988. "Computing the Internet Checksum," RFC 1071.
Bradner, S. 1996. "The Internet Standards Process-Revision3," RFC 2026.
Bush, R. 2001. "Delegation of IP6.ARPA," RFC 3152.

- Butenhof, D.R. 1997. *Programming with POSIX Threads*. Addison-Wesley, Reading, MA.
- Cain, B., Deering, S.E., Kouvelas, I., Fenner, B., and Thyagarajan, A. 2002. "Internet Group Management Protocol, Version 3," RFC 3376.
- Carpenter, B. and Moore, K. 2001. "Connection of IPv6 Domains via IPv4 Clouds," RFC 3056.
- CERT. 1996a. "UDP Port Denial-of-Service Attack," Advisory CA-96.01, Computer Emergency Response Team, Pittsburgh, PA.
- CERT. 1996b. "TCP SYN Flooding and IP Spoofing Attacks," Advisory CA-96.21, Computer Emergency Response Team, Pittsburgh, PA.
- Cheswick, W.R., Bellovin, S.M. and Rubin, A.D. 2003. *Firewalls and Internet Security: Repelling the Wily Hacker, Second Edition*. Addison-Wesley, Reading, MA.
- Conta, A., and Deering, S.E. 1998. "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 2463.
- Conta, A., and Deering, S.E. 2001. "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," draft-ietf-ipngwg-icmp-v3-02.txt (Internet Draft).
- 这是对 [Conta and Deering 1998] 的修订版本，期望最终取代它。
- Crawford, M. 1999a. "Transmission of IPv6 Packets over Ethernet Networks," RFC 2464.
- Crawford, M. 1998b. "Transmission of IPv6 Packets over FDDI Networks," RFC 2467.
- Crawford, M., Narten, T., and Thomas, S. 1998. "Transmission of IPv6 Packets over Token Ring Networks," RFC 2470.
- Deering, S.E. 1989. "Host Extensions for IP Multicasting," RFC 1112.
- Deering, S.E. and Hinden, R. 1998. "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460.
- Dewan, R.B.K., and Sosna, M. 1990. *Microprocessors: A Programmers View*. McGraw-Hill, NY.
- Draves, R. 2003. "Default Address Selection for Internet Protocol version 6 (IPv6)," RFC 3484.
- Eriksson, H. 1994. "MBONE: The Multicast Backbone," *Communications of the ACM*, vol.37, no.8, pp.54-60.
- Fenner, W.C. 1997. Private Communication.
- Fink, R. and Hinden, R. 2003. "6bone (IPv6 Testing Address Allocation) Phaseout," draft-fink-6bone-phaseout-04.txt (Internet Draft).
- Fuller, V., Li, T., Yu, J.Y., and Varadhan, K. 1993. "Classless Inter-Domain Routing (CIDR):an Address Assignment and Aggregation Strategy," RFC 1519.
- Garfinkel, S.L., Schwartz, A., and Spafford, E.H. 2003. *Practical UNIX & Internet Security, 3rd Edition*. O'Reilly & Associates, Sebastopol, CA.
- Gettys, J. and Nielsen, H.F. 1998. *SMUX Protocol Specification* (<http://www.w3.org/TR/WD-mux>).
- Gierth, A. 1996. Private Communication.
- Gilligan, R.E. and Nordmark, E. 2000. "Transition Mechanisms for IPv6 Hosts & Routers," RFC 2893.
- Gilligan, R.E., Thomson, S., Bound, J., McCann, J., and Stevens, W.R. 2003. "Basic Socket Interface Extensions for IPv6," RFC 3493.
- Gilligan, R.E., Thomson, S., Bound, J., and Stevens, W.R. 1997. "Basic Socket Interface Extensions for IPv6," RFC 2133(obsoleted by RFC 2553).
- Gilligan, R.E., Thomson, S., Bound, J., and Stevens, W.R. 1999. "Basic Socket Interface Extensions for

- IPv6,” RFC 2553(obsoleted by RFC 3493).
- Haberman, B. 2002. “Allocation Guidelines for IPv6 Multicast Addresses,” RFC 3307.
- Haberman, B. and Thaler, D. 2002. “Unicast-Prefix-based IPv6 Multicast Addresses,” RFC 3306.
- Handley, M. and Jacobson, V. 1998. “SDP: Session Description Protocol,” RFC 2327.
- Handley, M., Perkins, C., and Whelan, E. 2000. “Session Announcement Protocol,” RFC 2974.
- Harkins, D. and Carrel, D. 1998. “The Internet Key Exchange (IKE),” RFC 2409.
- Hinden, R. and Deering, S.E. 2003. “Internet Protocol Version 6 (IPv6) Addressing Architecture,” RFC 3513.
- Hinden, R., Deering, S.E., and Nordmark, E. 2003. “IP6 Global Unicast Address Format,” RFC 3587.
- Hinden, R., Fink, R., and Postel, J.B. 1998. “IPv6 Testing Address Allocation,” RFC 2471.
- Holbrook, H. and Cheriton, D. 1999. “IP multicast channels: EXPRESS support for large-scale single-source applications,” *Computer Communication Review*, vol.29, no.4, pp.65-78.
- Huitema, C. 2001. “An Anycast Prefix for 6to4 Relay Routers,” RFC 3068.
- IANA. 2003. *Protocol/Number Assignments Directory* (<http://www.iana.org/numbers.htm>).
- IEEE. 1996. “Information Technology-Portable Operationg System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language],” IEEE Std 1003.1, 1996 Edition, Institute of Electrical and Electronics Enginerrs, Piscataway, NJ.

这个版本的POSIX.1含有1990基本API、1003.1b实时扩展(1993)、1003.1c pthreads(1995)以及1003-li技术性更正(1995)。它同时也是国际标准ISO/IEC 9945-1: 1996(E)。IEEE正式标准和草案标准的订购信息可从<http://www.ieee.org>获取。

- IEEE. 1997. *Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority*. Institute of Electrical and Electronics Engineers, Piscataway, NJ. (<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>).
- Jacobson, V. 1988. “Congestion Avoidance and Control,” *Computer Communication Review*, vol.18, no.4, pp.314-329(<ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z>).

描述TCP的慢启动和拥塞避免算法的经典论文。

- Jacobson, V., Braden, R.T., and Borman, D.A. 1992. “TCP Extensions for High Performance,” RFC 1323.

描述窗口规模选项、时间戳选项、PAWS算法以及添加它们的理由。[Braden 1993] 是对它的更新。

- Jacobson, V., Braden, R.T., and Zhang, L. 1990. “TCP Extensions for High-Speed Paths,” RFC 1185 (obsoleted by RFC 1323).
- Josey, A., ed. 1997. *Go Solo 2: The Authorized Guide to Version 2 of the Single UNIX Specification*. Prentice Hall, Upper Saddle River, NJ.
- Josey, A., ed. 2002. *The Single UNIX Specification-The Authorized Guide to Version 3*. The Open Group, Berkshire, UK.
- Joy, W.N. 1994. *Private Communication*.
- Karn, P., and Partridge, C. 1991. “Improving Round-Trip Time Estimates in Reliable Transport Protocols,” *ACM Transactions on Computer Systems*, vol.9, no.4, pp.364-373.

- Katz, D. 1993. "Transmission of IP and ARP over FDDI Network," RFC 1390.
- Katz, D. 1997. "IP Router Alert Option," RFC 2113.
- Kent, S.T. 1991. "U.S. Department of Defense Security Options for the Internet Protocol," RFC 1108.
- Kent, S.T. 2003a. "IP Authentication Header," draft-ietf-ipsec-rfc2402bis-04.txt (Internet Draft).
- Kent, S.T. 2003b. "IP Encapsulating Security Payload (ESP)," draft-ietf-ipsec-esp-v3-06.txt (Internet Draft).
- Kent, S.T. and Atkinson, R.J. 1998a. "Security Architecture for the Internet Protocol," RFC 2401.
- Kent, S.T. and Atkinson, R.J. 1998b. "IP Authentication Header," RFC 2402.
- 本书写至此处时IETF IPsec工作组正在更新本RFC(见 [Kent 2003a]).
- Kent, S.T. and Atkinson, R.J. 1998c. "IP Encapsulating Security Payload (ESP)," RFC 2406.
- 本书写至此处时IETF IPsec工作组正在更新本RFC(见 [Kent 2003b]).
- Kernighan, B.W. and Pike, R. 1984. *The UNIX Programming Environment*. Prentice Hall, Englewood Cliffs, NJ.
- Kernighan, B.W. and Ritchie, D.M. 1988. *The C Programming Language, Second Edition*. Prentice Hall, Englewood Cliffs, NJ.
- Lanciani, D. 1996. "Re: sockets: AF_INET vs. PF_INET," Message-ID: <3561@news. IPSWI-TCH.COM>, USENET comp.protocols.tcp-ip Newsgroup (<http://www.unpbook.com/lanciani.96apr10.txt>).
- Maslen, T.M. 1997. "Re: gethostbyXXXX() and Threads," Message-ID: <maslen.862463530@shellx>, USENET comp.programming.threads Newsgroup (<http://www.unpbook.com/maslen.97may01.txt>).
- McCann, J., Deering, S.E., and Mogul, J.C. 1996. "Path MTU Discovery for IP version 6," RFC 1981.
- McCanne, S. and Jacobson, V. 1993. "The BSD Packet Filter: A New Architecture for User-Level Packet Capture," *Proceedings of the 1993 Winter USENIX Conference*, San Diego. CA, pp.259-269.
- McDonald, D.L., Metz, C.W., and Phan, B.G. 1998. "PF_KEY Key Management API, Version 2," RFC 2367.
- McKusick, M.K., Bostic, K., Karels, M.J., and Quarterman, J.S. 1996. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley, Reading, MA.
- Meyer, D. 1998. "Administratively Scoped IP Multicast," RFC 2365.
- Mills, D.L. 1992. "Network Time Protocol(Version 3) Specification, Implementation," RFC 1305.
- Mills, D.L. 1996. "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," RFC 2030.
- Mogul, J.C., and Deering, S.E. 1990. "Path MTU Discovery," RFC 1191.
- Mogul, J.C., and Postel, J.B. 1985. "Internet Standard Subnetting Procedure," RFC 950.
- Narten, T. and Draves, R. 2001. "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 3041.
- Nemeth, E. 1997. *Private Communication*.
- Nichols, K., Blake, S., Baker, F., and Black, D. 1998. "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474.

- Nordmark, E. 2000. "Stateless IP/ICMP Translation Algorithm (SITT)," RFC 2765.
- Ong, L., Rytina, I., Garcia, M., Schwarzbauer, H., Coene, L., Lin, H., Juhasz, I., Holdrege, M. and Sharp, C. 1999. "Framework Architecture for Signaling Transport," RFC 2719.
- Ong, L., and Yoakum, J. 2002. "An Introduction to the Stream Control Transmission Protocol (SCTP)," RFC 3286.
- The Open Group. 1997. *CAE Specification, Networking Services(XNS), Issue 5*. The Open Group, Berkshire, UK.

这是Unix 98中套接字和XTI这两个API的规范，现已被The Single Unix Specification, Version 3超越。本手册还在其附录中描述了XTI在NetBIOS、OSI协议族、SNA及Netware的IPX和SPX协议中的使用。另有三个附录介绍套接字和XTI这两个API在ATM中的使用。

- Partridge, C. and Jackson, A. 1999. "IPv6 Router Alert Option," RFC 2711.
- Partridge, C., Mendez, T., and Milliken, W. 1993. "Host Anycasting Service," RFC 1546.
- Partridge, C. and Pink, S. 1993. "A Faster UDP," *IEEE/ACM Transactions on Networking*, vol.1, no.4, pp.429-440.
- Paxson, V. 1996. "End-to-End Routing Behavior in the Internet," *Computer Communication Review*, vol.26, no.4, pp. 25-38 (<ftp://ftp.ee.lbl.gov/papers/routing.SIGCOMM.ps.Z>).
- Paxson, V. and Allman, M. 2000. "Computing TCP's Retransmission Timer," RFC 2988.
- Plauger, P.J. 1992. *The Standard C Library*. Prentice Hall, Englewood Cliffs, NJ.
- Postel, J.B. 1980. "User Datagram Protocol," RFC 768.
- Postel, J.B. 1981a. "Internet Protocol," RFC 791.
- Postel, J.B. 1981b. "Internet Control Message Protocol," RC 792.
- Postel, J.B. 1981c. "Transmission Control Protocol," RFC 793.
- Pusateri, T. 1993. "IP Multicast Over Token-Ring Local Area Networks," RFC 1469.
- Rago.S.A. 1993. *UNIX System V Network Programming*. Addison-Wesley, Reading, MA.
- Rajahalme, J., Conta, A., Carpenter, B., and Deering, S.E. 2003. "IPv6 Flow Label Specification," <draft-ietf-ipv6-flow-label-07.txt> (Internet Draft).
- Ramakrishnan, K., Floyd, S., and Black, D. 2001. "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168.
- Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and Lear, E. 1996. "Address Allocation for Private Internets," RFC 1918.
- Reynolds, J.K. 2002. "Assigned Numbers: RFC 1700 is Replaced by an On-line Database," RFC 3232.

本RFC指的数据就是 [IANA 2003] .

- Reynolds, J.K., and Postel, J.B. 1994. "Assigned Numbers," RFC 1700 (obsoleted by RFC 3232).

本RFC是 "Assigned Numbers" 系列RFC中的最后一个。由于其信息经常变化，因此整个目录都被放在因特网上。可参阅[Reynolds 2002]以获取更多解释，或是参考[IANA 2003]来了解数据库本身。

- Ritchie, D.M. 1984. "A Stream Input-Output System," *AT&T Bell Laboratories Technical Journal*, vol.63, no. 8, pp.1897-1910.

- Salus, P.H. 1994. *A Quarter Century of Unix*. Addison-Wesley, Reading, MA.
- Salus, P.H. 1995. *Casting the Net: From ARPANET to Internet and Beyond*. Addison-Wesley, Reading, MA.
- Schimmel, C. 1994. *UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers*. Addison-Wesley, Reading, MA.
- Spero, S. 1996. *Session Control Protocol (SCP)* (<http://www.w3.org/Protocols/HTTP-NG/http-ng-scp.html>).
- Srinivasan, R. 1995. "XDR: External Data Representation Standard," RFC 1832.
- Stevens, W.R. 1992. *Advanced Programming in the UNIX Environment*. Addison-Wesley, Reading, MA.

Unix编程的所有细节。本书称之为APUE。

- Stevens, W.R. 1994. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, MA.

对网际网协议的完整介绍。本书称之为TCPv1。

- Stevens, W.R. 1996. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*. Addison-Wesley, Reading, MA.

本书称之为TCPv3。

- Stevens, W.R. and Thomas, M. 1998. "Advanced Sockets API for IPv6," RFC 2292 (obsoleted by RFC 3542).

- Stevens, W.R., Thomas, M., Nordmark, E., and Jinmei, T. 2003. "Advanced Sockets Application Program Interface (API) for IPv6," RFC 3542.

- Stewart, R.R., Bestler, C., Jim, J., Ganguly, S., Shah, H., and Kashyap, V. 2003a. "Stream Control Transmission Protocol (SCTP) Remote Direct Memory Access (RDMA) Direct Data Placement (DDP) Adaptation," draft-stewart-rddp-sctp-02.txt (Internet Draft).

- Stewart, R.R., Ramalho, M., Xie, Q., Tuexen, M., Rytina, I., Belinchon, M., and Conrad, P. 2003b. "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration," draft-ietf-tsvwg-addip-sctp-07.txt (Internet Draft).

- Stewart, R.R. and Xie, Q. 2001. *Stream Control Transmission Protocol (SCTP): A Reference Guide*. Addison-Wesley, Reading, MA.

- Stewart, R.R., Xie, Q., Morneau, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and Paxson, V. 2000. "Stream Control Transmission Protocol," RFC 2960.

- Stone, J., Stewart, R.R., and Otis, D. 2002. "Stream Control Transmission Protocol (SCTP) Checksum Change," RFC 3309.

- Tanenbaum, A.S. 1987. *Operating Systems Design and Implementation*. Prentice Hall, Englewood Cliffs, NJ.

- Thomson, S. and Huitema, C. 1995. "DNS Extensions to Support IP version 6," RFC 1886.

- Torek, C. 1994. "Re: Delay in re-using TCP/IP port," Message-ID: <199501010028.QAA16863@elf.bsd.com>, USENET comp.unix.wizards Newsgroup (<http://www.umpbook.com/torek.94dec31.txt>).

- Touch, J. 1997. "TCP Control Block Interdependence," RFC2140.

Unix International. 1991. *Data Link Provider Interface Specification*, Unix International, Parsippany, NJ Revision 2.0.0. (<http://www.unpbook.com/dlpi.2.0.0.ps>).

本规范的较新版本可从Open Group的网页<http://www.rdg.opengroup.org/pubs/catalog/web.htm>在线获取。

Unix International. 1992a. *Network Provider Interface Specification*. Unix International, Parsippany, NJ, Revision 2.0.0 (<http://www.unpbook.com/npi.2.0.0.ps>).

Unix International. 1992b. *Transport Provider Interface Specification*. Unix International, Parsippany, NJ, Revision 1.5 (<http://www.unpbook.com/tpi.1.5.ps>).

本规范的较新版本可从Open Group的网页<http://www.rdg.opengroup.org/pubs/catalog/web.htm>在线获取。

Vixie, P.A. 1996. *Private Communication*.

Wright, G.R. and Stevens, W.R. 1995. *TCP/IP Illustrated, Volume 2: The Implementation*. AddisonWesley, Reading, MA.

网际网协议在4.4BSD-Lite操作系统上的实现。本书称之为TCPv2。



索引

网络编程是一个密布首字母缩写词的领域。我们不提供一个单独的词汇表（其中大多数条目将是首字母缩写词），不过本索引也可用作本书所用所有首字母缩写词的词汇表。可以首字母缩写的词条其主条目编排在缩写词之下。举例来说，所有对Internet Control Message Protocol（网际网控制消息协议）的引用出现在ICMP之下，在完整词条“Internet Control Message Protocol”之下的条目只是引用回ICMP之下的主条目。

每个C函数的“definition of(定义)”条目给出该函数带方框的函数原型即基本描述的所在页。每个结构的“definition of(定义)”条目给出该结构的基本定义的所在页。那些在本书中有源代码实现的函数还有“source code(源代码)”条目。

索引中的页码为英文原书页码，与书中页边标注的页码一致。

- 4.1cBSD, 98
- 4.2BSD, 20–21, 70, 79–80, 98–100, 106, 166, 390, 412, 536–537, 589
- 4.3BSD, 21, 51, 261, 371–372, 536
 Renø, 21, 68, 74, 210, 388–389, 485, 588, 711, 737, 755
 Tahoe, 21
- 4.4BSD, 21, 27, 34, 74, 76, 98–100, 103, 134, 166, 208, 212, 215, 253, 420, 466, 477, 486, 494, 737, 788, 829–832
- 4.4BSD-Lite, 20–21, 954
- 4.4BSD-Lite2, 20–21, 926
- 64-bit alignment (64位对齐), 72, 873
- 64-bit architectures (64位体系结构), 28–29, 79, 152, 918
- 6bone (IPv6 backbone), 887–889
- test address (测试地址), 879
6to4, 889
- Abell, V. A., 897
- absolute name (绝对名字), DNS, 303
- absolute time (绝对时间), 704
- accept function (accept函数), 14–15, 37–38, 63, 68, 75, 104–105, 107–112, 114–118, 120, 122, 126–127, 133–135, 138–140, 147, 165, 176, 180, 198, 208, 241, 251, 263, 269, 278, 307, 320, 330–331, 333, 340, 355–356, 359–360, 373, 375, 377–379, 421, 432, 436, 461–463, 649, 656, 658, 675, 681, 683–684, 710, 717, 777, 818, 826, 829–834, 836–838, 841–842, 844–847, 850, 915, 923, 933–934, 938
- connection abort (连接中止), 139–141
- definition of (accept函数定义), 109
- nonblocking (非阻塞accept函数), 461–463
- ACK (acknowledgment flag, TCP header), 38–39, 44, 58
 delayed (延滞ACK), 220, 237, 923
- acknowledgment flag, TCP header (确认标志, TCP首部),
 见ACK
- active (主动)
- close (主动关闭), 39–41, 43–44, 47–48, 62, 914, 916, 921
- open (主动打开), 37–38, 41, 45, 48, 53, 894
- socket (主动套接字), 104
- addr member (addr成员), 862
- ADDR_length member (ADDR_length成员), 860, 862
- ADDR_offset member (ADDR_offset成员), 860, 862
- address (地址)
- 6bone test (6bone测试地址), 879
- administratively scoped IPv4 multicast (管理上划分范围的多播地址), 553
- alias (别名地址), 103, 877
- broadcast (广播地址), 531–532
- classless (无类地址), 874–875
- ethernet mapping, picture of, IPv4 multicast (IPv4多播地址到以太网地址的映射图示), 550
- ethernet mapping, picture of, IPv6 multicast (IPv6多播地址到以太网地址的映射图示), 550
- global unicast (全球单播地址), 878–879
- IPv4 (IPv4地址), 874–877
- IPv4 destination (IPv4目的地址), 871
- IPv4 multicast (IPv4多播地址), 549–551
- IPv4 source (IPv4源地址), 871
- IPv4-compatible IPv6 (IPv4兼容的IPv6地址), 880
- IPv4-mapped IPv6 (IPv4映射的IPv6地址), 93, 322, 333, 354–360, 745, 879–880
- IPv6 (IPv6地址), 877–881
- IPv6 destination (IPv6目的地址), 873
- IPv6 multicast (IPv6多播地址), 551–552
- IPv6 source (IPv6源地址), 873
- link-local (链路局部地址), 881
- loopback (环回地址), 111, 365, 432, 876, 880
- multicast (多播地址), 549–553

- multicast group (多播组地址), 549
 picture of, IPv6 multicast (IPv6多播地址图示), 551
 private (私用地址), 876
 site-local (网点局部地址), 881
 subnet (子网地址), 875–876, 951
 unspecified (未指定地址), 876, 881
 well-known (众所周知的地址), 52
 wildcard (通配地址), 53, 87, 102, 122, 126, 147, 211,
 322, 354–355, 357, 362, 373, 560, 562, 568, 581–582,
 608, 610–611, 772, 779, 876, 881
 address request, ICMP (ICMP地址掩码请求), 739, 883
 Address Resolution Protocol (地址解析协议), 见ARP
 addrinfo structure (addrinfo结构), 99, 315–317, 319,
 321, 323–324, 330, 457, 745, 757
 definition of (addrinfo结构定义), 315
 administratively scoped IPv4 multicast address (管理上划分
 范围的IPv4多播地址), 553
 admin-local multicast scope (管理局部多播范围), 552
 Advanced Programming in the UNIX
 Environment (UNIX环境高级编程), 见APUE
 AF_ versus PF_ (对比AF_和PF_), 98–99
 AF_INET constant (AF_INET常值), 7–8, 10, 72–73, 83,
 86, 93, 97, 244, 310, 322, 361, 497, 745, 775
 AF_INET6 constant (AF_INET6常值), 32, 72–73, 83, 93,
 97, 227, 322, 497, 745, 775, 941
 AF_ISO constant (AF_ISO常值), 98
 AF_KEY constant (AF_KEY常值), 97–98, 511
 AF_LINK constant (AF_LINK常值), 73, 497, 502, 591
 AF_LOCAL constant (AF_LOCAL常值), 73, 97–98, 412,
 414, 416, 418–419
 AF_NS constant (AF_NS常值), 98
 AF_ROUTE constant (AF_ROUTE常值), 97–98, 213, 465,
 485–486, 492, 495, 497
 AF_UNIX constant (AF_UNIX常值), 98, 412
 AF_UNSPEC constant (AF_UNSPEC常值), 254, 316, 322,
 327, 329–330, 332, 339, 482, 497
 AH (authentication header), 719, 951
 AI_CANONNAME constant (AI_CANONNAME常值), 317,
 324
 AI_PASSIVE constant (AI_PASSIVE常值), 320, 322,
 324–325, 330, 620, 941
 ai_addr member (ai_addr成员), 315, 317, 321
 ai_addrlen member (ai_addrlen成员), 315, 317,
 320
 ai_canonname member (ai_canonname成员), 315,
 317, 321
 ai_family member (ai_family成员), 315–317, 322
 ai_flags member (ai_flags成员), 315–316, 322
 ai_next member (ai_next成员), 315–316
 ai_protocol member (ai_protocol成员), 315–317,
 319
 ai_socktype member (ai_socktype成员), 315–317,
 319–320
 aio_read function (aio_read函数), 159
 AIX, 22, 78, 108, 257, 262, 306, 486, 538
 alarm function (alarm函数), 381, 383–384, 409, 432,
 539, 541, 547, 603–604, 607, 620, 803
 Albitz, P., 304, 349, 947
 alias address (别名地址), 103, 877
 alignment (对齐), 150, 714, 721
 64-bit (64位对齐), 72, 873
 all-hosts multicast group (所有主机多播组), 550
 Allman, E., 315
 M., 35, 208, 360, 947–948, 952
 all-nodes multicast group (所有节点多播组), 552
 all-routers multicast group (所有路由器多播组), 550, 552
 Almquist, P., 215, 870, 948
 American National Standards Institute (美国国家标准局),
 见ANSI
 American Standard Code for Information
 Interchange (美国标准信息互换代码), 见ASCII
 ancillary data (辅助数据), 395–398
 object, definition of (辅助数据对象定义), 396
 picture of, IP_RECVDSTADDR (IP_RECVDSTADDR辅助数
 据图示), 394
 picture of, IP_RECVVIF (IP_RECVVIF辅助数据图示), 591
 picture of, IPV6_DSTOPTS (IPV6_DSTOPTS辅助数据图
 示), 722
 picture of, IPV6_HOPLIMIT (IPV6_HOPLIMIT辅助数据
 图示), 615
 picture of, IPV6_HOPOPTS (IPV6_HOPOPTS辅助数据图
 示), 722
 picture of, IPV6_NEXTHOP (IPV6_NEXTHOP辅助数据图
 示), 615
 picture of, IPV6_PKTINFO (IPV6_PKTINFO辅助数据图
 示), 615
 picture of, IPV6_RTHDR (IPV6_RTHDR辅助数据图示),
 727
 picture of, IPV6_TCLASS (IPV6_TCLASS辅助数据图示),
 615
 picture of, SCM_CREDS (SCM_CREDS辅助数据图示), 397
 picture of, SCM_RIGHTS (SCM_RIGHTS辅助数据图示),
 397
 ANSI (American National Standards Institute), 7
 C (ANSI C), 7–9, 29, 70–71, 80–81, 399, 466, 681, 683,
 685, 774, 910, 945
 anycasting (任播), 529, 952
 Apache Web server (Apache Web服务器), 834
 API (application program interface), 6
 sockets (套接字API), 8
 application
 ACK (应用ACK), 206
 protocol (应用协议), 4, 421
 APUE (Advanced Programming in the UNIX
 Environment), 953
 argc variable (argc变量), 370
 argument passing, thread (线程参数传递), 682–685
 ARP (Address Resolution Protocol), 34, 100, 234, 249, 467,
 481, 497–498, 530, 532, 740, 794
 cache operations, ioctl function (ARP高速缓存操作的
 ioctl函数), 481–483
 arp program (arp程序), 482

- arp_flags member (arp_flags成员), 481
 arp_ha member (arp_ha成员), 481–482
 arp_pa member (arp_pa成员), 481–482
 arpreq structure (arpreq结构), 467, 481
 definition of (arpreq结构定义), 481
ASCII (American Standard Code for Information Interchange), 8–9, 82–83, 110, 304, 916
 asctime function (asctime函数), 685
 asctime_r function (asctime_r函数), 685
asynchronous (异步)
 error (异步错误), 240, 249, 252–253, 769–786
 I/O (异步I/O), 160, 468, 663
 I/O model (异步I/O模型), 158–159
Asynchronous Transfer Mode (异步传送模式), 见ATM
at program (at程序), 364
ATF_COM constant (ATF_COM常值), 481–482
ATF_INUSE constant (ATF_INUSE常值), 481–482
ATF_PERM constant (ATF_PERM常值), 481–482
ATF_PUBL constant (ATF_PUBL常值), 481–482
Atkinson, R. J., 511, 719, 951
ATM (Asynchronous Transfer Mode), 952
atoi function (atoi函数), 427
attack, denial-of-service(拒绝服务攻击), 46, 108, 180, 463, 934
audio/video profile (音频/视频定制文件), 见AVP
authentication header (认证首部), 见AH
autoconf program (autoconf程序), 78, 904
automatic tunnel (自动形成的隧道), 880
AVP (audio/video profile), 575
awk program (awk程序), xxiii, 26

backoff, exponential (指数回退), 598, 802
Baker, F., 215, 772, 870–871, 948, 952
bandwidth-delay product (带宽-延迟积), 209
basename program (basename程序), 26
bash program (bash程序), 127, 143
batch input (批量输入), 169–172
Belinchon, M., 285, 953
Bellovin, S. M., 108, 711, 948
Berkeley Internet Name Domain (Berkeley因特网名字域), 见BIND
Berkeley Software Distribution (Berkeley软件发布), 见BSD
Berkeley-derived implementation, definition of (源自Berkeley之实现的定义), 20
Bestler, C., 285, 953
BGP (Border Gateway Protocol, routing protocol), 62
bibliography (参考文献), 947–954
big picture, TCP/IP (TCP/IP总图), 32–34
big-endian byte order (大端字节序), 77
BIND (Berkeley Internet Name Domain), 305–306, 341–342, 498
**bind function (bind函数), 13, 29, 37–38, 45, 52–53, 68, 70–71, 74, 76, 99, 101–104, 109, 111, 118, 120, 126, 140, 146–147, 178, 203, 210–213, 236–237, 242, 245, 248, 250, 252, 254, 261–262, 265, 317, 320, 330, 355, 361–362, 371, 373, 379, 413, 415–416, 419–420, 432–433, 564, 572, 576–577, 581, 585, 609–610, 613, 616, 630, 736, 739, 759, 769, 772, 777, 779, 792, 876, 881, 915, 921, 933, 935, 940
 definition of (bind函数定义), 101**
bind_ack structure (bind_ack结构), 862
bind_connect_listen function (bind_connect_listen函数), 213
bind_req structure (bind_req结构), 860
binding interface address (捆绑接口地址), UDP, 608–612
Black, D., 215, 870–871, 948, 952
black magic (神奇操作), 420
Blake, S., 215, 870–871, 948, 952
blocking, head of line (头端阻塞), 31, 293–299
blocking I/O model (阻塞式I/O模型), 154–155
BOOTP (Bootstrap Protocol), 57, 62, 532
Bootstrap Protocol (引导协议), 见BOOTP
Border Gateway Protocol (边界网关协议), routing protocol (路由协议), 见BGP
Borman, D. A., 35, 38–39, 51, 57, 106, 108, 599, 721, 753, 948, 950
Bostic, K., 20, 737, 951
Bound, J., 28, 71, 216, 346–347, 361, 504, 949
boundaries, message (消息边界), 31
Bourne shell, 26
BPF (BSD Packet Filter), 32, 34, 98, 787–790, 793, 810
Braden, R. T., 35, 38–39, 43–44, 203, 237, 247, 532, 576, 589, 599, 753, 877, 948, 950
Bradner, S., 28, 948
broadcast (广播), 199, 529–547
 address (地址), 531–532
 flooding (广播泛滥), 558
 IP fragmentation and (IP分片与广播), 537–538
 multicast versus (多播与广播对比), 553–556
 storm (广播风暴), 534
 versus unicast (广播与单播对比), 532–535
BSD (Berkeley Software Distribution), 20
 networking history (BSD网络支持历史), 20–21
 Packet Filter (BSD分组过滤器), 见BPF
BSD/OS, 20–21, 98, 108, 167, 392, 808
buf member (buf成员), 856
buffer sizes (缓冲区大小), 55–61
buffering, double (双缓冲), 789
BUFFSIZE constant, definition of (BUFFSIZE常值定义), 902
BUFSIZE constant (BUFSIZE常值), 491
bufmod STREAMS module (bufmod流模块), 790
Bush, R., 304, 948
Butenhof, D. R., 676, 948
byte manipulation functions (字节操纵函数), 80–81
byte order (字节序)
 big-endian (大端字节序), 77
 functions (字节序函数), 77–80
 host (主机字节序), 77, 103, 110, 120, 148, 737, 740, 915

- little-endian (小端字节序), 77
 network (网络字节序), 69, 79, 82, 110, 152, 311–312,
 319, 737–738, 740, 918
 byte-stream protocol (字节流协议), 9, 31, 34, 93, 98, 392,
 415, 435, 661
- C standard, ISO C99 (ISO C99 C标准), 15
 Cain, B., 564, 948
 calloc function (calloc函数), 478, 693
 canonical name record (规范名字记录), DNS, 见CNAME
 caplen member (caplen成员), 809
 Carpenter, B., 871, 889, 948, 952
 Carrel, D., 524, 949
 carriage return (回车符), 见CR
 CDE (Common Desktop Environment), 27
 CERT (Computer Emergency Response Team), 108, 934,
 948
 CFG, 790
 chargen port (字符生成服务端口), 61, 189, 349, 380, 930,
 934
 check_notification function (check_notification
 函数), 633
 checksum (校验和), 948
 ICMPv4 (ICMPv4校验和), 737, 753, 806, 882
 ICMPv6 (ICMPv6校验和), 738, 753–754, 882
 IGMP (IGMP校验和), 753
 IPv4 (IPv4校验和), 214, 737, 753, 871
 IPv6 (IPv6校验和), 216, 738, 873
 TCP (TCP校验和), 753
 UDP (UDP校验和), 259, 497–499, 753, 793–814
 Cheriton, D., 558, 950
 Cheswick, W. R., 108, 711, 948
 Child structure (Child结构), 837–838, 842
 child_main function (child_main函数), 827,
 830–831, 833, 835, 841
 child_make function (child_make函数), 827, 833,
 835, 837
 child.h header (child.h头文件), 837
 CIDR (classless interdomain routing), 874–875
 classless address (无类地址), 874–875
 classless interdomain routing (无类域间路由), 见CIDR
 cleanup function (cleanup函数), 799, 810
 client structure (client结构), 775, 777–780, 783
 client/server (客户/服务器)
 design alternatives (客户/服务器供选择的设计范式),
 817–850
 examples road map (客户/服务器例子导读图), 16–18
 clock resolution (时钟分辨率), 162
 clock_gettime function (clock_gettime函数),
 705
 close (关闭)
 active (主动关闭), 39–41, 43–44, 47–48, 62, 914, 916,
 921
 passive (被动关闭), 39–41, 47–48
- simultaneous (同时关闭), 40–41, 48
 close function (close函数), 12, 15, 37, 39–40, 47, 63,
 101, 114–115, 117, 120, 137, 172–173, 189, 202–206, 236,
 279, 343–344, 446, 462, 464, 681, 707, 780, 868, 915,
 919, 938
 definition of (close函数定义), 117
 CLOSE_WAIT state (CLOSE_WAIT状态), 41
 CLOSED state (CLOSED状态), 40–41, 47–48, 63, 101, 104,
 207
 closefrom function (closefrom函数), 369
 closelog function (closelog函数), 365–367
 definition of (closelog函数定义), 367
 CLOSING state (CLOSING状态), 41
 cmcred_euid member (cmcred_euid成员), 429
 cmcred_gid member (cmcred_gid成员), 429
 cmcred_groups member (cmcred_groups成员),
 429
 cmcred_ngroups member (cmcred_ngroups成员),
 429
 cmcred_pid member (cmcred_pid成员), 429
 cmcred_uid member (cmcred_uid成员), 429
 CMGROUP_MAX constant (CMGROUP_MAX常值), 429
 CMSG_DATA macro (CMSG_DATA宏), 425
 definition of (CMSG_DATA宏定义), 397
 CMSG_FIRSTHDR macro (CMSG_FIRSTHDR宏), 398, 590,
 730
 definition of (CMSG_FIRSTHDR宏定义), 397
 CMSG_LEN macro (CMSG_LEN宏), 398, 901
 definition of (CMSG_LEN宏定义), 397
 CMSG_NXTHDR macro (CMSG_NXTHDR宏), 398, 590, 730
 definition of (CMSG_NXTHDR宏定义), 397
 CMSG_SPACE macro (CMSG_SPACE宏), 398, 901
 definition of (CMSG_SPACE宏定义), 397
 cmsg_control member (cmsg_control成员), 398
 cmsg_data member (cmsg_data成员), 396–397, 425,
 722
 cmsg_len member (cmsg_len成员), 394, 396–398
 cmsg_level member (cmsg_level成员), 394, 396,
 616–619, 732
 cmsg_type member (cmsg_type成员), 394, 396,
 616–619, 732
 cmsgcred structure (cmsgcred结构), 429–430
 definition of (cmsgcred结构定义), 429
 cmsghdr structure (cmsghdr结构), 394, 396–398, 409,
 425, 615–619, 722, 727, 732
 definition of (cmsghdr结构定义), 396
 CNAME (canonical name record, DNS), 305, 307, 310
 code field (代码字段), ICMP, 882
 coding (代码编写, 编码)
 style (代码编写风格), 8, 12
 TLV (TLV编码), 720
 Coene, L., 267, 952
 Common Desktop Environment (公共桌面环境), 见CDE
 Common Standards Revision Group (公共标准修订组),
 见CSRG
 completed connection queue (已完成连接队列), 104

- completely duplicate binding (完全重复的捆绑), 211–213, 922
- Computer Emergency Response Team (计算机紧急响应组), 见CERT
- Computer Systems Research Group (计算机系统研究组), 见CSRG
- concurrent programming (并发编程), 698
- concurrent server (并发服务器), 15, 114–116
- one child per client (每个客户一个子进程), TCP, 822–825
 - one thread per client (每个客户一个线程), TCP, 842–843
 - port numbers and (端口号和并发服务器), 52–55
 - UDP (UDP并发服务器), 612–614
- condition variable (条件变量), 701–705
- config.h header (config.h头文件), 425, 904–909
- configure program (configure程序), 904
- congestion avoidance (拥塞避免), 461, 596, 950
- CONIND_number member (CONIND_number成员), 860, 862
- conn_req structure (conn_req结构), 863
- connect function (connect函数), 7, 9, 11, 13, 29, 37–38, 45, 52, 63, 68, 74, 76, 99–102, 104–105, 107, 118, 120, 125–127, 135, 140, 146, 152, 165, 184, 208, 213, 237, 239, 241, 245, 249, 252–257, 261–262, 307, 314, 317, 319, 327–329, 337, 350, 355, 357–359, 361–362, 367, 382–383, 386, 408–409, 415–416, 420, 432, 436, 448–449, 451–452, 454, 457–459, 461, 464, 694, 696, 707, 717, 736, 739, 769, 772, 777, 826, 892–893, 915, 920–921, 933, 935
- definition of (connect函数定义), 99
 - interrupted (被中断的connect调用), 451–452
 - nonblocking (非阻塞connect调用), 448–461
 - timeout (connect调用超时), 382–383
 - UDP (UDP上的connect函数), 252–255
- connect_nonb function (connect_nonb函数), 449, 454
- source code (connect_nonb函数源代码), 450
- connect_timeo function (connect_timeo函数), 382
- source code (connect_timeo函数源代码), 382
- connected TCP socket (已连接TCP套接字), 109
- connected UDP socket (已连接UDP套接字), 252
- connection (连接)
- abort (连接中止), accept function (accept函数), 139–141
 - establishment, SCTP (SCTP连接建立), 44–50
 - establishment, TCP (TCP连接建立), 37–43
 - persistent (持续连接), 825
 - queue, completed (已完成连接队列), 104
 - queue, incomplete (未完成连接队列), 104
 - termination, SCTP (SCTP连接终止), 44–50
 - termination, TCP (TCP连接终止), 37–43
- connectionless (无连接的), 34
- connection-oriented (面向连接的), 35
- Conrad, P., 285, 953
- const qualifier (const限定词), 81, 103, 162
- Conta, A., 871, 882, 884, 948, 952
- continent-local multicast scope (大洲局部多播范围), 552
- control information (控制信息), 见ancillary data (辅助数据)
- conventions (约定)
- source code (源代码约定), 7
 - typographical (印刷上的约定), 7
- COOKIE-ECHOED state (COOKIE-ECHOED状态), 47–48
- COOKIE-WAIT state (COOKIE-WAIT状态), 47–48
- Coordinated Universal Time (国际标准时间), 见UTC
- copy (复制)
- deep (深度复制), 321
 - shallow (浅度复制), 321
- copy-on-write (写时复制), 675
- crypto function (crypto函数), 680, 944
- core file (core文件), 369
- CORRECT_prim member (CORRECT_prim成员), 865
- cpio program (cpio程序), 26
- CPU_VENDOR_OS constant (CPU_VENDOR_OS常值), 78
- CR (carriage return), 9, 895, 916
- crashing and rebooting of server host (服务器主机崩溃并重启), 144–145
- crashing of server host (服务器主机崩溃), 144
- Crawford, M., 551, 948–949
- credentials, receiving sender (接收发送者凭证), 429–431
- creeping featurism (卑躬屈膝的特征主义), 741
- cron program (cron程序), 364, 366
- CSRG (Computer Systems Research Group), 20
- CSRG (Common Standards Revision Group), 25
- ctermid function (ctermid函数), 685
- ctime function (ctime函数), 15, 685
- ctime_r function (ctime_r函数), 685
- CTL_NET constant (CTL_NET常值), 496–497, 499
- daemon (守护程序, 守护进程), 16
- definition of (守护进程定义), 363
 - process (守护进程), 363–380
- daemon function (daemon函数), 367
- daemon_inetd function (daemon_inetd函数), 377–379
- source code (daemon_inetd函数源代码), 377
- daemon_init function (daemon_init函数), 367–372, 378–380
- source code (daemon_init函数源代码), 368
- daemon_proc variable (daemon_proc变量), 369, 378, 910
- data formats (数据格式), 147–151
- binary structures (二进制结构), 148–151
 - text strings (文本串), 147–148
- Data Link Provider Interface (数据链路提供者接口), 见DLPI
- data member (data成员), 405, 408
- datagram (数据报)
- service, reliable (可靠数据报服务), 597–608
 - socket (数据报套接字), 33

- truncation, UDP (UDP数据报截断), 594
 datalink socket address structure (数据链路套接字地址结构), routing socket (路由套接字), 486–487
 daytime port (时间获取服务端口), 61–62
 DCE (Distributed Computing Environment), 597
 RPC (DCE RPC), 62
 de Groot, G. J., 876, 952
 deadlock (死锁), 916
 debugging techniques (调试技术), 891–897
 deep copy (深度复制), 321
 Deering, S. E., 55–57, 216, 529, 550, 564, 721, 726, 871, 873, 877–879, 882, 884, 948–949, 951–952
 delayed ACK (延滞ACK), 220, 237, 923
 delta time (增量时间), 704
 denial-of-service attack (拒绝服务攻击), 46, 108, 180, 463, 934
 descriptor (描述符)
 passing (描述符传递), 420–428, 769, 836–842
 reference count (描述符引用计数), 117, 421
 set (描述符集), 162
 design alternatives, client/server (客户/服务器程序供选择的设计范式), 817–850
 DEST_length member (DEST_length成员), 863
 DEST_offset member (DEST_offset成员), 863
 destination (目的)
 address, IPv4 (IPv4目的地址), 871
 address, IPv6 (IPv6目的地址), 873
 IP address, recvmsg function, receiving (recvmsg函数接收IP目的地址), 588–593
 options, IPv6 (IPv6目的地选项), 719–725
 unreachable, fragmentation required, ICMP (ICMP目的地不可达, 需分片但DF位已设置), 56, 771, 883
 unreachable, ICMP (ICMP目的地不可达), 100–101, 144, 200, 249, 762, 764–765, 771, 775, 865, 883–884
 destructor function (析构函数), 690
 detached thread (已脱离的线程), 678
 Detailed Network Interface (详尽网络接口), 见DNI
 /dev/bpf device (/dev/bpf设备), 799
 /dev/console device (/dev/console设备), 364
 /dev/klog device (/dev/klog设备), 364
 /dev/kmem device (/dev/kmem设备), 482, 484
 /dev/log device (/dev/log设备), 364
 /dev/null device (/dev/null设备), 370, 701
 /dev/poll device (/dev/poll设备), 402–404
 /dev/tcp device (/dev/tcp设备), 859
 /dev/zero device (/dev/zero设备), 830, 836
 DF (don't fragment flag, IP header), 56, 444, 771, 871, 883
 DG structure (DG结构), 666
 dg_cli function (dg_cli函数), 244–246, 256–257, 383, 385–386, 419, 535–536, 541, 544–545, 547, 570, 599, 728, 772, 925
 dg_echo function (dg_echo函数), 242, 244–245, 257, 260, 419, 592, 666, 668, 729
 dg_send_recv function (dg_send_recv函数), 599, 601, 604, 606, 620
 source code (dg_send_recv函数源代码), 602
 DHCP (Dynamic Host Configuration Protocol), 62, 530, 532
 Differentiated Services (区分服务), 870–871
 Digital Unix, 257, 346, 700
 disaster, recipe for (导致灾难性后果), 684
 discard port (丢弃服务端口), 61
 DISCON_reason member (DISCON_reason成员), 865
 diskless node (无盘节点), 34
 DISPLAY environment variable (DISPLAY环境变量), 411
 Distributed Computing Environment (分布式计算环境), 见DCE
 DL_ATTACH_REQ constant (DL_ATTACH_REQ常值), 790
 DLPI (Data Link Provider Interface), 32, 34, 98, 787, 790–791, 793, 810, 854, 954
 DLT_EN10MB constant (DLT_EN10MB常值), 808
 DNI (Detailed Network Interface), 27
 DNS (Domain Name System), 9, 57, 62, 239, 303–306, 311–312, 789
 absolute name (DNS绝对名字), 303
 alternatives (DNS替代方法), 306
 canonical name record (DNS规范名字记录), 见CNAME
 mail exchange record (DNS邮件交换记录), 见MX
 pointer record (DNS指针记录), 见PTR
 resource record (DNS资源记录), 见RR
 round robin (DNS轮询), 822
 simple name (DNS简单名字), 303
 do_get_read function (do_get_read函数), 695–697, 705
 dom_family member (dom_family成员), 99
 Domain Name System (域名系统), 见DNS
 domain structure (domain结构), 99
 don't fragment flag, IP header (IP首部不分片标志), 见DF
 dotted-decimal notation (点分十进制数串记法), 874
 double buffering (双缓冲), 789
 DP_POLL constant (DP_POLL常值), 403
 dp_fds member (dp_fds成员), 403
 dp_nfds member (dp_nfds成员), 403
 dp_timeout member (dp_timeout成员), 403
 Draves, R., 317, 879, 949, 951
 driver, STREAMS (流驱动程序), 851
 DSCP, 215, 870–871
 dual-stack host (双栈主机), 322, 325, 330, 332–333, 353–357, 359
 definition of (双栈主机定义), 34
 dup function (dup函数), 829
 dup2 function (dup2函数), 373
 duplicate (重复分组)
 lost (迷途的重复分组), 43
 wandering (漫游的重复分组), 43
 Durst, W., 315
 Dynamic Host Configuration Protocol (动态主机配置协议), 见DHCP
 dynamic port (动态端口), 51

- EACCES error (EACCES错误), 199, 535
 EADDRINUSE error (EADDRINUSE错误), 103, 451, 609, 921
 EAFNOSUPPORT error (EAFNOSUPPORT错误), 83, 254
 EAGAIN error (EAGAIN错误), 436, 658, 677
 EAI_AGAIN constant (EAI_AGAIN常值), 321
 EAI_BADFLAGS constant (EAI_BADFLAGS常值), 321
 EAI_FAIL constant (EAI_FAIL常值), 321
 EAI_FAMILY constant (EAI_FAMILY常值), 321
 EAI_MEMORY constant (EAI_MEMORY常值), 321
 EAI_NONAME constant (EAI_NONAME常值), 321
 EAI_OVERFLOW constant (EAI_OVERFLOW常值), 321
 EAI_SERVICE constant (EAI_SERVICE常值), 321
 EAI_SOCKTYPE constant (EAI_SOCKTYPE常值), 321
 EAI_SYSTEM constant (EAI_SYSTEM常值), 321
 EBUSY error (EBUSY错误), 790
 echo port (回射服务端口), 61–62, 380
 echo reply, ICMP (ICMP回射应答), 735, 741, 883–884
 echo request, ICMP (ICMP回射请求), 735, 739, 741, 883–884
 ECN, 215, 870–871
 ECONNABORTED error (ECONNABORTED错误), 140, 463
 ECONNREFUSED error (ECONNREFUSED错误), 13, 99, 257, 416, 451, 771, 865, 883–884
 ECONNRESET error (ECONNRESET错误), 142, 145, 200, 921
 EDESTADDRREQ error (EDESTADDRREQ错误), 253
 EEXIST error (EEXIST错误), 495
 EHOSTDOWN error (EHOSTDOWN错误), 884
 EHOSTUNREACH error (EHOSTUNREACH错误), 100–101, 144, 201–202, 771, 865, 883–884
 EINPROGRESS error (EINPROGRESS错误), 436, 448–449
 EINTR error (EINTR错误), 90, 134–135, 138, 162, 181, 263, 383, 451, 463, 536, 545, 547, 669, 765, 803, 939
 EINVAL error (EINVAL错误), 162, 232, 274, 475, 514, 648, 651, 775, 915
 EISCONN error (EISCONN错误), 253, 451
 EMSGSIZE error (EMSGSIZE错误), 59, 225, 537, 771, 883, 925
 encapsulating security payload (安全净荷封装), 见ESP
 end of option list (选项列表结束选项), 见EOL
 end-of-file (文件结束符), 见EOF
 ENETUNREACH error (ENETUNREACH错误), 100, 144, 199
 ENOBUFS error (ENOBUFS错误), 60
 ENOENT error (ENOENT错误), 514
 ENOMEM error (ENOMEM错误), 496
 ENOPROTOOPT error (ENOPROTOOPT错误), 197, 883–884
 ENOSPC error (ENOSPC错误), 83
 ENOTCONN error (ENOTCONN错误), 253, 451
 environ variable (environ变量), 113
 environment variable (环境变量)
 DISPLAY (DISPLAY环境变量), 411
 LISTENQ (LISTENQ环境变量), 107
 PATH (PATH环境变量), 23, 113
 EOL (end of option list), 709, 713, 945
 EOPNOTSUPP error (EOPNOTSUPP错误), 231, 274
 ephemeral port (临时端口), 50–51, 53–54, 87, 99, 101–103, 111, 120, 122, 245–246, 250, 262, 341, 416, 613, 769, 772, 779, 915
 definition of (临时端口定义), 50
 EPIPE error (EPIPE错误), 142–143, 916
 Epoch (纪元), 14, 606
 EPROTO error (EPROTO错误), 140, 463, 832
 Eriksson, H., 885, 949
 err_doit function, source code (err_doit函数源代码), 910
 err_dump function (err_dump函数), 910
 source code (err_dump函数源代码), 910
 err_msg function (err_msg函数), 370, 910
 source code (err_msg函数源代码), 910
 err_quit function (err_quit函数), 11, 142, 380, 910
 source code (err_quit函数源代码), 910
 err_ret function (err_ret函数), 910
 source code (err_ret函数源代码), 910
 err_sys function (err_sys函数), 8, 11–13, 100, 257, 658, 910
 source code (err_sys函数源代码), 910
 errata availability (勘误表可获性), xxii
 errno variable (errno变量), 12–13, 30, 83, 140, 165, 167, 181, 184, 200, 249, 308, 321, 343–345, 365, 383, 424, 427, 451, 604, 676–677, 769, 771, 775, 783, 882–884, 910, 913
 error (错误)
 asynchronous (异步错误), 240, 249, 252–253, 769–786
 functions (错误处理函数), 910–912
 hard (硬错误), 99
 soft (软错误), 100
 ERROR_prim member (ERROR_prim成员), 862
 ESP (encapsulating security payload), 719, 951
 ESRCH error (ESRCH错误), 495
 ESTABLISHED state (ESTABLISHED状态), 40–41, 47–48, 63, 101, 104, 106, 127, 140, 916
 /etc/hosts file (/etc/hosts文件), 306, 348
 /etc/inetd.conf file (/etc/inetd.conf文件), 372–373, 379
 /etc/irs.conf file (/etc/irs.conf文件), 306
 /etc/netsvc.conf file (/etc/netsvc.conf文件), 306
 /etc/networks file (/etc/networks文件), 348–349
 /etc/nsswitch.conf file (/etc/nsswitch.conf文件), 306
 /etc/passwd file (/etc/passwd文件), 372
 /etc/protocols file (/etc/protocols文件), 348, 372–373
 /etc/rc file (/etc/rc文件), 363, 371
 /etc/resolv.conf file (/etc/resolv.conf文件), 254, 306, 317
 /etc/services file (/etc/services文件), 61, 311, 319, 348, 372, 379, 933
 /etc/syslog.conf file (/etc/syslog.conf文件), 364, 366, 379
 ETH_P_ARP constant (ETH_P_ARP常值), 792

- E**
- ETH_P_IP** constant (ETH_P_IP常值), 792
 - ETH_P_IPV6** constant (ETH_P_IPV6常值), 792
 - Ethernet (以太网), 34, 42, 55, 57, 63, 199, 208, 354–355, 471, 473–474, 482, 486, 502, 532, 534–535, 538, 550–551, 554–555, 792, 808–809, 870, 879, 914, 939
 - ETIME error (ETIME错误), 704
 - ETIMEDOUT error (ETIMEDOUT错误), 13, 99–101, 144, 200, 202, 383, 449, 451, 604, 865, 919, 924
 - EUI (extended unique identifier), 509, 879, 950
 - 64 format, modified (经修正的EUI-64格式), 879
 - EV_ADD** constant (EV_ADD常值), 406
 - EV_CLEAR** constant (EV_CLEAR常值), 406
 - EV_DELETE** constant (EV_DELETE常值), 406, 408
 - EV_DISABLE** constant (EV_DISABLE常值), 406
 - EV_ENABLE** constant (EV_ENABLE常值), 406
 - EV_EOF** constant (EV_EOF常值), 406
 - EV_ERROR** constant (EV_ERROR常值), 406
 - EV_ONESHOT** constant (EV_ONESHOT常值), 406
 - EV_SET** macro (EV_SET宏), 406
 - definition of (EV_SET宏定义), 405
 - events member (events成员), 183, 185, 188
 - EVFILT_AIO** constant (EVFILT_AIO常值), 406
 - EVFILT_PROC** constant (EVFILT_PROC常值), 406
 - EVFILT_READ** constant (EVFILT_READ常值), 406
 - EVFILT_SIGNAL** constant (EVFILT_SIGNAL常值), 406
 - EVFILT_TIMER** constant (EVFILT_TIMER常值), 406
 - EVFILT_VNODE** constant (EVFILT_VNODE常值), 406
 - EVFILT_WRITE** constant (EVFILT_WRITE常值), 406
 - EWOULDBLOCK** error (EWOULDBLOCK错误), 155, 203, 207, 386, 435–436, 439, 441–442, 463, 648, 657–658, 671, 945
 - examples road map, client/server (客户/服务器例子导读图), 16–18
 - exec** function (exec函数), 26, 111–114, 118–119, 147, 372–374, 376–377, 420–423, 676, 825, 850, 934
 - definition of (exec函数定义), 113
 - exec1** function (exec1函数), 423
 - definition of (exec1函数定义), 113
 - execle** function, definition of (execle函数定义), 113
 - execlp** function, definition of (execlp函数定义), 113
 - execv** function, definition of (execv函数定义), 113
 - execve** function, definition of (execve函数定义), 113
 - execvp** function, definition of (execvp函数定义), 113
 - exercises, solutions to (习题解答), 913–946
 - exit** function (exit函数), 9, 39, 114, 128, 137, 237, 401–402, 409, 427, 614, 679–680, 910, 935
 - expedited data (经加速数据), 见out-of-band data
 - exponential backoff (指数回退), 598, 802
 - extended unique identifier (经扩展的唯一标识符), 见EUI
 - extension headers, IPv6 (IPv6扩展首部), 719
 - external data representation (外部数据表示), 见XDR
- F**
- F_CONNECTING** constant (F_CONNECTING常值), 457–459
 - F_DONE** constant (F_DONE常值), 459, 697
 - F_GETFL** constant (F_GETFL常值), 235
 - F_GETOWN** constant (F_GETOWN常值), 234–236, 467
 - F_JOINED** constant (F_JOINED常值), 706
 - F_READING** constant (F_READING常值), 458–459
 - F_SETFL** constant (F_SETFL常值), 234–235, 468, 664
 - F_SETOWN** constant (F_SETOWN常值), 234–235, 467, 664
 - F_UNLCK** constant (F_UNLCK常值), 834
 - F_WRLCK** constant (F_WRLCK常值), 834
 - f_flags** member (f_flags成员), 458
 - f_tid** member (f_tid成员), 694
 - family_to_level** function (family_to_level函数), 567
- FAQ** (frequently asked question), 142, 210
- FASYNC** constant (FASYNC常值), 234
- fcntl** function (fcntl函数), 114, 191, 233–236, 439, 449, 466–468, 647, 649, 664, 669, 833–834
 - definition of (fcntl函数定义), 235
- fcred** structure (fcred结构), 397
- fd** member (fd成员), 183, 185, 188
- FD_CLOEXEC** constant (FD_CLOEXEC常值), 114
- FD_CLR** macro (FD_CLR宏), 376
 - definition of (FD_CLR宏定义), 163
- FD_ISSET** macro (FD_ISSET宏), 164
 - definition of (FD_ISSET宏定义), 163
- FD_SET** macro (FD_SET宏), 168, 697
 - definition of (FD_SET宏定义), 163
- FD_SETSIZE** constant (FD_SETSIZE常值), 163, 166, 177, 185
- FD_ZERO** macro (FD_ZERO宏), 168
 - definition of (FD_ZERO宏定义), 163
- fd_set** datatype (fd_set数据类型), 163–164, 185
- FDDI** (Fiber Distributed Data Interface), 34, 550–551
- fdopen** function (fdopen函数), 399–400
- Fenner, B., 564, 948
- fflags** member (fflags成员), 405
- fflush** function (fflush函数), 400–402
- fgets** function (fgets函数), 15, 121, 125–126, 128, 141–142, 153, 167–169, 171, 245, 287, 292, 400–401, 536, 851, 915–916, 924
- Fiber Distributed Data Interface (光纤分布式数据接口), 见FDDI
- FIFO (first in, first out), 243
- FILE** structure (FILE结构), 402, 679
- file** structure (file结构), 455, 459, 694–695, 706, 829
- file table** (文件表), 421
- File Transfer Protocol** (文件传送协议), 见FTP
- fileno** function (fileno函数), 168, 400
- filter** member (filter成员), 405
- filtering (过滤)
 - ICMPv6 type (ICMPv6类型过滤), 740–741
 - imperfect multicast (不完备多播过滤), 555
 - perfect (完备过滤), 555
- FIN** (finish flag, TCP header), 39–40, 179, 789
- FIN_WAIT_1** state (FIN_WAIT_1状态), 40–41
- FIN_WAIT_2** state (FIN_WAIT_2状态), 41, 128, 944
- finish flag**, TCP header (TCP首部完成标志), 见FIN

- Fink, R., 879, 888, 949
- FIOASYNC** constant (FIOASYNC常值), 234, 467–468, 664
- FIOGETOWN** constant (FIOGETOWN常值), 467–468
- FIONBIO** constant (FIONBIO常值), 234, 467–468
- FIONREAD** constant (FIONREAD常值), 234, 399, 409, 467–468
- FIOSETOWN** constant (FIOSETOWN常值), 467–468
- firewall (防火墙), 893, 948
- first in, first out (先进先出), 见FIFO
- flags member (Flags成员), 405
- flock** structure (flock结构), 834
- flooding (泛滥)
- broadcast (广播泛滥), 558
 - SYN (SYN分节泛滥), 108, 948
- flow control (流控), 35
- UDP lack of (UDP缺乏流控), 257–261
- flow label field, IPv6 (IPv6流标签字段), 871
- Floyd, S., 35, 215, 870–871, 947–948, 952
- FNDELAY** constant (FNDELAY常值), 234
- fopen** function (fopen函数), 851
- fork** function (fork函数), 15–16, 26, 53, 95, 111–115, 118, 120, 122, 126, 132, 139, 175, 243, 263, 368–369, 371, 373–377, 379–380, 405, 420–423, 430, 432, 446–448, 464, 577, 609, 612–614, 675–677, 679, 681, 698, 707, 717, 817–818, 820, 822–823, 825–827, 829–830, 837, 842, 850, 934, 944, 946
- definition of (fork函数定义), 111
- format prefix (格式前缀), 878
- formats (格式)
- binary structures, data (二进制结构数据格式), 148–151
 - data (数据格式), 147–151
 - text strings, data (文本串数据格式), 147–148
- four-way handshake (四路握手), 45
- SCTP (SCTP四路握手), 45–46
- fpathconf** function (fpathconf函数), 209
- fprintf** function (fprintf函数), 344, 365, 369–370, 439, 443
- fputs** function (fputs函数), 9, 11, 121, 125, 168–169, 245, 288, 400–402, 680, 919
- FQDN (fully qualified domain name), 303, 309, 317, 340
- fragmentation (分片), 56–57, 59, 719, 737, 739, 771–772, 870, 873, 883–884, 914, 926, 945
- and broadcast, IP (IP分片与广播), 537–538
 - and multicast, IP (IP分片与多播), 571
 - offset field, IPv4 (IPv4分片偏移字段), 871
- frame type (帧类型), 532, 534–535, 555, 791–792
- free** function (free函数), 508, 684
- free_ifi_info** function (free_ifi_info函数), 471, 478
- source code (free_ifi_info函数源代码), 480
- freeaddrinfo** function (freeaddrinfo函数), 321, 327, 345
- definition of (freeaddrinfo函数定义), 321
- FreeBSD, 20–24, 78, 108, 197, 260–262, 299, 405, 469, 473, 497, 538, 658, 666, 710, 775, 882–883, 891, 897, 904, 926, 934, 939–940
- freehostent** function (freehostent函数), 347
- definition of (freehostent函数定义), 347
- frequently asked question (常问问题), 见FAQ
- fseek** function (fseek函数), 400
- fsetpos** function (fsetpos函数), 400
- fstat** function (fstat函数), 406
- fstat** program (fstat程序), 897
- FTP (File Transfer Protocol), 20, 62, 201, 311–312, 360, 362, 366, 375, 662, 914, 947
- fudge factor (模糊因子), 106, 500
- full-duplex (全双工), 36, 415
- Fuller, V., 874, 949
- fully buffered standard I/O stream (完全缓冲的标准I/O流), 401
- fully qualified domain name (全限定域名), 见FQDN
- function (函数)
- destructor (析构函数), 690
 - system call versus (系统调用和函数对比), 891
 - wrapper (包裹函数), 11–13
- gai_strerror** function (gai_strerror函数), 320–321
- definition of (gai_strerror函数定义), 321
- Ganguly, S., 285, 953
- Garcia, M., 267, 952
- Garfinkel, S. L., 15, 949
- gated program (gated程序), 199, 485, 735
- gather write (集中写), 389
- generic socket address structure (通用套接字地址结构), 70–71
- new (新的通用套接字地址结构), 72–73
- get_ifi_info** function (get_ifi_info函数), 469–480, 482, 484, 500–503, 582, 608
- source code (get_ifi_info函数源代码), 474, 501
- get_rtaddrs** function (get_rtaddrs函数), 492–493, 502, 505
- getaddrinfo** function (getaddrinfo函数), 10, 15, 38, 93, 232, 303, 307, 315–329, 332–336, 338, 340–341, 343, 345–347, 349, 357, 361, 620, 746, 932, 941
- definition of (getaddrinfo函数定义), 315
 - examples (getaddrinfo函数例子), 324–325
 - IPv6 (getaddrinfo函数: IPv6), 322–323
- getc_unlocked** function (getc_unlocked函数), 685
- getchar_unlocked** function (getchar_unlocked函数), 685
- getconninfo** function (getconninfo函数), 315
- getgrid** function (getgrid函数), 685
- getgrid_r** function (getgrid_r函数), 685
- getgrnam** function (getgrnam函数), 685
- getgrnam_r** function (getgrnam_r函数), 685
- gethostbyaddr** function (gethostbyaddr函数), 303, 305–306, 310, 315, 341–343, 346, 348–350, 361, 685, 928–930

- definition of (gethostbyaddr函数定义),** 310
gethostbyaddr_r function (gethostbyaddr_r函数), 344–346
 definition of (gethostbyaddr_r函数定义), 345
gethostbyname function (gethostbyname函数), 303, 305–310, 312, 314–315, 320, 329, 341–350, 355, 361, 685, 929–930, 932–933
 definition of (gethostbyname函数定义), 307
gethostbyname2 function (gethostbyname2函数), 342, 346–347
 definition of (gethostbyname2函数定义), 347
gethostbyname_r function (gethostbyname_r函数), 344–346
 definition of (gethostbyname_r函数定义), 345
gethostent function (gethostent函数), 349
getifaddrs function (getifaddrs函数), 469
getipnodebyaddr function (getipnodebyaddr函数), 347
getipnodebyname function (getipnodebyname函数), 347
 definition of (getipnodebyname函数定义), 347
getlogin function (getlogin函数), 685
getlogin_r function (getlogin_r函数), 685
getmsg function (getmsg函数), 155, 809–810, 855–857, 860, 862, 864–868, 891
 definition of (getmsg函数定义), 856
getnameinfo function (getnameinfo函数), 38, 93, 303, 320, 331, 340–341, 343, 345, 347, 349–350, 361, 762, 933
 definition of (getnameinfo函数定义), 340
getnameinfo_timeo function (getnameinfo_timeo函数), 350
getnetbyaddr function (getnetbyaddr函数), 348
getnetbyname function (getnetbyname函数), 348
 getopt function (getopt函数), 516, 796
getpeername function (getpeername函数), 52, 68, 75, 117–120, 147, 275, 329, 340, 377–378, 451
 definition of (getpeername函数定义), 118
getpid function (getpid函数), 678
getpmsg function (getpmsg函数), 855, 857, 868
 definition of (getpmsg函数定义), 857
getppid function (getppid函数), 111, 938
getprotobynumber function (getprotobynumber函数), 348
getprotobynumber function (getprotobynumber函数), 348
getpwnam function (getpwnam函数), 373, 685
getpwnam_r function (getpwnam_r函数), 685
getpwuid function (getpwuid函数), 685
getpwuid_r function (getpwuid_r函数), 685
getrlimit function (getrlimit函数), 919
getrusage function (getrusage函数), 824, 827
gets function (gets函数), 15
getsatatypebyname function (getsatatypebyname函数), 516
getservbyaddr function (getservbyaddr函数), 348
getservbyname function (getservbyname函数), 303, 311–314, 320, 329, 343, 348–349, 373
 definition of (getservbyname函数定义), 311
getservbyport function (getservbyport函数), 303, 311–314, 343, 348
 definition of (getservbyport函数定义), 312
getsockname function (getsockname函数), 68, 75, 103, 117–120, 146–147, 211, 230, 251, 261, 340, 413–414, 769, 779, 915, 932
 definition of (getsockname函数定义), 118
getsockopt function (getsockopt函数), 76, 165, 191–194, 197, 200, 215, 218, 222–223, 226, 230, 237, 278, 451, 459, 559, 617, 710, 714, 717–718, 733, 740
 definition of (getsockopt函数定义), 192
gettimeofday function (gettimeofday函数), 582, 606, 704–705, 747
Gettys, J., 294, 949
getuid function (getuid函数), 799
gf_time function (gf_time函数), 442
 source code (gf_time函数源代码), 442
Gierth, A., 462, 949
GIF (graphics interchange format), 454, 825
Gilligan, R. E., 28, 71, 216, 346–347, 361, 504, 880, 949
global multicast scope (全球多播范围), 552–553
global routing prefix (全球路由前缀), 878
global unicast address (全球单播地址), 878–879
global unicast scope (全球单播范围), 878
gmtime function (gmtime函数), 685
gmtime_r function (gmtime_r函数), 685
goto, nonlocal (非本地跳转), 543, 803
gpic program (gpic程序), xxiii
gr_group member (gr_group成员), 560
gr_interface member (gr_interface成员), 560
graphics interchange format (图形交换格式), 见GIF
grep program (grep程序), 128, 913
group ID (组ID), 429, 431, 676
group_req structure (group_req结构), 193
 definition of (group_req结构定义), 560
group_source_req structure (group_source_req结构), 193
 definition of (group_source_req结构定义), 562
gsr_group member (gsr_group成员), 562
gsr_interface member (gsr_interface成员), 562
gsr_source member (gsr_source成员), 562
gtbl program (gtbl程序), xxiii

h_addr_list member (h_addr_list成员), 307–308, 929
h_addrtype member (h_addrtype成员), 307–308, 932
h_aliases member (h_aliases成员), 307–308
h_errno member (h_errno成员), 308, 345–346
h_length member (h_length成员), 307–308

- `h_name` member (`h_name`成员), 307–308, 310, 349
 Haberman, B., 551–552, 949
 hacker (黑客), 15, 108, 718, 786, 948
 half-close (半关闭), 39, 173, 895
 half-open connection (半打开的连接), 201, 236
 Handley, M., 571, 949
 hard error (硬错误), 99
 Harkins, D., 524, 949
`HAVE_MSGHDR_MSG_CONTROL` constant (`HAVE_MSGHDR_MSG_CONTROL`常值), 425
`HAVE_SOCKADDR_SA_LEN` constant (`HAVE_SOCKADDR_SA_LEN`常值), 68
`hdr` structure (`hdr`结构), 601, 603–604, 941
 head of line blocking (头端阻塞), 31, 293–299
 head, STREAMS (流头), 852
 header (首部)
 extension length (首部扩展长度), 719, 725
 length field, IPv4 (IPv4首部长度字段), 870
 High-Performance Parallel Interface (高性能并行接口),
 见HIPPI
 high-priority, STREAMS message (高优先级流消息), 183, 854
 Hinden, R., 55, 57, 216, 529, 721, 726, 871, 873, 877–879, 888, 948–949
 HIPPI (High-Performance Parallel Interface), 55
 historical advanced API, IPv6 (历史性IPv6高级API), 732
 history, BSD networking (BSD网络支持历史), 20–21
 Holbrook, H., 558, 950
 Holdrege, M., 267, 952
`home_page` function (`home_page`函数), 455–456, 694, 697
 hop count, routing (路由跳数), 481
 hop limit (跳限), 43, 217–218, 552, 559, 563, 566, 617, 750, 755, 757, 761, 772, 872–873, 884
 hop-by-hop options, IPv6 (IPv6步跳选项), 719–725
 host byte order (主机字节序), 77, 103, 110, 120, 148, 737, 740, 915
 Host Requirements RFC (主机要求RFC), 948
`HOST_NOT_FOUND` constant (`HOST_NOT_FOUND`常值), 308
`host_serv` function (`host_serv`函数), 325–326, 457, 713, 717, 728, 745, 757, 798
 definition of (`host_serv`函数定义), 325
 source code (`host_serv`函数源代码), 326
`hostent` structure (`hostent`结构), 307–308, 310, 345, 347–348, 929
 definition of (`hostent`结构定义), 307
`hostent_data` structure (`hostent_data`结构), 346
 HP-UX, 22, 78, 108, 257, 262, 306, 343, 346, 390, 538, 793
`hstrerror` function (`hstrerror`函数), 308, 310
 HTML (Hypertext Markup Language), 454, 825
`htonl` function (`htonl`函数), 79, 103, 152, 918
 definition of (`htonl`函数定义), 79
`htons` function (`htons`函数), 8, 311
 definition of (`htons`函数定义), 79
 HTTP (Hypertext Transfer Protocol), 9, 40, 62, 103, 106, 211, 452, 456, 459, 595, 696, 820, 825, 896
 Huitema, C., 304, 889, 950, 954
 Hypertext Markup Language (超文本标记语言), 见HTML
 Hypertext Transfer Protocol (超文本传送协议), 见HTTP
`I_RECVFD` constant (`I_RECVFD`常值), 420
`I_SENDFD` constant (`I_SENDFD`常值), 420
 IANA (Internet Assigned Numbers Authority), 50–52, 215, 311, 950, 953
 ICMP (Internet Control Message Protocol), 33, 62, 200, 249, 256–257, 735, 739, 742, 755, 896, 922, 925
 address request (ICMP地址掩码请求), 739, 883
 code field (ICMP代码字段), 882
 destination unreachable (ICMP目的地不可达错误), 100–101, 144, 200, 249, 762, 764–765, 771, 775, 865, 883–884
 destination unreachable, fragmentation required (ICMP目的地不可达, 需分片但DF位已设置), 56, 771, 883
 echo reply (ICMP回射应答), 735, 741, 883–884
 echo request (ICMP回射请求), 735, 739, 741, 883–884
 header, picture of (ICMP首部图示), 882
 message daemon, implementation (ICMP消息守护程序实现), 769–786
 packet too big (ICMP分组太大错误), 56, 771, 884
 parameter problem (ICMP参数问题错误), 720, 883–884
 port unreachable (ICMP端口不可达错误), 249, 253, 257, 265, 534, 755, 761, 764, 771, 794, 815, 883–884, 925
 redirect (ICMP重定向), 485, 497, 883–884
 router advertisement (ICMP路由器通告), 735, 741, 883–884
 router solicitation (ICMP路由器征求), 735, 883–884
 source quench (ICMP源熄灭), 771–772, 883
 time exceeded (ICMP超时), 755, 761, 764, 771, 883–884
 timestamp request (ICMP时间戳选项), 739, 883
 type field (ICMP类型字段), 882
`ICMP6_FILTER` socket option (`ICMP6_FILTER`套接字选项), 216, 740
`ICMP6_FILTER_SETBLOCK` macro, definition of (`ICMP6_FILTER_SETBLOCK`宏定义), 740
`ICMP6_FILTER_SETBLOCKALL` macro, definition of (`ICMP6_FILTER_SETBLOCKALL`宏定义), 740
`ICMP6_FILTER_SETPASS` macro, definition of (`ICMP6_FILTER_SETPASS`宏定义), 740
`ICMP6_FILTER_SETPASSALL` macro, definition of (`ICMP6_FILTER_SETPASSALL`宏定义), 740
`ICMP6_FILTER_WILLBLOCK` macro, definition of (`ICMP6_FILTER_WILLBLOCK`宏定义), 740
`ICMP6_FILTER_WILLPASS` macro, definition of (`ICMP6_FILTER_WILLPASS`宏定义), 740
`icmp6_filter` structure (`icmp6_filter`结构), 193, 216, 740
`icmpcode_v4` function (`icmpcode_v4`函数), 765
`icmpcode_v6` function (`icmpcode_v6`函数), 765
`icmpd` program (`icmpd`程序), 769, 772, 774–786, 946
`icmpd_dest` member (`icmpd_dest`成员), 772

- icmpd_err member (icmpd_err成员), 771, 774, 783–784
 icmpd_errno member (icmpd_errno成员), 771
 icmpd.h header (icmpd.h头文件), 775
ICMPv4 (Internet Control Message Protocol)
 version 4), 33–34, 735, 740, 769, 871, 882–884
 checksum (ICMPv4校验和), 737, 753, 806, 882
 header (ICMPv4首部), 743, 755
 message types (ICMPv4消息类型), 883
ICMPv6 (Internet Control Message Protocol)
 version 6), 33–34, 216, 735, 738, 769, 882–884
 checksum (ICMPv6校验和), 738, 753–754, 882
 header (ICMPv6首部), 744, 755
 message types (ICMPv6消息类型), 884
 multicast listener done (ICMPv6多播收听者结束), 884
 multicast listener query (ICMPv6多播收听者查询), 884
 multicast listener report (ICMPv6多播收听者汇报), 884
 neighbor advertisement (ICMPv6邻居通告), 884
 neighbor advertisement, inverse (ICMPv6反向邻居通告), 884
 neighbor solicitation (ICMPv6邻居征求), 884
 neighbor solicitation, inverse (ICMPv6反向邻居征求), 884
 socket option (ICMPv6套接字选项), 216
 type filtering (ICMPv6类型过滤), 740–741
id program (id程序), 431
ident member (ident成员), 405
 identification field, IPv4 (IPv4标识字段), 870
IEC (International Electrotechnical Commission), 26, 950
IEEE (Institute of Electrical and Electronics Engineers), 26, 509, 550, 879, 950
IEEE-IX, 26
IETF (Internet Engineering Task Force), 28, 947
if_announcemsg_hdr structure (if_announcemsg_hdr结构), 487
 definition of (if_announcemsg_hdr结构定义), 488
if_freenameindex function (if_freenameindex函数), 504–508
 definition of (if_freenameindex函数定义), 504
 source code (if_freenameindex函数源代码), 508
if_index member (if_index成员), 504, 903
if_indextoname function (if_indextoname函数), 504–508, 566, 568, 593
 definition of (if_indextoname函数定义), 504
 source code (if_indextoname函数源代码), 506
if_msghdr structure (if_msghdr结构), 487, 502
 definition of (if_msghdr结构定义), 488
if_name member (if_name成员), 504, 508, 903
if_nameindex function (if_nameindex函数), 486, 504–508
 definition of (if_nameindex函数定义), 504
 source code (if_nameindex函数源代码), 507
if_nameindex structure (if_nameindex结构), 504, 507–508, 903
 definition of (if_nameindex结构定义), 504
if_nametoindex function (if_nametoindex函数), 486, 504–508, 566–567, 569
 definition of (if_nametoindex函数定义), 504
 source code (if_nametoindex函数源代码), 505
ifa_msghdr structure (ifa_msghdr结构), 487
 definition of (ifa_msghdr结构定义), 488
ifam_addrs member (ifam_addrs成员), 489, 493
ifc_buf member (ifc_buf成员), 469–470
ifc_len member (ifc_len成员), 77, 468, 470
ifc_req member (ifc_req成员), 469
ifconf structure (ifconf结构), 77, 467–468, 470
 definition of (ifconf结构定义), 469
ifconfig program (ifconfig程序), 23, 25, 103, 234, 471, 480
IFF_BROADCAST constant (IFF_BROADCAST常值), 480
IFF_POINTOPOINT constant (IFF_POINTOPOINT常值), 480
IFF_PROMISC constant (IFF_PROMISC常值), 792
IFF_UP constant (IFF_UP常值), 480
ifi_hlen member (ifi_hlen成员), 473, 478, 502
ifi_index member (ifi_index成员), 502
ifi_info structure (ifi_info结构), 469, 471, 473, 475, 478, 484, 500, 502, 608
ifi_next member (ifi_next成员), 471, 478
ifm_addrs member (ifm_addrs成员), 489, 493
ifm_type member (ifm_type成员), 502
ifma_msghdr structure (ifma_msghdr结构), 487
 definition of (ifma_msghdr结构定义), 488
ifmam_addrs member (ifmam_addrs成员), 489
IFNAMSIZ constant (IFNAMSIZ常值), 504
ifr_addr member (ifr_addr成员), 469, 480–481
ifr_broadaddr member (ifr_broadaddr成员), 469, 481, 484
ifr_data member (ifr_data成员), 469
ifr_dstaddr member (ifr_dstaddr成员), 469, 481, 484
ifr_flags member (ifr_flags成员), 469, 480–481
ifr_metric member (ifr_metric成员), 469, 481
ifr_name member (ifr_name成员), 470, 480
ifreq structure (ifreq结构), 467–468, 470, 475, 477, 480, 484, 568
 definition of (ifreq结构定义), 469
IFT_NONE constant (IFT_NONE常值), 591
IGMP (Internet Group Management Protocol), 33–34, 556, 735, 739–740, 871
 checksum (IGMP校验和), 753
ILP32, programming model (ILP32编程模型), 28
imperfect multicast filtering (不完备多播过滤), 555
implementation (实现)
ICMP message daemon (ICMP消息守护程序实现), 769–786
ping program (ping程序实现), 741–754
traceroute program (traceroute程序实现), 755–768
imr_interface member (imr_interface成员), 560, 562, 568
imr_multiaddr member (imr_multiaddr成员),

- 560, 562
imr_sourceaddr member (**imr_sourceaddr**成员), 562
IN6_IS_ADDR_LINKLOCAL macro, definition of (**IN6_IS_ADDR_LINKLOCAL**宏定义), 360
IN6_IS_ADDR_LOOPBACK macro, definition of (**IN6_IS_ADDR_LOOPBACK**宏定义), 360
IN6_IS_ADDR_MC_GLOBAL macro, definition of (**IN6_IS_ADDR_MC_GLOBAL**宏定义), 360
IN6_IS_ADDR_MC_LINKLOCAL macro, definition of (**IN6_IS_ADDR_MC_LINKLOCAL**宏定义), 360
IN6_IS_ADDR_MC_NODELOCAL macro, definition of (**IN6_IS_ADDR_MC_NODELOCAL**宏定义), 360
IN6_IS_ADDR_MC_ORGLOCAL macro, definition of (**IN6_IS_ADDR_MC_ORGLOCAL**宏定义), 360
IN6_IS_ADDR_MC_SITELOCAL macro, definition of (**IN6_IS_ADDR_MC_SITELOCAL**宏定义), 360
IN6_IS_ADDR_MULTICAST macro, definition of (**IN6_IS_ADDR_MULTICAST**宏定义), 360
IN6_IS_ADDR_SITELOCAL macro, definition of (**IN6_IS_ADDR_SITELOCAL**宏定义), 360
IN6_IS_ADDR_UNSPECIFIED macro, definition of (**IN6_IS_ADDR_UNSPECIFIED**宏定义), 360
IN6_IS_ADDR_V4COMPAT macro, definition of (**IN6_IS_ADDR_V4COMPAT**宏定义), 360
IN6_IS_ADDR_V4MAPPED macro (**IN6_IS_ADDR_V4MAPPED**宏定义), 355, 360, 362, 745
 definition of (**IN6_IS_ADDR_V4MAPPED**宏定义), 360
in6_addr structure (**in6_addr**结构), 193, 561
 definition of (**in6_addr**结构定义), 71
in6_pktnfo structure (**in6_pktnfo**结构), 588, 615–617, 731
 definition of (**in6_pktnfo**结构定义), 616
IN6ADDR_ANY_INIT constant (**IN6ADDR_ANY_INIT**常值), 103, 320, 322, 412, 616, 881
IN6ADDR_LOOPBACK_INIT constant (**IN6ADDR_LOOPBACK_INIT**常值), 880
in6addr_any constant (**in6addr_any**常值), 103, 881
in6addr_loopback constant (**in6addr_loopback**常值), 880
in_addr structure (**in_addr**结构), 70, 193, 308, 310, 358, 560, 563
 definition of (**in_addr**结构定义), 68
in_addr_t datatype (**in_addr_t**数据类型), 69–70
in_cksum function (**in_cksum**函数), 753
 source code (**in_cksum**函数源代码), 753
in_pcdbdetach function (**in_pcdbdetach**函数), 140
in_port_t datatype (**in_port_t**数据类型), 69
INADDR_ANY constant (**INADDR_ANY**常值), 13, 53, 102–103, 122, 126, 214, 242, 288, 320, 322, 412, 534, 560–563, 859, 876, 915
INADDR_LOOPBACK constant (**INADDR_LOOPBACK**常值), 876
INADDR_MAX_LOCAL_GROUP constant (**INADDR_MAX_LOCAL_GROUP**常值), 915
INADDR_NONE constant (**INADDR_NONE**常值), 82, 901, 915
in-addr.arpa domain (**in-addr.arpa域**), 304, 310
in-band data (带内数据), 645
incarnation, definition of (化身定义), 44
incomplete connection queue (未完成连接队列), 104
index, interface (接口索引), 217, 489, 498, 502, 504–508, 560–563, 566, 569, 577, 616, 731
INET6_ADDRSTRLEN constant (**INET6_ADDRSTRLEN**常值), 83, 86, 901
inet6_opt_append function (**inet6_opt_append**函数), 723–724
 definition of (**inet6_opt_append**函数定义), 723
inet6_opt_find function (**inet6_opt_find**函数), 725
 definition of (**inet6_opt_find**函数定义), 724
inet6_opt_finish function (**inet6_opt_finish**函数), 723–724
 definition of (**inet6_opt_finish**函数定义), 723
inet6_opt_get_val function (**inet6_opt_get_val**函数), 725
 definition of (**inet6_opt_get_val**函数定义), 724
inet6_opt_init function (**inet6_opt_init**函数), 723–724
 definition of (**inet6_opt_init**函数定义), 723
inet6_option_alloc function (**inet6_option_alloc**函数), 732
inet6_option_append function (**inet6_option_append**函数), 732
inet6_option_find function (**inet6_option_find**函数), 732
inet6_option_init function (**inet6_option_init**函数), 732
inet6_option_next function (**inet6_option_next**函数), 732
inet6_option_space function (**inet6_option_space**函数), 732
inet6_opt_next function (**inet6_opt_next**函数), 724–725
 definition of (**inet6_opt_next**函数定义), 724
inet6_opt_set_val function (**inet6_opt_set_val**函数), 723–725
 definition of (**inet6_opt_set_val**函数定义), 723
inet6_rth_add function (**inet6_rth_add**函数), 727–728
 definition of (**inet6_rth_add**函数定义), 727
inet6_rthdr_add function (**inet6_rthdr_add**函数), 732
inet6_rthdr_getaddr function (**inet6_rthdr_getaddr**函数), 732
inet6_rthdr_getflags function (**inet6_rthdr_getflags**函数), 732
inet6_rthdr_init function (**inet6_rthdr_init**函数), 732
inet6_rthdr_lasthop function (**inet6_rthdr_lasthop**函数), 732
inet6_rthdr_reverse function (**inet6_rthdr_reverse**函数), 732
inet6_rthdr_segments function (**inet6_rthdr_segments**函数), 732

- inet6_rthdr_space function (inet6_rthdr_space函数), 732
 inet6_rth_getaddr function (inet6_rth_getaddr函数), 728, 731
 definition of (inet6_rth_getaddr函数定义), 728
 inet6_rth_init function (inet6_rth_init函数), 727–728
 definition of (inet6_rth_init函数定义), 727
 inet6_rth_reverse function (inet6_rth_reverse函数), 728, 730
 definition of (inet6_rth_reverse函数定义), 728
 inet6_rth_segments function (inet6_rth_segments函数), 728, 731
 definition of (inet6_rth_segments函数定义), 728
 inet6_rth_space function (inet6_rth_space函数), 727–728
 definition of (inet6_rth_space函数定义), 727
 inet6_srcrt_print function (inet6_srcrt_print函数), 730–731
 INET_ADDRSTRLEN constant (INET_ADDRSTRLEN常值), 83, 86, 901
 inet_addr function (inet_addr函数), 9, 67, 82–83, 93
 definition of (inet_addr函数定义), 82
 inet_aton function (inet_aton函数), 82–83, 93, 314
 definition of (inet_aton函数定义), 82
 inet_ntoa function (inet_ntoa函数), 67, 82–83, 343, 685
 definition of (inet_ntoa函数定义), 82
 inet_ntop function (inet_ntop函数), 67, 82–86, 93, 110, 309, 341, 343, 345, 350, 593, 731
 definition of (inet_ntop函数定义), 83
 IPv4-only version, source code (inet_ntop函数仅适合IPv4版本的源代码), 85
 inet_pton function (inet_pton函数), 8–9, 11, 67, 82–85, 93, 290, 333, 343, 930
 definition of (inet_pton函数定义), 83
 IPv4-only version, source code (inet_pton函数仅适合IPv4版本的源代码), 85
 inet_pton_loose function (inet_pton_loose函数), 93
 inet_srcrt_add function (inet_srcrt_add函数), 713, 715
 inet_srcrt_init function (inet_srcrt_init函数), 712, 715
 inet_srcrt_print function (inet_srcrt_print函数), 714
 inetd program (inetd程序), 61, 114, 118–119, 154, 363, 371–380, 587, 613–614, 825, 850, 897, 934, 945
 Information Retrieval Service (信息检索服务), 见IRS
 INF_TIM constant (INF_TIM常值), 184, 902
 init program (init程序), 132, 145, 938
 init_v6 function (init_v6函数), 749
 initial thread (初始线程), 676
 in.rdisc program (in.rdisc程序), 735
 Institute of Electrical and Electronics Engineers (电气电子工程师协会), 见IEEE
 int16_t datatype (int16_t数据类型), 69
 int32_t datatype (int32_t数据类型), 69
 int8_t datatype (int8_t数据类型), 69
 interface (接口)
 address, UDP, binding (UDP接口地址捆绑), 608–612
 configuration, ioctl function (ioctl函数接口配置), 468–469
 index (接口索引), 217, 489, 498, 502, 504–508, 560–563, 566, 569, 577, 616, 731
 index, recvmsg function, receiving (recvmsg函数接收接口索引), 588–593
 logical (逻辑接口), 877
 loopback (环回接口), 23, 792, 799, 809, 876–877
 message-based (基于消息的接口), 858
 operations, ioctl function (ioctl函数接口操作), 480–481
 UDP determining outgoing (UDP确定外出接口), 261–262
 interface-local multicast scope (接口局部多播范围), 552–553
 International Electrotechnical Commission (国际电工委员会), 见IEC
 International Organization for Standardization (国际标准化组织), 见ISO
 Internet (因特网; 网际网), 5, 22
 Internet Assigned Numbers Authority (因特网已分配数值权威机构), 见IANA
 Internet Control Message Protocol (网际网控制消息协议), 见ICMP
 Internet Control Message Protocol version 4 (网际网控制消息协议版本4), 见ICMPv4
 Internet Control Message Protocol version 6 (网际网控制消息协议版本6), 见ICMPv6
 Internet Draft (因特网草案), 947
 Internet Engineering Task Force (因特网工程任务攻坚组), 见IETF
 Internet Group Management Protocol (网际网组管理协议), 见IGMP
 Internet Protocol (网际网协议), 见IP
 Internet Protocol next generation (下一代网际网协议), 见IPng
 Internet Protocol version 4 (网际网协议版本4), 见IPv4
 Internet Protocol version 6 (网际网协议版本6), 见IPv6
 Internet service provider (因特网业务供应商), 见ISP
 Internetwork Packet Exchange (网络间分组交换), 见IPX
 interoperability (互操作性)
 IPv4 and IPv6 (IPv4和IPv6之间的互操作性), 353–362
 IPv4 client IPv6 server (IPv4客户和IPv6服务器之间的互操作性), 354–357
 IPv6 client IPv4 server (IPv6客户和IPv4服务器之间的互操作性), 357–359
 source code portability (源代码可移植性与互操作性), 361
 interprocess communication (进程间通信), 见IPC
 interrupts, software (软件中断), 129

- inverse, ICMPv6 neighbor advertisement (ICMPv6 反向邻居通告), 884
- ICMPv6 neighbor solicitation (ICMPv6 邻居征求), 884
- I/O
- asynchronous (异步 I/O), 160, 468, 663
 - definition of, Unix (Unix I/O 定义), 399
 - model, asynchronous (异步 I/O 模型), 158–159
 - model, blocking (阻塞式 I/O 模型), 154–155
 - model, comparison of (I/O 模型比较), 159–160
 - model, I/O, multiplexing (I/O 复用模型), 156–157
 - model, nonblocking (非阻塞式 I/O 模型), 155–156
 - model, signal-driven (信号驱动式 I/O 模型), 157–158
 - models (I/O 模型), 154–160
 - multiplexing (I/O 复用), 153–189
 - multiplexing I/O, model (I/O 复用模型), 156–157
 - nonblocking (非阻塞式 I/O), 88, 165, 234–235, 388, 398, 435–464, 468, 665, 669, 671, 919, 945
 - signal-driven (信号驱动式 I/O), 200, 234–235, 663–673
 - standard (标准 I/O), 168, 344, 399–402, 409, 437, 935, 952
 - synchronous (同步 I/O), 160
- ioctl function (ioctl 函数), 191, 222, 233–234, 399, 403–404, 409, 420, 465–469, 474–475, 477–478, 480–485, 500, 538, 566, 568, 585, 647, 654, 664, 666, 669, 790, 792, 799, 852, 857, 868
- ARP cache operations (ioctl 函数 ARP 高速缓存操作), 481–483
- definition of (ioctl 函数定义), 466, 857
- file operations (ioctl 函数文件操作), 468
- interface configuration (ioctl 函数接口配置), 468–469
- interface operations (ioctl 函数接口操作), 480–481
- routing table operations (ioctl 函数路由表操作), 483–484
- socket operations (ioctl 函数套接字操作), 466–467
- STREAMS (ioctl 函数流处理), 857–858
- IOV_MAX constant (IOV_MAX 常值), 390
- iov_base member (iov_base 成员), 389
- iov_len member (iov_len 成员), 389, 392
- iovec structure (iovec 结构), 389–391, 393, 601
- definition of (iovec 结构定义), 389
- IP (Internet Protocol), 33
- fragmentation and broadcast (IP 分片与广播), 537–538
 - fragmentation and multicast (IP 分片与多播), 571
 - Multicast Infrastructure (IP 多播基础设施), 571, 584–585
 - Multicast Infrastructure session
 - announcements (IP 多播基础设施会话声明), 571–575
 - routing (IP 路由), 869
 - spoofing (IP 欺骗), 108, 948
 - version number field (IP 版本号字段), 869, 871
- ip6_mtuinfo structure, definition of (ip6_mtuinfo 结构定义), 619
- ip6_arpa domain (ip6_arpa 域), 304
- ip6m_addr member (ip6m_addr 成员), 619
- ip6m_mtu member (ip6m_mtu 成员), 619
- IP_ADD_MEMBERSHIP socket option (IP_ADD_MEMBERSHIP 套接字选项), 193, 560, 562
- IP_ADD_SOURCE_MEMBERSHIP socket option (IP_ADD_SOURCE_MEMBERSHIP 套接字选项), 193, 560
- IP_BLOCK_SOURCE socket option (IP_BLOCK_SOURCE 套接字选项), 193, 560, 562
- IP_DROP_MEMBERSHIP socket option (IP_DROP_MEMBERSHIP 套接字选项), 193, 560–561
- IP_DROP_SOURCE_MEMBERSHIP socket option (IP_DROP_SOURCE_MEMBERSHIP 套接字选项), 193, 560
- IP_HDRINCL socket option (IP_HDRINCL 套接字选项), 193, 214, 710, 736–738, 753, 755, 790, 793, 805–806
- IP_MULTICAST_IF socket option (IP_MULTICAST_IF 套接字选项), 193, 559, 563, 945
- IP_MULTICAST_LOOP socket option (IP_MULTICAST_LOOP 套接字选项), 193, 559, 563
- IP_MULTICAST_TTL socket option (IP_MULTICAST_TTL 套接字选项), 193, 215, 559, 563, 871, 945
- IP_OPTIONS socket option (IP_OPTIONS 套接字选项), 193, 214, 709–710, 718, 733, 945
- IP_RECVSTADDR socket option (IP_RECVSTADDR 套接字选项), 193, 211, 214, 251, 265, 392–396, 587–588, 590, 592, 608, 616, 620, 666, 895
- ancillary data, picture of (IP_RECVSTADDR 套接字选项作为辅助数据的图示), 394
- IP_RECVIF socket option (IP_RECVIF 套接字选项), 193, 215, 395, 487, 588, 590, 592, 608, 620, 666
- ancillary data, picture of (IP_RECVIF 套接字选项作为辅助数据的图示), 591
- IP_TOS socket option (IP_TOS 套接字选项), 193, 215, 870, 895
- IP_TTL socket option (IP_TTL 套接字选项), 193, 215, 218, 755, 761, 871, 895
- IP_UNBLOCK_SOURCE socket option (IP_UNBLOCK_SOURCE 套接字选项), 193, 560
- ip_id member (ip_id 成员), 740, 806
- ip_len member (ip_len 成员), 737, 740, 806
- ip_mreq structure (ip_mreq 结构), 193, 560, 568
- definition of (ip_mreq 结构定义), 560
- ip_mreq_source structure (ip_mreq_source 结构), 193
- definition of (ip_mreq_source 结构定义), 562
- ip_off member (ip_off 成员), 737, 740
- IPC (interprocess communication), 411–412, 545–547, 675
- ipi6_addr member (ipi6_addr 成员), 616
- ipi6_ifindex member (ipi6_ifindex 成员), 616
- ipi_addr member (ipi_addr 成员), 588, 901
- ipi_ifindex member (ipi_ifindex 成员), 588, 901
- IPng (Internet Protocol next generation), 871
- ipopt_dst member (ipopt_dst 成员), 714
- ipopt_list member (ipopt_list 成员), 714
- ipoption structure, definition of (ipoption 结构及其定义), 714
- IPPROTO_ICMP constant (IPPROTO_ICMP 常值), 736
- IPPROTO_ICMPV6 constant (IPPROTO_ICMPV6 常值), 193, 216, 738, 740
- IPPROTO_IP constant (IPPROTO_IP 常值), 214,

- 394–395, 591, 710
IPTTRO_IPV6 constant (常值), 216, 395, 615–619, 722, 727
IPTTRO_RAW constant (**IPTTRO_RAW常值**), 737
IPTTRO_SCTP constant (**IPTTRO_SCTP常值**), 97, 222, 288
IPTTRO_TCP constant (**IPTTRO_TCP常值**), 97, 219, 288, 519
IPTTRO_UDP constant (**IPTTRO_UDP常值**), 97
IPsec, 951
IPv4 (Internet Protocol version 4), 33, 869
 address (IPv4地址), 874–877
 and IPv6 interoperability (IPv4与IPv6之间的互操作性), 353–362
 checksum (IPv4校验和), 214, 737, 753, 871
 client IPv6 server, interoperability (IPv4客户与IPv6服务器之间的互操作性), 354–357
 destination address (IPv4目的地址), 871
 fragment offset field (IPv4片段偏移字段), 871
 header (IPv4首部), 743, 755, 869–871
 header length field (IPv4首部长度字段), 870
 header, picture of (IPv4首部图示), 870
 identification field (IPv4标识字段), 870
 multicast address (IPv4多播地址), 549–551
 multicast address, ethernet mapping, picture of (IPv4多播地址到以太网地址的映射图示), 550
 options (IPv4选项), 214, 709–711, 871
 protocol field (IPv4协议字段), 871
 receiving packet information (IPv4接收分组信息), 588–593
 server, interoperability, IPv6 client (IPv6客户与IPv4服务器之间的互操作性), 357–359
 socket address structure (IPv4套接字地址结构), 68–70
 socket option (IPv4套接字选项), 214–215
 source address (IPv4源地址), 871
 source routing (IPv4源路由), 711–719
 total length field (IPv4总长度字段), 870
IPv4-compatible IPv6 address (IPv4兼容的IPv6地址), 880
IPv4/IPv6 host, definition of (IPv4/IPv6主机定义), 34
IPv4-mapped IPv6 address (IPv4映射的IPv6地址), 93, 322, 333, 354–360, 745, 879–880
IPv6 (Internet Protocol version 6), 33, 871
 address (IPv6地址), 877–881
 backbone (IPv6主干), 见6bone
 checksum (IPv6校验和), 216, 738, 873
 client IPv4 server, interoperability (IPv6客户与IPv4服务器之间的互操作性), 357–359
 destination address (IPv6目的地址), 873
 destination options (IPv6目的地选项), 719–725
 extension headers (IPv6扩展首部), 719
 flow label field (IPv6流标签字段), 871
 getaddrinfo function (适合IPv6的getaddrinfo函数), 322–323
 header (IPv6首部), 744, 755, 871–874
 header, picture of (IPv6首部图示), 872
 historical advanced API (IPv6历史性高级API), 732
 hop-by-hop options (IPv6步跳选项), 719–725
 interoperability, IPv4 and (IPv4与IPv6之间的互操作性), 353–362
 multicast address (IPv6多播地址), 551–552
 multicast address, ethernet mapping, picture of (IPv6多播地址到以太网地址的映射图示), 550
 multicast address, picture of (IPv6多播地址图示), 551
 next header field (IPv6下一个首部字段), 872
 options (IPv6选项), 见IPv6, extension headers
 path MTU control (IPv6路径MTU控制), 618–619
 payload length field (IPv6净荷长度字段), 872
 receiving packet information (IPv6接收分组信息), 615–618
 routing header (IPv6路由首部), 725–731
 server, interoperability, IPv4 client (IPv4客户与IPv6服务器之间的互操作性), 354–357
 socket address structure (IPv6套接字地址结构), 71–72
 socket option (IPv6套接字选项), 216–218
 source address (IPv6源地址), 873
 source routing (IPv6源路由), 725–731
 source routing segments left (IPv6剩余源路由网段), 725
 source routing type (IPv6源路由类型), 725
 sticky options (IPv6粘附的选项), 731–732
IPV6_ADD_MEMBERSHIP socket option (**IPV6_ADD_MEMBERSHIP**套接字选项), 560–561
IPV6_ADDRFORM socket option (**IPV6_ADDRFORM**套接字选项), 361
IPV6_CHECKSUM socket option (**IPV6_CHECKSUM**套接字选项), 193, 216, 738
IPV6_DONTFRAG socket option (**IPV6_DONTFRAG**套接字选项), 216, 619
IPV6_DROP_MEMBERSHIP socket option (**IPV6_DROP_MEMBERSHIP**套接字选项), 560–561
IPV6_DSTOPTS socket option (**IPV6_DSTOPTS**套接字选项), 193, 395, 732
 ancillary data, picture of (**IPV6_DSTOPTS**套接字选项作为辅助数据的图示), 722
IPV6_HOPLIMIT socket option (**IPV6_HOPLIMIT**套接字选项), 193, 395, 617, 732, 749–750, 873
 ancillary data, picture of (**IPV6_HOPLIMIT**套接字选项作为辅助数据的图示), 615
IPV6_HOPOPTS socket option (**IPV6_HOPOPTS**套接字选项), 193, 395, 732
 ancillary data, picture of (**IPV6_HOPOPTS**套接字选项作为辅助数据的图示), 722
IPV6_JOIN_GROUP socket option (**IPV6_JOIN_GROUP**套接字选项), 193, 560, 562
IPV6_LEAVE_GROUP socket option (**IPV6_LEAVE_GROUP**套接字选项), 193, 561
IPV6_MULTICAST_HOPS socket option (**IPV6_MULTICAST_HOPS**套接字选项), 193, 559, 563, 617, 873
IPV6_MULTICAST_IF socket option (**IPV6_MULTICAST_IF**套接字选项), 193, 559, 563, 616
IPV6_MULTICAST_LOOP socket option (**IPV6_MULTICAST_LOOP**套接字选项), 193, 559, 563

- I**
- IPV6_NEXTHOP socket option (IPV6_NEXTHOP套接字选项), 193, 217, 395, 617, 732
 - ancillary data, picture of (IPV6_NEXTHOP套接字选项作为辅助数据的图示), 615
 - IPV6_PATHMTU socket option (IPV6_PATHMTU套接字选项), 217, 619
 - IPV6_PKTINFO socket option (IPV6_PKTINFO套接字选项), 193, 251, 395, 561, 608, 616, 620, 666, 732
 - ancillary data, picture of (IPV6_PKTINFO套接字选项作为辅助数据的图示), 615
 - IPV6_PKTOPTIONS socket option (IPV6_PKTOPTIONS套接字选项), 732
 - IPV6_RECVDSTOPTS socket option (IPV6_RECVDSTOPTS套接字选项), 217, 722
 - IPV6_RECVHOPLIMIT socket option (IPV6_RECVHOPLIMIT套接字选项), 217–218, 617, 749, 873
 - IPV6_RECVHOPOPTS socket option (IPV6_RECVHOPOPTS套接字选项), 217, 722
 - IPV6_RECVPATHMTU socket option (IPV6_RECVPATHMTU套接字选项), 216–217, 619
 - IPV6_RECVPKTINFO socket option (IPV6_RECVPKTINFO套接字选项), 217, 616–617, 620
 - IPV6_RECVRTHDR socket option (IPV6_RECVRTHDR套接字选项), 218, 727, 729
 - IPV6_RECVTCLASS socket option (IPV6_RECVTCLASS套接字选项), 218, 618
 - IPV6_RTHDR socket option (IPV6_RTHDR套接字选项), 193, 395, 732
 - ancillary data, picture of (IPV6_RTHDR套接字选项作为辅助数据的图示), 727
 - IPV6_RTHDR_TYPE_0 constant (IPV6_RTHDR_TYPE_0常值), 727
 - IPV6_TCLASS socket option (IPV6_TCLASS套接字选项), 395, 618, 732, 871
 - ancillary data, picture of (IPV6_TCLASS套接字选项作为辅助数据的图示), 615
 - IPV6_UNICAST_HOPS socket option (IPV6_UNICAST_HOPS套接字选项), 193, 218, 617, 755, 761, 873
 - IPV6_USE_MIN_MTU socket option (IPV6_USE_MIN_MTU套接字选项), 218, 618–619
 - IPV6_V6ONLY socket option (IPV6_V6ONLY套接字选项), 218, 357
 - IPV6_XXX socket options (IPV6_XXX套接字选项), 218
 - ipv6_mreq structure (ipv6_mreq结构), 193, 560, 569
 - definition of (ipv6_mreq结构定义), 560
 - ipv6mr_interface member (ipv6mr_interface成员), 560, 569
 - ipv6mr_multiaddr member (ipv6mr_multiaddr成员), 560
 - IPX (Internetwork Packet Exchange), 952
 - IRS (Information Retrieval Service), 306
 - ISO (International Organization for Standardization), 18, 26, 950
 - ISO 8859, 573
 - ISP (Internet service provider), 875
 - iterative server (迭代服务器), 15, 114, 243, 821–822
 - Jackson, A., 721, 952
 - Jacobson, V., 35, 38–39, 44, 571, 596, 598–599, 737, 788, 790, 896, 949–951
 - Jim, J., 285, 953
 - Jinmei, T., 28, 216, 397, 719, 738, 744, 953
 - joinable thread (可汇合线程), 678
 - Josey, A., 25, 27, 950
 - Joy, W. N., 106, 950
 - Juhasz, I., 267, 952
 - jumbo payload length (特大净荷长度), 721
 - jumbogram (特大报), 872
 - Kalla, M., 36, 280, 954
 - KAME, 512
 - SCTP implementation (KAME SCTP实现), 299
 - Karels, M. J., 20, 315, 737, 951
 - Karn, P., 599, 950
 - Karn's algorithm (Karn的算法), 599
 - Karrenberg, D., 876, 952
 - Kashyap, V., 285, 953
 - Katz, D., 550, 710, 950
 - kdump program (kdump程序), 892
 - keep-alive option (保持存活选项), 200–202, 238, 923–924
 - Kent, S. T., 511, 719, 950–951
 - Kernighan, B. W., 12, 910, 951
 - kevent function (kevent函数), 405–406, 408
 - definition of (kevent函数定义), 405
 - kevent structure (kevent结构), 405–406, 408
 - definition of (kevent结构定义), 405
 - key management socket (密钥管理套接字), 511–528
 - Key structure (Key结构), 687–688, 690
 - kill program (kill程序), 141–142, 946
 - Kouvelas, I., 564, 948
 - kqueue function (kqueue函数), 405–406, 408
 - definition of (kqueue函数定义), 405
 - ktrace program (ktrace程序), 891
 - l_fixedpt member (l_fixedpt成员), 580
 - l_len member (l_len成员), 834
 - l_linger member (l_linger成员), 202–203, 237, 462
 - l_onoff member (l_onoff成员), 202–203, 237, 462
 - l_start member (l_start成员), 834
 - l_type member (l_type成员), 834
 - l_whence member (l_whence成员), 834
 - LAN (local area network), 5, 35, 219, 448, 530, 549, 553–556, 579, 596–597, 879, 885, 888
 - Lanciani, D., 98, 238, 951
 - LAST_ACK state (LAST_ACK状态), 41
 - latency, scheduling (调度延迟), 162
 - LDAP (Lightweight Directory Access Protocol), 306
 - leader (头; 长)
 - process group (进程组长), 369
 - session (会话头进程), 369
 - leak, memory (内存空间泄漏), 345
 - Lear, E., 876, 952

- least significant bit (最低有效位), 见LSB
len member (**len成员**), 809, 856
Leres, C., 896
LF (linefeed), 9, 895, 916
Li, T., 874, 949
libnet library (**libnet函数库**), 793
libnet_build_dnsv4 function (**libnet_build_dnsv4函数**), 814
libnet_build_ipv4 function (**libnet_build_ipv4函数**), 814
libnet_build_udp function (**libnet_build_udp函数**), 814
libnet_init function (**libnet_init函数**), 812
libnet_write function (**libnet_write函数**), 814
libpcap library (**libpcap函数库**), 788, 792–793
Lightweight Directory Access Protocol (**轻权目录访问协议**), 见LDAP
lightweight process (**轻权进程**), 675
Lin, H., 267, 952
line buffered standard I/O stream (**行缓冲的标准I/O流**), 402
linefeed (**换行符**), 见LF
linger structure (**linger结构**), 192–193, 921
 definition of (**linger结构定义**), 202
link-local (**链路局部**)
 address (**链路局部地址**), 881
 multicast group (**链路局部多播组**), 551
 multicast scope (**链路局部多播范围**), 552–553
 unicast scope (**链路局部单播范围**), 881
Linux, 20, 22–23, 25, 33, 78, 98, 108, 127, 143, 162, 249, 257, 262, 346, 390, 538, 666, 737, 740, 787, 791–793, 797, 809–810, 815, 940
listen function (**listen函数**), 12–13, 37–38, 45, 101, 104–109, 120, 122, 126, 132, 140, 178, 208, 210, 213, 271, 320, 330, 339, 362, 373, 379, 622, 777, 826, 841, 915, 924
 definition of (**listen函数定义**), 104
LISTEN state (**LISTEN状态**), 41, 104, 126–128, 379, 921
Listen wrapper function, source code (**Listen包裹函数源代码**), 107
listening socket (**监听套接字**), 53, 109
LISTENQ constant (**LISTENQ常值**), 13
 definition of (**LISTENQ常值定义**), 902
LISTENQ environment variable (**LISTENQ环境变量**), 107
little-endian byte order (**小端字节序**), 77
Liu, C., 304, 349, 947
LLADDR macro, definition of (**LLADDR宏定义**), 486
local area network (**局域网**), 见LAN
/local service (**/local服务**), 936
localtime function (**localtime函数**), 685
localtime_r function (**localtime_r函数**), 685
LOG_ALERT constant (**LOG_ALERT常值**), 366
LOG_AUTH constant (**LOG_AUTH常值**), 366
LOG_AUTHPRIV constant (**LOG_AUTHPRIV常值**), 366
LOG_CONS constant (**LOG_CONS常值**), 367
LOG_CRIT constant (**LOG_CRIT常值**), 366
LOG_CRON constant (**LOG_CRON常值**), 366
LOG_DAEMON constant (**LOG_DAEMON常值**), 366, 380
LOG_DEBUG constant (**LOG_DEBUG常值**), 366
LOG_EMERG constant (**LOG_EMERG常值**), 366
LOG_ERR constant (**LOG_ERR常值**), 366, 910
LOG_FTP constant (**LOG_FTP常值**), 366
LOG_INFO constant (**LOG_INFO常值**), 366, 910
LOG_KERN constant (**LOG_KERN常值**), 366
LOG_LOCAL0 constant (**LOG_LOCAL0常值**), 366
LOG_LOCAL1 constant (**LOG_LOCAL1常值**), 366
LOG_LOCAL2 constant (**LOG_LOCAL2常值**), 366
LOG_LOCAL3 constant (**LOG_LOCAL3常值**), 366
LOG_LOCAL4 constant (**LOG_LOCAL4常值**), 366
LOG_LOCAL5 constant (**LOG_LOCAL5常值**), 366
LOG_LOCAL6 constant (**LOG_LOCAL6常值**), 366
LOG_LOCAL7 constant (**LOG_LOCAL7常值**), 366
LOG_LPR constant (**LOG_LPR常值**), 366
LOG_MAIL constant (**LOG_MAIL常值**), 366
LOG_NDELAY constant (**LOG_NDELAY常值**), 367
LOG_NEWS constant (**LOG_NEWS常值**), 366
LOG_NOTICE constant (**LOG_NOTICE常值**), 365–366, 380
LOG_PERROR constant (**LOG_PERROR常值**), 367
LOG_PID constant (**LOG_PID常值**), 367
LOG_SYSLOG constant (**LOG_SYSLOG常值**), 366
LOG_USER constant (**LOG_USER常值**), 366, 370, 379
LOG_UUCP constant (**LOG_UUCP常值**), 366
LOG_WARNING constant (**LOG_WARNING常值**), 366
logger program (**logger程序**), 367
logical interface (**逻辑接口**), 877
login name (**登录名**), 372–373
long-fat pipe (**长胖管道**), 39, 209, 236, 599, 950
 definition of (**长胖管道定义**), 39
loom program (**loom程序**), xxiii
loopback (环回)
 address (**环回地址**), 111, 365, 432, 876, 880
 broadcast (**环回式广播**), 535
 interface (**环回接口**), 23, 792, 799, 809, 876–877
 logical (**逻辑环回**), 535, 564
 multicast (**环回式多播**), 559, 563, 566, 570, 577
 physical (**物理环回**), 535, 564
 routing (**路由环回**), 173, 213, 509
loose source and record route (**宽松的源与记录路径**), 见LSRR
lost datagrams, UDP (**丢失的UDP数据报**), 245–246
lost duplicate (**迷途的重复分组**), 43
LP64, programming model (**LP64编程模型**), 28
LPR, 62
ls program (**ls程序**), 414
LSB (least significant bit), 77
lseek function (**lseek函数**), 159, 400
lsof program (**lsof程序**), 897
LSRR (loose source and record route), 710–712
M_DATA constant (**M_DATA常值**), 855–856, 866

- M_PCPROTO constant** (*M_PCPROTO常值*), 855–856, 860, 865
- M_PROTO constant** (*M_PROTO常值*), 855–856, 860, 863, 865, 867
- MAC** (medium access control), 486, 879
- MacOS X**, 22, 78, 108, 262, 473, 538, 921–922, 940
- mail exchange record, DNS** (DNS邮件交换记录), 见**MX**
- main function** (*main函数*), 825
- main thread** (主线程), 676
- malloc function** (*malloc函数*), 29, 246, 317, 320–321, 345, 425, 508, 536, 666, 684, 687–688, 707, 728
- management information base** (管理信息库), 见**MIB**
- Maslen, T. M.**, 346, 951
- MAX_IPOPTLEN constant** (*MAX_IPOPTLEN常值*), 714
- MAXFILES constant** (*MAXFILES常值*), 455
- maximum segment lifetime** (最长分节生命期), 见**MSL**
- maximum segment size** (最大分节大小), 见**MSS**
- maximum transmission unit** (最大传输单元), 见**MTU**
- maxlen member** (*maxlen成员*), 856
- MAXLINE constant** (*MAXLINE常值*), 7, 92, 592, 899
 definition of (*MAXLINE常值定义*), 902
- MBone** (multicast backbone), 571, 885–887
- MCAST_BLOCK_SOURCE socket option** (*MCAST_BLOCK_SOURCE套接字选项*), 193, 560, 562
- MCAST_JOIN_GROUP socket option** (*MCAST_JOIN_GROUP套接字选项*), 193, 560, 562
- MCAST_JOIN_SOURCE_GROUP socket option** (*MCAST_JOIN_SOURCE_GROUP套接字选项*), 193, 560
- MCAST_LEAVE_GROUP socket option** (*MCAST_LEAVE_GROUP套接字选项*), 193, 560–561
- MCAST_LEAVE_SOURCE_GROUP socket option** (*MCAST_LEAVE_SOURCE_GROUP套接字选项*), 193, 560
- MCAST_UNBLOCK_SOURCE socket option** (*MCAST_UNBLOCK_SOURCE套接字选项*), 193, 560
- mcast_block_source function** (*mcast_block_source函数*), 565–569
 definition of (*mcast_block_source函数定义*), 565
- mcast_get_if function** (*mcast_get_if函数*), 565–569
 definition of (*mcast_get_if函数定义*), 565
- mcast_get_loop function** (*mcast_get_loop函数*), 565–569
 definition of (*mcast_get_loop函数定义*), 565
- mcast_get_ttl function** (*mcast_get_ttl函数*), 565–569
 definition of (*mcast_get_ttl函数定义*), 565
- mcast_join function** (*mcast_join函数*), 561, 565–569, 572, 577, 582
 definition of (*mcast_join函数定义*), 565
 source code (*mcast_join函数源代码*), 567
- mcast_join_source_group function** (*mcast_join_source_group函数*), 565–569
 definition of (*mcast_join_source_group函数定义*), 565
- mcast_leave function** (*mcast_leave函数*), 561, 565–569
 definition of (*mcast_leave函数定义*), 565
- definition of** (*mcast_leave函数定义*), 565
- mcast_leave_source_group function** (*mcast_leave_source_group函数*), 565–569
 definition of (*mcast_leave_source_group函数定义*), 565
- mcast_set_if function** (*mcast_set_if函数*), 565–569, 585
 definition of (*mcast_set_if函数定义*), 565
- mcast_set_loop function** (*mcast_set_loop函数*), 565–569, 577
 definition of (*mcast_set_loop函数定义*), 565
 source code (*mcast_set_loop函数源代码*), 570
- mcast_set_ttl function** (*mcast_set_ttl函数*), 565–569
 definition of (*mcast_set_ttl函数定义*), 565
- mcast_unblock_source function** (*mcast_unblock_source函数*), 565–569
 definition of (*mcast_unblock_source函数定义*), 565
- McCann, J.**, 28, 56, 71, 216, 346–347, 504, 949, 951
- McCanne, S.**, 788, 790, 896, 951
- McDonald, D. L.**, 511, 519, 951
- McKusick, M. K.**, 20, 737, 951
- medium access control** (媒体访问控制), 见**MAC**
- memcmp function** (*memcmp函数*), 80–81, 246
 definition of (*memcmp函数定义*), 81
- memcpy function** (*memcpy函数*), 80–81, 860, 930
 definition of (*memcpy函数定义*), 81
- memmove function** (*memmove函数*), 81, 930
- memory leak** (内存空间泄漏), 345
- memset function** (*memset函数*), 8, 80–81, 901
 definition of (*memset函数定义*), 81
- Mendez, T.**, 529, 952
- message** (消息)
 boundaries (消息边界), 31
 high-priority, STREAMS (流高优先级消息), 183, 854
 normal, STREAMS (流普通消息), 183, 854
 priority band, STREAMS (流优先级带消息), 183, 854
 types, ICMPv4 (ICMPv4消息类型), 883
 types, ICMPv6 (ICMPv6消息类型), 884
 types, STREAMS (流消息类型), 854–855
- message-based interface** (基于消息的接口), 858
- meter function** (*meter函数*), 830
- Metz, C. W.**, 360, 511, 519, 947, 951
- Meyer, D.**, 552–553, 951
- MF** (more fragments flag, IP header), 871
- MIB** (management information base), 496
- Milliken, W.**, 529, 952
- Mills, D. L.**, 579, 951
- minimum link MTU** (最小链路MTU), 55
- minimum reassembly buffer size** (最小重组缓冲区大小), 57
- mkfifo function** (*mkfifo函数*), 421
- mktemp function** (*mktemp函数*), 834
- mmap function** (*mmap函数*), 26, 830, 836
- MODE_CLIENT constant** (*MODE_CLIENT常值*), 582
- modules, STREAMS** (流模块), 852
- Mogul, J. C.**, 56, 875, 951

- monitor mode (监视器模式), 787
 Moore, K., 889, 948
 more fragments flag, IP header (IP首部还有片段标志), 见
 MF
 MORE_flag member (MORE_flag成员), 867
 MORECTL constant (MORECTL常值), 857
 MOREDATA constant (MOREDATA常值), 857
 Moreneau, K., 36, 280, 954
 Moskowitz, B., 876, 952
 most significant bit (最高有效位), 见MSB
 mrouted program (mrouted程序), 735, 886–887
 MRP (multicast routing protocol), 556
 MSB (most significant bit), 77
 MSG_ABORT constant (MSG_ABORT常值), 225, 301
 MSG_ADDR_OVER constant (MSG_ADDR_OVER常值),
 225, 271
 MSG_ANY constant (MSG_ANY常值), 857
 MSG_BAND constant (MSG_BAND常值), 857
 MSG_BCAST constant (MSG_BCAST常值), 391–392
 MSG_CTRUNC constant (MSG_CTRUNC常值), 391–392
 MSG_DONTROUTE constant (MSG_DONTROUTE常值),
 199, 388, 391
 MSG_DONTWAIT constant (MSG_DONTWAIT常值), 388,
 391, 398
 MSG_EOF constant (MSG_EOF常值), 225, 301
 MSG_EOR constant (MSG_EOR常值), 277, 285, 389,
 391–392, 432, 936
 MSG_HIPRI constant (MSG_HIPRI常值), 857
 MSG_MCAST constant (MSG_MCAST常值), 391–392
 MSG_NOTIFICATION constant (MSG_NOTIFICATION
 常值), 225, 277, 279–280, 290, 391–392
 MSG_OOB constant (MSG_OOB常值), 207, 388, 391–392,
 646–648, 650–651, 654, 657, 659, 662
 MSG_PEEK constant (MSG_PEEK常值), 388, 391, 398–399,
 409, 421, 895, 934
 MSG_PR_BUFFER constant (MSG_PR_BUFFER常值),
 225
 MSG_PR_SCTP constant (MSG_PR_SCTP常值), 225
 MSG_TRUNC constant (MSG_TRUNC常值), 391–392, 594
 MSG_UNORDERED constant (MSG_UNORDERED常值),
 225, 629
 MSG_WAITALL constant (MSG_WAITALL常值), 90, 388,
 391, 435
 msg_acrights member (msg_acrights成员),
 390, 421, 425, 427
 msg_acrightslen member (msg_acrightslen
 成员), 390
 msg_control member (msg_control成员), 390–391,
 394–396, 398, 421, 425, 590
 msg_controllen member (msg_controllen成员),
 77, 390–392, 394–396, 398
 msg_flags member (msg_flags member成员), 225,
 277, 280, 285, 389–392, 394, 588, 590, 594, 936
 msg iov member (msg iov成员), 390–391
 msg iovlen member (msg iovlen成员), 390–391
 msg_name member (msg_name成员), 390–391, 394
 msg_namelen member (msg_namelen成员), 77,
 390–391, 394, 590
 msghdr structure (msghdr结构), 77, 277, 389–393, 395,
 398, 421, 428, 588, 590, 594, 601, 729
 definition of (msghdr结构定义), 390
 MSL (maximum segment lifetime), 41, 43–44, 151, 203, 915
 definition of (MSL定义), 43
 MSS (maximum segment size), 42, 57–60, 63, 208, 219, 237,
 895, 914, 920–921
 definition of (MSS定义), 38
 option, TCP (TCP MSS选项), 38
 MTU (maximum transmission unit), 18, 23, 25, 56–57, 59,
 537–538, 595, 737, 772, 874, 884, 914
 definition of (MTU定义), 55
 discovery, path, definition of (路径MTU发现定义), 56
 minimum link (最小链路MTU), 55
 path (路径MTU), 59, 63, 219, 444, 771, 874, 921, 951
 path, definition of (路径MTU定义), 56
 multicast (多播), 549–585
 address (多播地址), 549–553
 address, administratively scoped IPv4 (管理上划分范围
 的IPv4多播地址), 553
 address, ethernet mapping, picture of, IPv4 (IPv4多播地
 址到以太网地址的映射图示), 550
 address, ethernet mapping, picture of, IPv6 (IPv6多播地
 址到以太网地址的映射图示), 550
 address, IPv4 (IPv4多播地址), 549–551
 address, IPv6 (IPv6多播地址), 551–552
 address, picture of, IPv6 (IPv6多播地址图示), 551
 backbone (多播主干), 见MBone
 filtering, imperfect (非完备多播过滤), 555
 group address (多播组地址), 549
 group, all-hosts (所有主机多播组), 550
 group, all-nodes (所有节点多播组), 552
 group, all-routers (所有路由器多播组), 550, 552
 group ID (多播组ID), 549
 group, link-local (链路局部多播组), 551
 group, transient (临时多播组), 551
 group, well-known (众所周知的多播组), 551, 571
 IP fragmentation and (IP分片与多播), 571
 listener done, ICMPv6 (ICMPv6多播收听者结束), 884
 listener query, ICMPv6 (ICMPv6多播收听者查询), 884
 listener report, ICMPv6 (ICMPv6多播收听者汇报), 884
 on WAN (WAN上的多播), 556–558
 routing protocol (多播路由协议), 见MRP
 scope (多播范围), 360, 552–553
 scope, admin-local (管区局部多播范围), 552
 scope, continent-local (大洲局部多播范围), 552
 scope, global (全球多播范围), 552–553
 scope, interface-local (接口局部多播范围), 552–553
 scope, link-local (链路局部多播范围), 552–553
 scope, organization-local (组织机构局部多播范围),
 552–553
 scope, region-local (地区局部多播范围), 552
 scope, site-local (网点局部多播范围), 552–553

- sending and receiving** (多播发送与接收), 575–579
session (多播会话), 553
session, SSM (多播会话SSM), 559
socket option (多播套接字选项), 559–564
versus broadcast (多播与广播对比), 553–556
versus unicast (多播与单播对比), 553
multihomed (多宿的), 52–54, 103, 122, 147, 247–248, 250, 262, 312, 314, 324, 532–533, 561, 582, 786, 796, 877, 925
multihoming (多宿), 31
multiplexor (多路复用器), STREAMS, 852–853
mutex (互斥锁), 697–701
MX (mail exchange record, DNS), 304, 308, 310, 349
my_lock_init function (*my_lock_init* 函数), 833–834, 836
my_lock_release function (*my_lock_release* 函数), 836
my_lock_wait function (*my_lock_wait* 函数), 836
my_open function (*my_open* 函数), 421, 423, 427
my_read function (*my_read* 函数), 92, 692
mycat program (*mycat* 程序), 421–422
mydg_echo function (*mydg_echo* 函数), 609–611
- Nagle algorithm** (Nagle算法), 219–221, 229, 390, 402, 923, 928
 definition of (Nagle算法定义), 219
name server (名字服务器), 305–306, 310, 361, 788, 793–794, 803, 811–812
Narten, T., 551, 879, 949, 951
neighbor advertisement, ICMPv6 (ICMPv6邻居通告), 884 inverse, ICMPv6 (ICMPv6反向邻居通告), 884
neighbor discovery (邻居发现), 881
neighbor solicitation, ICMPv6 (ICMPv6邻居征求), 884 inverse, ICMPv6 (ICMPv6反向邻居征求), 884
Nemeth, E., 38, 951
Net/1, 21, 718
Net/2, 21, 737
Net/3, 21, 388
NET_RT_DUMP constant (NET_RT_DUMP常值), 497
NET_RT_FLAGS constant (NET_RT_FLAGS 常值), 497–498
NET_RT_IFLIST constant (NET_RT_IFLIST常值), 497–500
net_rt_iflist function (*net_rt_iflist* 函数), 500, 502, 505–506, 508
NetBIOS, 952
NetBSD, 20–21
netbuf structure (*netbuf* 结构), 856
<netdb.h> header (*<netdb.h>* 头文件), 308, 315, 348
netent structure (*netent* 结构), 348
<net/if_arp.h> header (*<net/if_arp.h>* 头文件), 481
<net/if_dl.h> header (*<net/if_dl.h>* 头文件), 486
<net/if.h> header (*<net/if.h>* 头文件), 480, 504
<netinet/icmp6.h> header (*<netinet/icmp6.h>* 头文件), 740
<netinet/in.h> header (*<netinet/in.h>* 头文件), 68, 71–72, 83, 103, 120, 616, 619, 736
<netinet/ip_var.h> header (*<netinet/ip_var.h>* 头文件), 714
<netinet/udp_var.h> header (*<netinet/udp_var.h>* 头文件), 499
<net/pfkeyv2.h> header (*<net/pfkeyv2.h>* 头文件), 512
<net/route.h> header (*<net/route.h>* 头文件), 483, 487, 489
Netscape, 452, 461
netstat program (*netstat* 程序), 23–24, 31, 37, 40, 53, 63, 126–128, 141, 151, 237, 248, 258–259, 349, 379, 480, 484–485, 576, 612, 896–897, 917, 926
Netware, 952
network (网络)
 byte order (网络字节序), 69, 79, 82, 110, 152, 311–312, 319, 737–738, 740, 918
interface tap (网络接口龙头), 见NIT
topology, discovering (网络拓扑发现), 23–25
virtual (虚拟网络), 885–889
virtual terminal (网络虚拟终端), 见NVT
Network File System (网络文件系统), 见NFS
Network Information System (网络信息系统), 见NIS
Network News Transfer Protocol (网络新闻传送协议), 见NNTP
Network Provider Interface (网络提供者接口), 见NPI
Network Time Protocol (网络时间协议), 见NTP
new generic socket address structure (新的通用套接字地址结构), 72–73
next header field, IPv6 (IPv6下一个首部字段), 872
next_pcaps function (*next_pcaps* 函数), 808
nfds_t datatype (*nfds_t* 数据类型), 184
NFS (Network File System), 62, 208, 213, 239, 596–597, 789
NI_DGRAM constant (NI_DGRAM常值), 340–341
NI_NAMEREQD constant (NI_NAMEREQD常值), 340, 350
NI_NOFQDN constant (NI_NOFQDN常值), 340–341
NI_NUMERICHOST constant (NI_NUMERICHOST常值), 340–341, 933
NI_NUMERICSCOPE constant (NI_NUMERICSCOPE常值), 340–341
NI_NUMERICSERV constant (NI_NUMERICSERV常值), 340–341, 933
nibble (四位组), 304
Nichols, K., 215, 870–871, 948, 952
Nielsen, H. F., 294, 949
NIS (Network Information System), 306
NIT (network interface tap), 788, 793
NNTP (Network News Transfer Protocol), 62
no operation (无操作), 见NOP
NO_ADDRESS constant (NO_ADDRESS常值), 308
NO_DATA constant (NO_DATA常值), 308
NO_RECOVERY constant (NO_RECOVERY常值), 308
nonblocking (非阻塞)

- accept function (非阻塞accept函数), 461–463
 connect function (非阻塞connect函数), 448–461
 I/O(非阻塞式I/O), 88, 165, 234–235, 388, 398, 435–464,
 468, 665, 669, 671, 919, 945
 I/O model (非阻塞式I/O模型), 155–156
 nonlocal goto (非本地跳转), 543, 803
 NOP (no operation), 709, 711–714, 718, 733
 Nordmark, E., 28, 216, 397, 719, 738, 744, 878, 880, 949,
 952–953
 normal, STREAMS message (流普通消息), 183, 854
 notifications, SCTP (SCTP通知), 625–629
 NPI (Network Provider Interface), 854, 954
 ntohs function (ntohs函数), 79, 152, 918
 definition of (ntohs函数定义), 79
 ntohs function (ntohs函数), 110
 definition of (ntohs函数定义), 79
 NTP (Network Time Protocol), 62, 530, 536, 561, 575, 585,
 665–666, 672, 951
 ntpd function (ntpd函数), 162
 ntpdata structure (ntpdata结构), 580
 ntp.h header (ntp.h头文件), 580
 NVT (network virtual terminal), 916
- O_ASYNC constant (O_ASYNC常值), 234–235, 468, 664,
 669
 O_NONBLOCK constant (O_NONBLOCK常值), 234–235,
 468, 669
 O_RDONLY constant (O_RDONLY常值), 423
 O_SIGIO constant (O_SIGIO常值), 664
 octet, definition of (八位组定义), 80
 one-to-many SCTP interface model (一到多SCTP接口模型), 270–272
 one-to-one SCTP interface model (一到一SCTP接口模型),
 269–270
 Ong, L., 36, 267, 952
 open (打开; 开放)
 active (主动打开), 37–38, 41, 45, 48, 53, 894
 passive (被动打开), 37, 41, 45, 48, 52–53, 894
 shortest path first, routing protocol (开放的最短路径优先
 路由协议), 见OSPF
 simultaneous (同时打开), 40–41
 systems interconnection (开放系统互连), 见OSI
 open function (open函数), 135, 370, 415, 421, 423, 427,
 790, 836
 Open Group, The (开放团体), 27–28, 952
 Open Software Foundation (开放软件基金会), 见OSF
 OPEN_MAX constant (OPEN_MAX常值), 186
 open_output function (open_output函数), 799, 805,
 812
 open_pcap function (open_pcap函数), 799, 801
 OpenBSD, 20–21, 737
 openfile program (openfile程序), 422–424, 427
 openlog function (openlog函数), 365–367, 370, 378
 definition of (openlog函数定义), 367
 operating system (操作系统), 见OS
 OPT_length member (OPT_length成员), 863, 865
 OPT_offset member (OPT_offset成员), 863, 865
 opt_val_str member (opt_val_str成员), 194, 196
 optarg variable (optarg变量), 516
 optarg variable (opterr变量), 516
 optind variable (optind变量), 516
 options (选项)
 IPv4 (IPv4选项), 214, 709–711, 871
 IPv6 (IPv6选项), 见IPv6 extension headers
 Socket (套接字选项), 191–238
 TCP (TCP选项), 38–39
 optopt variable (optopt变量), 516
 organization-local multicast scope (组织机构局部多播范
 围), 552–553
 OS (operating system), 22
 OSF (Open Software Foundation), 27
 OSI (open systems interconnection), 18, 20, 68, 98, 389, 392,
 395, 952
 model (OSI模型), 18–19
 OSPF (open shortest path first, routing protocol), 62, 64, 735,
 914
 Ostermann, S., 360, 947
 Otis, D., 36
 out-of-band (带外)
 data (带外数据), 130, 162, 164–166, 184, 188, 207, 234,
 388, 392, 466, 645–662, 855
 data mark (带外数据标记), 648, 654
 data, TCP (TCP带外数据), 645–653, 661–662
 output (输出)
 SCTP (SCTP输出), 60–61
 TCP (TCP输出), 58–59
 UDP (UDP输出), 59–60
 owner, socket (套接字属主), 234–236, 649, 664, 669
 oxymoron (矛盾修饰法), 597
- packet (分组)
 information, IPv4 receiving (IPv4接收分组信息),
 588–593
 information, IPv6 receiving (IPv6接收分组信息),
 615–618
 too big, ICMP (ICMP分组过大错误), 56, 771, 884
 PACKET_ADD_MEMBERSHIP socket option (PACKET_ADD_
 MEMBERSHIP套接字选项), 792
 PACKET_MR_PROMISC socket option (PACKET_MR_PROMISC
 套接字选项), 792
 parallel programming (并行编程), 698
 parameter problem, ICMP (ICMP参数问题错误), 720,
 883–884
 partial delivery, SCTP (SCTP部分递送), 622–625
 Partridge, C., 35, 255, 529, 599, 721, 753, 947–948, 950, 952
 passive (被动)
 close (被动关闭), 39–41, 47–48
 open (被动打开), 37, 41, 45, 48, 52–53, 894
 socket (被动套接字), 104
 PATH environment variable (PATH环境变量), 23, 113
 path MTU (路径MTU), 59, 63, 219, 444, 771, 874, 921, 951
 definition of (路径MTU定义), 56

- path MTU discovery, definition of (路径MTU发现定义), 56
 pause function (pause函数), 189, 362, 447, 658
PAWS (protection against wrapped sequence numbers), 950
 Paxson, V., 35–36, 56, 208, 280, 948, 952, 954
 payload length field, IPv6 (IPv6净荷长度字段), 872
 pcap_compile function (pcap_compile函数), 789, 801
 pcap_datalink function (pcap_datalink函数), 801, 808
 pcap_lookupdev function (pcap_lookupdev函数), 799
 pcap_lookupnet function (pcap_lookupnet函数), 801
 pcap_next function (pcap_next函数), 808–809
 pcap_open_live function (pcap_open_live函数), 799, 809
 pcap_pkthdr structure (pcap_pkthdr结构), 808
 definition of (pcap_pkthdr结构定义), 809
 pcap_setfilter function (pcap_setfilter函数), 801, 809
 pcap_stats function (pcap_stats函数), 811
 _PC_SOCK_MAXBUF constant (_PC_SOCK_MAXBUF常值), 209
 pending error (待处理错误), 165, 199
 perfect filtering (完备过滤), 555
 Perkins, C., 571, 949
 Perkinson, M., 420
 perror function (perror函数), 370
 persistent connection (持续连接), 825
 PF_KEY constant (PF_KEY常值), 511–512
 PF_PACKET constant (PF_PACKET常值), 791–793
 pfmod STREAMS module (pfmod流模块), 790
 Phan, B. G., 511, 519, 951
 PID (process ID), 135, 234–236, 369, 467, 742
 piggybacking (捎带), 42
 PII (Protocol Independent Interfaces), 27
 Pike, R., 12, 951
 ping program (ping程序), 25, 33, 62, 169, 209, 237, 265, 585, 733, 925, 945
 implementation (ping程序实现), 741–754
 ping.h header (ping.h头文件), 742
 Pink, S., 255, 952
 pipe function (pipe函数), 415, 421
 pipe, long-fat (长胖管道), 39, 209, 236, 599, 950
 pkey structure (pkey结构), 687–688, 690
 Plauger, P. J., 399, 952
 pointer record, DNS (DNS指针记录), 见PTR
 Point-to-Point Protocol (点到点协议), 见PPP
 poll function (poll函数), 142, 145, 151, 153–154, 156, 163, 168, 182–187, 189, 320, 402–403, 409, 662, 770, 943
 definition of (poll函数定义), 182
 POLLERR constant (POLLERR常值), 183–184, 188
 pollfd structure (pollfd结构), 183, 185–186, 403–404
 definition of (pollfd结构定义), 183
 <poll.h> header (<poll.h>头文件), 184
 POLLHUP constant (POLLHUP常值), 183
 POLLIN constant (POLLIN常值), 183
 polling (轮询), 156, 161, 702
 POLLNVAL constant (POLLNVAL常值), 183
 POLLOUT constant (POLLOUT常值), 183
 POLLPRI constant (POLLPRI常值), 183
 POLLRDBAND constant (POLLRDBAND常值), 183
 POLLRDNORM constant (POLLRDNORM常值), 183, 186, 188
 POLLWRBAND constant (POLLWRBAND常值), 183
 POLLWRNORM constant (POLLWRNORM常值), 183
port (端口)
 chargen (字符生成服务端口), 61, 189, 349, 380, 930, 934
 daytime (时间获取服务端口), 61–62
 discard (丢弃服务端口), 61
 dynamic (动态端口), 51
 echo (回射服务端口), 61–62, 380
 ephemeral (临时端口), 50–51, 53–54, 87, 99, 101–103, 111, 120, 122, 245–246, 250, 262, 341, 416, 613, 769, 772, 779, 915
 mapper, RPC (RPC端口映射器), 102
 mirroring (端口镜像), 787
 numbers (端口号), 50–52
 numbers and concurrent server (端口号与并发服务器), 52–55
 private (私用端口), 51
 registered (经注册端口), 51, 122
 reserved (保留端口), 51–52, 101, 111, 122, 213
 stealing (端口盗用), 212, 350
 time (流逝时间获取服务端口), 61
 unreachable, ICMP (ICMP端口不可达错误), 249, 253, 257, 265, 534, 755, 761, 764, 771, 794, 815, 883–884, 925
 well-known (众所周知的端口), 50
Portable Operating System Interface (可移植操作系统接口), 见POSIX
 POSIX, 26–27, 68–69, 75, 79, 98–99, 106, 120, 130, 133, 140, 153–154, 158–160, 162, 173, 181, 183–186, 202, 209, 234–235, 252–253, 315, 322, 346, 369, 390, 397, 411–412, 414–415, 421, 436, 448, 463, 465, 467, 516, 536, 539, 541, 543, 594, 654, 663–664, 669–670, 679, 685, 687, 705, 775, 833, 930
 POSIX.1, 685, 919, 950
 definition of (POSIX.1定义), 26
 POSIX.1b, 26, 950
 POSIX.1c, 26, 676, 950
 POSIX.1g, 27–29
 definition of (POSIX.1g定义), 27
 POSIX.1i, 26, 950
 POSIX.2, 26, 28
 Postel, J. B., 34–35, 50–51, 213, 869, 875, 879, 882, 949, 951–953
 PPP (Point-to-Point Protocol), 55, 497, 808
 pr_cpu_time function (pr_cpu_time函数), 824, 827
 prefix length (前缀长度), 874

- preforked server** (预先派生子进程的服务器)
distribution of connections to children, TCP (TCP预先派生子进程的服务器中连接在子进程中的分布), 830–831, 835
select function collisions, TCP (TCP预先派生子进程的服务器中的select函数冲突), 831–832
TCP (TCP预先派生子进程的服务器), 826–842
too many children, TCP (TCP预先派生子进程的服务器中过多子进程的影响), 830, 834
prethreaded server, TCP (TCP预先创建线程的服务器), 844–849
prifinfo program (prifinfo程序), 484, 500
PRIM_type member (PRIM_type成员), 860, 862–863, 865, 867
print_notification function (print_notification函数), 628
print_sadb_msg function (print_sadb_msg函数), 516, 522, 527
printf function, calling from signal handler (从信号处理函数中调用printf函数), 133
priority band, STREAMS message (优先级带流消息), 183, 854
private address (私用地址), 876
private port (私用端口), 51
proc structure (proc结构), 829
proc_v4 function (proc_v4函数), 747–749
proc_v6 function (proc_v6函数), 747, 749–750
process (进程)
 daemon (守护进程), 363–380
 group ID (进程组ID), 234–236, 368, 467
 group leader (进程组长), 369
 ID (进程ID), 见PID
 lightweight (轻权进程), 675
programming model (编程模型)
 ILP32 (ILP32编程模型), 28
 LP64 (LP64编程模型), 28
promiscuous, mode (混杂模式), 555, 787, 790, 792, 799–800
protection against wrapped sequence numbers (针对序列号回绕的保护措施), 见PAWS
proto structure (proto结构), 743, 745, 755, 757
protocol (协议)
 application (应用协议), 4, 421
 byte-stream (字节流协议), 9, 31, 34, 93, 98, 392, 415, 435, 661
 dependence (协议相关性), 10, 244
 field, IPv4 (IPv4协议字段), 871
 independence (协议无关性), 10–11, 244
 usage by common applications (常见因特网应用的协议使用), 62
Protocol Independent Interfaces (协议无关接口), 见PII
protoent structure (protoent结构), 348
ps program (ps程序), 127, 129, 137
pselect function (pselect函数), 153, 181–182, 185, 188, 541, 543, 704
 definition of (pselect函数定义), 181
 source code (pselect函数源代码), 543
 pseudoheader (伪首部), 216, 738, 806
Pthread structure (Pthread结构), 687–688
PTHREAD_MUTEX_INITIALIZER constant (PTHREAD_MUTEX_INITIALIZER常值), 700, 834, 836
Pthread_mutex_lock wrapper function, source code (Pthread_mutex_lock包裹函数源代码), 12
PTHREAD_PROCESS_PRIVATE constant (PTHREAD_PROCESS_PRIVATE常值), 836
PTHREAD_PROCESS_SHARED constant (PTHREAD_PROCESS_SHARED常值), 835–836
pthread_attr_t datatype (pthread_attr_t数据类型), 677
pthread_cond_broadcast function (pthread_cond_broadcast函数), 704
 definition of (pthread_cond_broadcast函数定义), 704
pthread_cond_signal function (pthread_cond_signal函数), 704, 847
 definition of (pthread_cond_signal函数定义), 702
pthread_cond_t datatype (pthread_cond_t数据类型), 702
pthread_cond_timedwait function (pthread_cond_timedwait函数), 704
 definition of (pthread_cond_timedwait函数定义), 704
pthread_cond_wait function (pthread_cond_wait函数), 703–704, 706, 847
 definition of (pthread_cond_wait函数定义), 702
pthread_create function (pthread_create函数), 676–679, 681, 683, 842
 definition of (pthread_create函数定义), 677
pthread_detach function (pthread_detach函数), 676–679
 definition of (pthread_detach函数定义), 678
pthread_exit function (pthread_exit函数), 676–679
 definition of (pthread_exit函数定义), 678
pthread_getspecific function (pthread_getspecific函数), 688, 691–693
 definition of (pthread_getspecific函数定义), 691
pthread_join function (pthread_join函数), 676–679, 696, 701, 705–706
 definition of (pthread_join函数定义), 677
pthread_key_create function (pthread_key_create函数), 687–688, 690–691
 definition of (pthread_key_create函数定义), 690
pthread_key_t datatype (pthread_key_t数据类型), 691
pthread_mutexattr_t datatype (pthread_mutexattr_t数据类型), 836
pthread_mutex_init function (pthread_mutex_init函数), 700, 836
pthread_mutex_lock function (pthread_mutex_lock函数), 845
 definition of (pthread_mutex_lock函数定义), 700
pthread_mutex_t datatype (pthread_mutex_t数据类型), 700, 834, 836

- pthread_mutex_unlock function** (pthread_mutex_unlock函数), 704, 845
definition of (pthread_mutex_unlock函数定义), 700
pthread_once function (pthread_once函数), 688, 690–692
definition of (pthread_once函数定义), 690
pthread_once_t datatype (pthread_once_t数据类型), 691
pthread_self function (pthread_self函数), 676–679
definition of (pthread_self函数定义), 678
pthread_setspecific function (pthread_setspecific函数), 688, 691, 693
definition of (pthread_setspecific函数定义), 691
pthread_t datatype (pthread_t数据类型), 677
<pthread.h> header (<pthread.h>头文件), 679, 694
PTR (pointer record, DNS), 304, 310, 331
Pusateri, T., 550, 952
putc_unlocked function (putc_unlocked函数), 685
putchar_unlocked function (putchar_unlocked函数), 685
putmsg function (putmsg函数), 852, 855–857, 860, 863, 867–868, 891
definition of (putmsg函数定义), 856
putpmsg function (putpmsg函数), 855, 857, 868
definition of (putpmsg函数定义), 857
- QSIZE constant** (QSIZE常值), 666
Quarterman, J. S., 20, 737, 951
queue (队列)
 completed connection (已完成连接队列), 104
 incomplete connection (未完成连接队列), 104
 STREAMS (流队列), 854
queued data (已排队数据), 398–399
queueing, signal (信号排队), 132, 138, 670–671
- race condition** (竞争状态), 237, 384, 538–547, 921
 definition of (竞争状态定义), 538
Rago, S. A., 851, 854–855, 952
Rajahalme, J., 871, 952
Ramakrishnan, K., 215, 870–871, 948, 952
Ramalho, M., 285, 953
rand function (rand函数), 685
rand_r function (rand_r函数), 685
RARP (Reverse Address Resolution Protocol), 34, 787, 789–790
raw socket (原始套接字), 18, 31, 62, 97, 214–216, 411, 485, 492, 495, 735–786, 788, 791, 793–794, 805–807, 809, 884, 945
 creating (创建原始套接字), 736
 input (原始套接字输入), 739–741
 output (原始套接字输出), 737–738
read function (read函数), 7, 9, 11, 29–30, 88, 90, 92–93, 117, 123, 126, 134–135, 159, 167, 171, 174–175, 177, 180, 184, 188, 200–201, 205–206, 210, 240–241, 252–253, 256–257, 265, 381–382, 387–390, 395, 399–400, 408–409, 425, 429, 432, 435, 437, 439–441, 451, 458–459, 490, 492, 545, 650, 655–657, 665, 789–790, 809–810, 841, 852, 854, 856, 892, 914, 919, 923–924, 935–936
read_cred function (read_cred函数), 429
read_fd function (read_fd函数), 424–425, 428, 779, 841
 source code (read_fd函数源代码), 426
readable_conn function (readable_conn函数), 778–779
readable_listen function (readable_listen函数), 777–778
readable_timeo function (readable_timeo函数), 385
 source code (readable_timeo函数源代码), 385
readable_v4 function (readable_v4函数), 781–782
readable_v6 function (readable_v6函数), 784
readdir function (readdir函数), 685
readdir_r function (readdir_r函数), 685
readline function (readline函数), 88–93, 121, 125–126, 128, 133, 142, 144–145, 151, 168–169, 172, 188, 288, 680, 686, 688, 690–693, 707, 843, 899, 916, 919, 921, 923
 definition of (readline函数定义), 88
 source code (readline函数源代码), 90–91, 693
readline_destructor function (readline_destructor函数), 691, 707
readline_once function (readline_once函数), 691–692, 707
readlinebuf function (readlinebuf函数), 92
readline.c function (readline.c函数), 92
readloop function (readloop函数), 746, 752
readn function (readn函数), 88–93, 149–150, 388, 435, 918
 definition of (readn函数定义), 88
 source code (readn函数源代码), 89
readv function (readv函数), 210, 381, 389–391, 395, 408, 435
 definition of (readv函数定义), 389
realloc function (realloc函数), 623
Real-time Transport Protocol (实时传输协议), 见RTP
reassembly (重组), 56, 870, 883–884, 914, 926
 buffer size, minimum (最小重组缓冲区大小), 57
rebooting of server host, crashing and (服务器主机崩溃并重启), 144–145
rec structure (rec结构), 755
receive timeout, BPF (BPF接收超时), 789
receiving sender credentials (接收发送者凭证), 429–431
record boundaries (记录边界), 9, 34, 93, 206, 415–416, 935
record route (记录路由), 711
recv function (recv函数), 90, 210, 241, 252, 381, 387–391, 395, 399, 408–409, 435, 594, 647, 650–651, 657–659, 662
 definition of (recv函数定义), 387
recv_all function (recv_all函数), 577

- `recv_v4` function (`recv_v4`函数), 761–762, 765
`recv_v6` function (`recv_v6`函数), 761, 765
`recvfrom` function (`recvfrom`函数), 68, 75, 134, 155–160, 210, 239–241, 243–249, 251–252, 256, 264–265, 307, 320, 335, 340, 350, 356, 359–361, 382–386, 388–391, 395, 399, 408, 419, 435, 536, 539, 541, 543–545, 574, 577, 582, 588, 590, 592, 594, 599, 601, 611, 614, 647, 664, 671–672, 761, 763, 765, 769, 792, 809–810, 924–926, 934, 945
definition of (`recvfrom`函数定义), 240
with a timeout (带超时的`recvfrom`函数), 383–386
`recvfrom_flags` function (`recvfrom_flags`函数), 588–589, 592–593
`recvmsg` function (`recvmsg`函数), 68, 76–77, 210, 214–218, 224–225, 241, 251–252, 271, 277, 280, 285, 381, 389–395, 397, 408, 421, 425, 429, 435, 561, 588, 590, 592, 594, 601, 603–604, 615–619, 647, 722, 727, 729, 731–733, 936, 941
definition of (`recvmsg`函数定义), 390
receiving destination IP address (`recvmsg`函数接收目的IP地址), 588–593
receiving flags (`recvmsg`函数接收标志), 588–593
receiving interface index (`recvmsg`函数接收接口索引), 588–593
redirect, ICMP (ICMP重定向), 485, 497, 883–884
re-entrant (可重入), 83, 86, 92, 133, 341–346, 684–685
reference count, descriptor (描述符引用计数), 117, 421
region-local multicast scope (地区局部多播范围), 552
registered port (经注册端口), 51, 122
Rekhter, Y., 876, 952
reliable datagram service (可靠数据报服务), 597–608
remote procedure call (远程过程调用), 见RPC
remote terminal protocol (远程终端协议), 见Telnet
rename function (`rename`函数), 366
Request for Comments (请求评注), 见RFC
RES_length member (RES_length成员), 865
RES_offset member (RES_offset成员), 865
RES_USE_INET6 constant (RES_USE_INET6常值), 346
res_init function (`res_init`函数), 349
reserved port (保留端口), 51–52, 101, 111, 122, 213
reset flag, TCP header (TCP首部复位标志), 见RST
resolver (解析器), 305–306, 317, 346, 359–360, 362, 597, 879–880, 933
resource discovery (资源发现), 530
resource record, DNS (DNS资源记录), 见RR
retransmission (重传)
ambiguity problem, definition of (重传二义性问题定义), 598
timeout (重传超时), 见RTO
revents member (revents成员), 183–185, 403
Reverse Address Resolution Protocol(反向地址解析协议), 见RARP
rewind function (`rewind`函数), 400
Reynolds, J. K., 50–51, 953
RFC (Request for Comments), 34, 914, 947
768 (RFC 768), 34, 952
791 (RFC 791), 869, 952
792 (RFC 792), 882, 952
793 (RFC 793), 35, 213, 952
862 (RFC 862), 61
863 (RFC 863), 61
864 (RFC 864), 61
867 (RFC 867), 61
868 (RFC 868), 61
950 (RFC 950), 875, 951
1071 (RFC 1071), 753, 948
1108 (RFC 1108), 950
1112 (RFC 1112), 550, 564, 949
1122 (RFC 1122), 43, 237, 247, 532, 576, 589, 877, 948
1185 (RFC 1185), 44, 950
1191 (RFC 1191), 56, 951
1305 (RFC 1305), 579, 951
1323 (RFC 1323), 35, 38–39, 236, 497, 599, 885, 950
1337 (RFC 1337), 203, 948
1349 (RFC 1349), 215, 870, 948
1390 (RFC 1390), 550, 950
1469 (RFC 1469), 550, 952
1519 (RFC 1519), 874, 949
1546 (RFC 1546), 529, 952
1700 (RFC 1700), 50–51, 953
1812 (RFC 1812), 772, 948
1832 (RFC 1832), 150, 953
1886 (RFC 1886), 304, 954
1918 (RFC 1918), 876, 952
1981 (RFC 1981), 56, 951
2026 (RFC 2026), 28, 948
2030 (RFC 2030), 579, 951
2113 (RFC 2113), 710, 950
2133 (RFC 2133), 361, 949
2140 (RFC 2140), 294, 954
2292 (RFC 2292), 732, 953
2327 (RFC 2327), 571, 949
2365 (RFC 2365), 552–553, 951
2367 (RFC 2367), 511, 519, 951
2401 (RFC 2401), 511, 951
2402 (RFC 2402), 719, 951
2406 (RFC 2406), 719, 951
2409 (RFC 2409), 524, 949
2428 (RFC 2428), 360, 947
2460 (RFC 2460), 55, 216, 721, 726, 871, 873, 949
2463 (RFC 2463), 882, 948
2464 (RFC 2464), 551, 948
2467 (RFC 2467), 551, 948
2470 (RFC 2470), 551, 949
2471 (RFC 2471), 879, 949
2474 (RFC 2474), 215, 870–871, 948, 952
2553 (RFC 2553), 346–347, 949
2581 (RFC 2581), 35, 208, 948
2675 (RFC 2675), 57, 721, 948
2711 (RFC 2711), 721, 952
2719 (RFC 2719), 267, 952

- 2765 (RFC 2765), 880, 952
 2893 (RFC 2893), 880, 949
 2960 (RFC 2960), 36, 280, 954
 2974 (RFC 2974), 571, 949
 2988 (RFC 2988), 35, 952
 3041 (RFC 3041), 879, 951
 3056 (RFC 3056), 889, 948
 3068 (RFC 3068), 889, 950
 3152 (RFC 3152), 304, 948
 3168 (RFC 3168), 215, 870–871, 948, 952
 3232 (RFC 3232), 50, 953
 3286 (RFC 3286), 36, 952
 3306 (RFC 3306), 551, 949
 3307 (RFC 3307), 552, 949
 3309 (RFC 3309), 36
 3376 (RFC 3376), 564, 948
 3390 (RFC 3390), 35, 947
 3484 (RFC 3484), 317, 949
 3493 (RFC 3493), 28, 71, 216, 346–347, 504, 949
 3513 (RFC 3513), 529, 877–879, 949
 3542 (RFC 3542), 28, 216, 397, 719, 738, 744, 953
 3587 (RFC 3587), 878, 949
 Host Requirements (主机要求RFC), 948
 obtaining (获取RFC), 914
 RIP (Routing Information Protocol, routing protocol), 57, 62, 535
 Ritchie, D. M., 851, 910, 951, 953
rl_cnt member (*rl_cnt*成员), 693
rl_key function (*rl_key*函数), 691
rl_once function (*rl_once*函数), 691
rlim_cur member (*rlim_cur*成员), 919
rlim_max member (*rlim_max*成员), 919
 RLIMIT_NOFILE constant (RLIMIT_NOFILE常值), 919
 Rline structure (Rline结构), 691–693
 Rlogin, 219–220, 308, 661–662
rlogin program (*rlogin*程序), 52
rlogind program (*rlogind*程序), 718–719, 733, 945
 road map, client/server examples (客户/服务器例子导读图), 16–18
 Rose, M. T., 315
 round robin, DNS (DNS顺序循环), 822
 round-trip time (往返时间), 见RTT
route program (*route*程序), 234, 483
 routed program (*routed* 程序), 199, 481, 530, 535
 router (路由器), 5
 advertisement, ICMP (ICMP路由器通告), 735, 741, 883–884
 alert (路由器告警), 721
 solicitation, ICMP (ICMP路由器征求), 735, 883–884
 routing (路由)
 header, IPv6 (IPv6路由首部), 725–731
 hop count (路由跳数), 481
 IP (IP路由), 869
 socket (路由套接字), 485–509
 socket, datalink socket address structure (路由套接字、数据链路套接字地址结构), 486–487
 socket, reading and writing (路由套接字读写), 487–495
 socket, sysctl operations (路由套接字sysctl操作), 495–499
 table operations, ioctl function (ioctl函数路由表操作), 483–484
 Routing Information Protocol, routing protocol (路由信息协议之路由协议), 见RIP
 RPC (remote procedure call), 102, 150, 372, 597
 DCE (DCE RPC), 62
 port mapper (RPC端口映射器), 102
 Sun (Sun RPC), 9, 62
 RR (resource record, DNS), 304–305
rresvport function (*rresvport*函数), 52
 RS_HIPRI constant (RS_HIPRI常值), 856–857, 860
rsh program (*rsh*程序), 44, 52, 312, 340
rshd program (*rshd*程序), 718–719
 RST (reset flag, TCP header), 44, 99–101, 107, 140, 142–143, 145, 167, 179, 184, 188–189, 200, 202–203, 207, 236, 256, 462–463, 789, 794, 916, 921, 938
rt_msghdr structure (*rt_msghdr*结构), 487, 490–492
 definition of (*rt_msghdr*结构定义), 488
 RTA_AUTHOR constant (RTA_AUTHOR常值), 489
 RTA_BRD constant (RTA_BRD常值), 489
 RTA_DST constant (RTA_DST常值), 489–490
 RTA_GATEWAY constant (RTA_GATEWAY常值), 489
 RTA_GENMASK constant (RTA_GENMASK常值), 489
 RTA_IFA constant (RTA_IFA常值), 489
 RTA_IFF constant (RTA_IFF常值), 489
 RTA_NETMASK constant (RTA_NETMASK常值), 489
 RTAX_AUTHOR constant (RTAX_AUTHOR常值), 489
 RTAX_BRD constant (RTAX_BRD常值), 489
 RTAX_DST constant (RTAX_DST常值), 489
 RTAX_GATEWAY constant (RTAX_GATEWAY常值), 489
 RTAX_GENMASK constant (RTAX_GENMASK常值), 489
 RTAX_IFA constant (RTAX_IFA常值), 489
 RTAX_IFF constant (RTAX_IFF常值), 489, 506
 RTAX_MAX constant (RTAX_MAX常值), 489, 493
 RTAX_NETMASK constant (RTAX_NETMASK常值), 489
rtentry structure (*rtentry*结构), 467, 483
 RTF_LLINFO constant (RTF_LLINFO常值), 497–498
 RTM_ADD constant (RTM_ADD常值), 487
 RTM_CHANGE constant (RTM_CHANGE常值), 487
 RTM_DELADDR constant (RTM_DELADDR常值), 487
 RTM_DELETE constant (RTM_DELETE常值), 487
 RTM_DELMADDR constant (RTM_DELMADDR常值), 487
 RTM_GET constant (RTM_GET常值), 487, 489–490, 497
 RTM_IFANNOUNCE constant (RTM_IFANNOUNCE常值), 487
 502, 505, 508
 RTM_IFINFO constant (RTM_IFINFO常值), 487, 498, 502, 505, 508
 RTM_LOCK constant (RTM_LOCK常值), 487
 RTM_LOSING constant (RTM_LOSING常值), 487
 RTM_MISS constant (RTM_MISS常值), 487
 RTM_NEWADDR constant (RTM_NEWADDR常值), 487, 498, 502
 RTM_NEWMADDR constant (RTM_NEWMADDR常值), 487

- RTM_REDIRECT constant (RTM_REDIRECT常值), 487
 RTM_RESOLVE constant (RTM_RESOLVE常值), 487
`rtm_addrs` member (`rtm_addrs`成员), 489–490, 492–493
`rtm_type` member (`rtm_type`成员), 490
 RTO (retransmission timeout), 598–599, 604, 606–607
 RTP (Real-time Transport Protocol), 575
 RTT (round-trip time), 35, 105–106, 169–170, 209, 220, 237, 436, 445, 447, 461, 595, 597–608, 620, 742, 745, 749–750, 762, 923
 RTT_RTOCALC macro (RTT_RTOCALC宏), 606
`rtt_info` structure (`rtt_info`结构), 601
`rtt_init` function (`rtt_init`函数), 601, 606
 source code (`rtt_init`函数源代码), 605
`rtt_minmax` function (`rtt_minmax`函数), 606
 source code (`rtt_minmax`函数源代码), 605
`rtt_newpack` function (`rtt_newpack`函数), 603, 606
 source code (`rtt_newpack`函数源代码), 606
`rtt_start` function (`rtt_start`函数), 603, 607
 source code (`rtt_start`函数源代码), 606
`rtt_stop` function (`rtt_stop`函数), 604, 607
 source code (`rtt_stop`函数源代码), 607
`rtt_timeout` function (`rtt_timeout`函数), 604, 607
 source code (`rtt_timeout`函数源代码), 607
`rtt_ts` function (`rtt_ts`函数), 603–604, 606, 941
 source code (`rtt_ts`函数源代码), 606
 Rubin, A. D., 108, 711, 948
 RUSAGE_CHILDREN constant (RUSAGE_CHILDREN常值), 824
 RUSAGE_SELF constant (RUSAGE_SELF常值), 824
 Rytina, I., 36, 267, 280, 285, 952–954

`s6_addr` member (`s6_addr`成员), 71
 SA (security association), 511
 SA macro (SA宏), 9, 71
`s_addr` member (`s_addr`成员), 68–69
`s_aliases` member (`s_aliases`成员), 311
`s_fixedpt` member (`s_fixedpt`成员), 580
`s_name` member (`s_name`成员), 311
`s_port` member (`s_port`成员), 311
`s_proto` member (`s_proto`成员), 311
 SA_INTERRUPT constant (SA_INTERRUPT常值), 131
 SA_RESTART constant (SA_RESTART常值), 131, 134, 162, 383
`sa_data` member (`sa_data`成员), 70, 482, 792
`sa_family` member (`sa_family`成员), 70–71, 482, 490, 494
`sa_family_t` datatype (`sa_family_t`数据类型), 69
`sa_handler` member (`sa_handler`成员), 131
`sa_len` member (`sa_len`成员), 70, 493–494
`sa_mask` member (`sa_mask`成员), 131–132
`sac_info` member (`sac_info`成员), 282
 SACK (selective acknowledgment), 61
 SADB (security association database), 511
 SADB_AALG_MD5HMAC constant (SADB_AALG_MD5HMAC常值), 518
 SADB_AALG_NONE constant (SADB_AALG_NONE常值), 518
 SADB_AALG_SHA1HMAC constant (SADB_AALG_SHA1HMAC常值), 518
 SADB_ACQUIRE constant (SADB_ACQUIRE常值), 513
 SADB_ADD constant (SADB_ADD常值), 513, 519, 522
 SADB_DELETE constant (SADB_DELETE常值), 513
 SADB_DUMP constant (SADB_DUMP常值), 513
 SADB_EALG_3DESCBC constant (SADB_EALG_3DESCBC常值), 518
 SADB_EALG_DESCBC constant (SADB_EALG_DESCBC常值), 518
 SADB_EALG_NONE constant (SADB_EALG_NONE常值), 518, 521
 SADB_EALG_NULL constant (SADB_EALG_NULL常值), 518
 SADB_EXPIRE constant (SADB_EXPIRE常值), 513, 523
 SADB_EXT_ADDRESS_DST constant (SADB_EXT_ADDRESS_DST常值), 514, 519, 522
 SADB_EXT_ADDRESS_PROXY constant (SADB_EXT_ADDRESS_PROXY常值), 514, 519
 SADB_EXT_ADDRESS_SRC constant (SADB_EXT_ADDRESS_SRC常值), 514, 519, 522
 SADB_EXT_IDENTITY_DST constant (SADB_EXT_IDENTITY_DST常值), 514
 SADB_EXT_IDENTITY_SRC constant (SADB_EXT_IDENTITY_SRC常值), 514
 SADB_EXT_KEY_AUTH constant (SADB_EXT_KEY_AUTH常值), 514, 519, 522
 SADB_EXT_KEY_ENCRYPT constant (SADB_EXT_KEY_ENCRYPT常值), 514, 519
 SADB_EXT_LIFETIME_CURRENT constant (SADB_EXT_LIFETIME_CURRENT常值), 514
 SADB_EXT_LIFETIME_HARD constant (SADB_EXT_LIFETIME_HARD常值), 514
 SADB_EXT_LIFETIME_SOFT constant (SADB_EXT_LIFETIME_SOFT常值), 514
 SADB_EXT_PROPOSAL constant (SADB_EXT_PROPOSAL常值), 514
 SADB_EXT_SA constant (SADB_EXT_SA常值), 514
 SADB_EXT_SENSITIVITY constant (SADB_EXT_SENSITIVITY常值), 514
 SADB_EXT_SPIRANGE constant (SADB_EXT_SPIRANGE常值), 514
 SADB_EXT_SUPPORTED_AUTH constant (SADB_EXT_SUPPORTED_AUTH常值), 514
 SADB_EXT_SUPPORTED_ENCRYPT constant (SADB_EXT_SUPPORTED_ENCRYPT常值), 514
 SADB_FLUSH constant (SADB_FLUSH常值), 513
 SADB_GET constant (SADB_GET常值), 513
 SADB_GETSPI constant (SADB_GETSPI常值), 513
 SADB_LIFETIME_CURRENT constant (SADB_LIFETIME_CURRENT常值), 523
 SADB_LIFETIME_HARD constant (SADB_LIFETIME_HARD常值), 523
 SADB_LIFETIME_SOFT constant (SADB_LIFETIME_SOFT常值), 523

- 常值), 523
SADB_REGISTER constant (SADB_REGISTER常值), 513
SADB_SAFLAGS_PFS constant (SADB_SAFLAGS_PFS常值), 519
SADB_SASTATE_DEAD constant (SADB_SASTATE_DEAD常值), 518
SADB_SASTATE_DYING constant (SADB_SASTATE_DYING常值), 518
SADB_SASTATE_LARVAL constant (SADB_SASTATE_LARVAL常值), 518
SADB_SASTATE_MATURE constant (SADB_SASTATE_MATURE常值), 518, 521
SADB_SATYPE_AH constant (SADB_SATYPE_AH常值), 513-514
SADB_SATYPE_ESP constant (SADB_SATYPE_ESP常值), 513-514, 524
SADB_SATYPE_MIP constant (SADB_SATYPE_MIP常值), 513
SADB_SATYPE_OSPFV2 constant (SADB_SATYPE_OSPFV2常值), 513
SADB_SATYPE_RIPV2 constant (SADB_SATYPE_RIPV2常值), 513-514
SADB_SATYPE_RSVP constant (SADB_SATYPE_RSVP常值), 513
SADB_UPDATE constant (SADB_UPDATE常值), 513
sadb_address structure (sadb_address结构), 514, 519
 definition of (sadb_address结构定义), 519
sadb_address_exttype member (sadb_address_exttype成员), 519
sadb_address_len member (sadb_address_len成员), 519
sadb_address_prefixlen member (sadb_address_prefixlen成员), 519
sadb_address_proto member (sadb_address_proto成员), 519
sadb_address_reserved member (sadb_address_reserved成员), 519
sadb_alg structure (sadb_alg结构), 524
 definition of (sadb_alg结构定义), 524
sadb_alg_id member (sadb_alg_id成员), 524
sadb_alg_ivlen member (sadb_alg_ivlen成员), 524
sadb_alg_maxbits member (sadb_alg_maxbits成员), 524
sadb_alg_minbits member (sadb_alg_minbits成员), 524
sadb_dump function (sadb_dump函数), 516
sadb_ident structure (sadb_ident结构), 514
sadb_key structure (sadb_key结构), 514, 519
 definition of (sadb_key结构定义), 519
sadb_key_bits member (sadb_key_bits成员), 519
sadb_key_exttype member (sadb_key_exttype成员), 519
sadb_keylen member (sadb_keylen成员), 519
sadb_lifetime structure (sadb_lifetime结构), 514
 definition of (sadb_lifetime结构定义), 523
sadb_lifetime_addtime member (sadb_lifetime_addtime成员), 523
sadb_lifetime_allocations member (sadb_lifetime_allocations成员), 523
sadb_lifetime_bytes member (sadb_lifetime_bytes成员), 523
sadb_lifetime_exttype member (sadb_lifetime_exttype成员), 523
sadb_lifetime_len member (sadb_lifetime_len成员), 523
sadb_lifetime_usetime member (sadb_lifetime_usetime成员), 523
sadb_msg structure (sadb_msg结构), 512
 definition of (sadb_msg结构定义), 513
sadb_msg_errno member (sadb_msg_errno成员), 513
sadb_msg_len member (sadb_msg_len成员), 513, 521
sadb_msg_pid member (sadb_msg_pid成员), 513
sadb_msg_reserved member (sadb_msg_reserved成员), 513
sadb_msg_satype member (sadb_msg_satype成员), 513
sadb_msg_seq member (sadb_msg_seq成员), 513
sadb_msg_type member (sadb_msg_type成员), 512-513
sadb_msg_version member (sadb_msg_version成员), 513
sadb_prop structure (sadb_prop结构), 514
sadb_sa structure (sadb_sa结构), 514, 517
 definition of (sadb_sa结构定义), 518
sadb_sa_auth member (sadb_sa_auth成员), 518
sadb_sa_encrypt member (sadb_sa_encrypt成员), 518
sadb_sa_exttype member (sadb_sa_exttype成员), 518
sadb_sa_flags member (sadb_sa_flags成员), 518
sadb_sa_len member (sadb_sa_len成员), 518
sadb_sa_replay member (sadb_sa_replay成员), 518
sadb_sa_reply member (sadb_sa_reply成员), 518
sadb_sa_spi member (sadb_sa_spi成员), 518, 521
sadb_sa_state member (sadb_sa_state成员), 518
sadb_sens structure (sadb_sens结构), 514
sadb_spirange structure (sadb_spirange结构), 514
sadb_supported structure (sadb_supported结构), 514, 524
 definition of (sadb_supported结构定义), 524

- sadb_supported_exttype member (sadb_supported_exttype成员), 524
 sadb_supported_len member (sadb_supported_len成员), 524
 Salus, P. H., 30, 953
 sanity check (理智检查), 536
 SAP (Session Announcement Protocol), 571, 573–574
 sasoc_asocmaxrxt member (sasoc_asocmaxrxt成员), 222–223, 639
 sasoc_assoc_id member (sasoc_assoc_id成员), 222–223
 sasoc_cookie_life member (sasoc_cookie_life成员), 222–223
 sasoc_local_rwnd member (sasoc_local_rwnd成员), 222–223
 sasoc_number_peer_destinations member (sasoc_number_peer_destinations成员), 222–223
 sasoc_peer_rwnd member (sasoc_peer_rwnd成员), 222–223
 scatter read (分散读), 389
 scheduling latency (调度延迟), 162
 Schimmel, C., 830, 953
 Schwartz, A., 15, 949
 Schwarzbauer, H., 36, 267, 280, 952, 954
 SCM_CREDS socket option (SCM_CREDS套接字选项), 395
 ancillary data, picture of (SCM_CREDS套接字选项作为辅助数据的图示), 397
 SCM_RIGHTS socket option (SCM_RIGHTS套接字选项), 395
 ancillary data, picture of (SCM_RIGHTS套接字选项作为辅助数据的图示), 397
 scope (范围)
 admin-local multicast (管区局部多播范围), 552
 continent-local multicas (大洲局部多播范围), 552
 global multicast (全球多播范围), 552–553
 global unicast (全球单播范围), 878
 interface-local multicast (接口局部多播范围), 552–553
 link-local multicast (链路局部多播范围), 552–553
 link-local unicast (链路局部单播范围), 881
 multicast (多播范围), 360, 552–553
 organization-local multicast (组织结构局部多播范围), 552–553
 region-local multicast (地区局部多播范围), 552
 site-local multicast (网点局部多播范围), 552–553
 site-local unicast (网点局部单播范围), 881
 _SC_OPEN_MAX constant (_SC_OPEN_MAX常值), 186
 script program (script程序), 699
 SCTP (Stream Control Transmission Protocol), 33, 36–37
 address information (SCTP地址信息), 631–635
 association autoclose (SCTP关联自动关闭), 621–622
 connection establishment (SCTP连接建立), 44–50
 connection termination (SCTP连接终止), 44–50
 four-way handshake (SCTP四路握手), 45–46
 heartbeat mechanism (SCTP心搏机制), 636–637
 implementation, KAME (KAME SCTP实现), 299
 interface model, converting (SCTP接口模型转换), 637–639
 interface model, one-to-many (一到多SCTP接口模型), 270–272
 interface model, one-to-one (一到一SCTP接口模型), 269–270
 interface models (SCTP接口模型), 268–272
 introduction, TCP, UDP, and (TCP, UDP与SCTP简介), 31–64
 notifications (SCTP通知), 625–629
 output (SCTP输出), 60–61
 partial delivery (SCTP部分递送), 622–625
 performance tuning (SCTP性能调整), 639–641
 socket (SCTP套接字), 267–286, 621–643
 socket option (SCTP套接字选项), 222–233
 state transition diagram (SCTP状态转换图), 47–49
 unordered data (SCTP无序的数据), 629
 versus TCP (SCTP与TCP对比), 641–642
 watching the packets (SCTP观察分组), 49
 SCTP_ACTIVE constant (SCTP_ACTIVE常值), 227
 SCTP_ADAPTION_INDICATION constant (SCTP_ADAPTION_INDICATION常值), 285
 SCTP_ADAPTION_LAYER socket option (SCTP_ADAPTION_LAYER套接字选项), 194, 222, 285
 SCTP_ADDR_ADDED constant (SCTP_ADDR_ADDED常值), 283
 SCTP_ADDR_AVAILABLE constant (SCTP_ADDR_AVAILABLE常值), 283
 SCTP_ADDR_CONFIRMED constant (SCTP_ADDR_CONFIRMED常值), 283
 SCTP_ADDR_MADE_PRIM constant (SCTP_ADDR_MADE_PRIM常值), 283
 SCTP_ADDR_REMOVED constant (SCTP_ADDR_REMOVED常值), 283
 SCTP_ADDR_UNCONFIRMED constant (SCTP_ADDR_UNCONFIRMED常值), 227
 SCTP_ADDR_UNREACHABLE constant (SCTP_ADDR_UNREACHABLE常值), 283
 SCTP_ASSOC_CHANGE constant (SCTP_ASSOC_CHANGE常值), 281
 SCTP_ASSOCINFO socket option (SCTP_ASSOCINFO套接字选项), 222–223
 SCTP_AUTOCLOSE socket option (SCTP_AUTOCLOSE套接字选项), 194, 223, 622
 SCTP_BINDX_ADD_ADDR constant (SCTP_BINDX_ADD_ADDR常值), 273–274
 SCTP_BINDX_REMOVE_ADDR constant (SCTP_BINDX_REMOVE_ADDR常值), 273–274
 SCTP_CANT_STR_ASSOC constant (SCTP_CANT_STR_ASSOC常值), 282
 SCTP_CLOSED constant (SCTP_CLOSED常值), 233
 SCTP_COMM_LOST constant (SCTP_COMM_LOST常值), 282
 SCTP_COMM_UP constant (SCTP_COMM_UP常值), 281, 633
 SCTP_COOKIE_ECHOED constant (SCTP_COOKIE_ECHOED常值), 233

- SCTP_COOKIE_WAIT constant (SCTP_COOKIE_WAIT常值), 233
- SCTP_DATA_SENT constant (SCTP_DATA_SENT常值), 284
- SCTP_DATA_UNSENT constant (SCTP_DATA_UNSENT常值), 284
- SCTP_DEFAULT_SEND_PARAM socket option (SCTP_DEFAULT_SEND_PARAM套接字选项), 194, 224–225
- SCTP_DISABLE_FRAGMENTS socket option (SCTP_DISABLE_FRAGMENTS套接字选项), 194, 225
- SCTP_ESTABLISHED constant (SCTP_ESTABLISHED常值), 233
- SCTP_EVENTS socket option (SCTP_EVENTS套接字选项), 194, 225–226, 271, 277, 280–281
- SCTP_GET_PEER_ADDR_INFO socket option (SCTP_GET_PEER_ADDR_INFO套接字选项), 194, 222, 226–227
- SCTP_INACTIVE constant (SCTP_INACTIVE常值), 227
- SCTP_INITMSG socket option (SCTP_INITMSG套接字选项), 194, 228
- SCTP_ISSUE_HB constant (SCTP_ISSUE_HB常值), 230, 636–637
- SCTP_I_WANT_MAPPED_V4_ADDR socket option (SCTP_I_WANT_MAPPED_V4_ADDR套接字选项), 194, 227
- SCTP_MAXBURST socket option (SCTP_MAXBURST套接字选项), 194, 228
- SCTP_MAXSEG socket option (SCTP_MAXSEG套接字选项), 57, 194, 229, 233, 236, 269, 928
- SCTP_NODELAY socket option (SCTP_NODELAY套接字选项), 194, 229, 236, 269, 928
- SCTP_NO_HB constant (SCTP_NO_HB常值), 230, 636–637
- SCTP_PARTIAL_DELIVERY_ABORTED constant (SCTP_PARTIAL_DELIVERY_ABORTED常值), 286
- SCTP_PARTIAL_DELIVERY_EVENT constant (SCTP_PARTIAL_DELIVERY_EVENT常值), 285
- SCTP_PEER_ADDR_CHANGE constant (SCTP_PEER_ADDR_CHANGE常值), 282
- SCTP_PEER_ADDR_PARAMS socket option (SCTP_PEER_ADDR_PARAMS套接字选项), 194, 222, 229–230, 635
- SCTP_PRIMARY_ADDR socket option (SCTP_PRIMARY_ADDR套接字选项), 194, 222, 230
- SCTP_REMOTE_ERROR constant (SCTP_REMOTE_ERROR常值), 283
- SCTP_RESTART constant (SCTP_RESTART常值), 282, 633
- SCTP_RTOINFO socket option (SCTP_RTOINFO套接字选项), 194, 222, 230–231
- SCTP_SEND_FAILED constant (SCTP_SEND_FAILED常值), 284
- SCTP_SET_PEER_ADDR_PARAMS socket option (SCTP_SET_PEER_ADDR_PARAMS套接字选项), 201
- SCTP_SET_PEER_PRIMARY_ADDR socket option (SCTP_SET_PEER_PRIMARY_ADDR套接字选项), 194, 231–32
- SCTP_SHUTDOWN_ACK_SENT constant (SCTP_SHUTDOWN_ACK_SENT常值), 233
- SCTP_SHUTDOWN_COMP constant (SCTP_SHUTDOWN_COMP常值), 282
- SCTP_SHUTDOWN_EVENT constant (SCTP_SHUTDOWN_EVENT常值), 284
- SCTP_SHUTDOWN_PENDING constant (SCTP_SHUTDOWN_PENDING常值), 233
- SCTP_SHUTDOWN_RECEIVED constant (SCTP_SHUTDOWN_RECEIVED常值), 233
- SCTP_SHUTDOWN_SENT constant (SCTP_SHUTDOWN_SENT常值), 233
- SCTP_STATUS socket option (SCTP_STATUS套接字选项), 194, 222, 232–233, 278, 290
- sctp_adaption_event structure, definition of (sctp_adaption_event结构定义), 285
- sctp_adaption_layer_event member (sctp_adaption_layer_event成员), 226
- sctp_address_event member (sctp_address_event成员), 226
- sctp_assoc_change structure, definition of (sctp_assoc_change结构定义), 281
- sctp_association_event member (sctp_association_event成员), 226
- sctp_assocparams structure (sctp_assocparams结构), 222
 - definition of (sctp_assocparams结构定义), 222
 - sctp_assoc_t datatype (sctp_assoc_t数据类型), 271
- sctp_bind_arg_list function (sctp_bind_arg_list函数), 630–631
- sctp_bindx function (sctp_bindx函数), 272–274, 286, 630–631
 - definition of (sctp_bindx函数定义), 272
- sctp_connectx function (sctp_connectx函数), 274, 286
 - definition of (sctp_connectx函数定义), 274
- sctp_data_io_event member (sctp_data_io_event成员), 226, 271, 277, 280, 288
- sctp_event_subscribe structure (sctp_event_subscribe结构), 225–226
 - definition of (sctp_event_subscribe结构定义), 226
- sctp_freeladdrs function (sctp_freeladdrs函数), 276, 633
 - definition of (sctp_freeladdrs函数定义), 276
- sctp_freepaddrs function (sctp_freepaddrs函数), 275, 633
 - definition of (sctp_freepaddrs函数定义), 275
- sctp_getladdrs function (sctp_getladdrs函数), 275–276, 286, 633–634
 - definition of (sctp_getladdrs函数定义), 275
- sctp_get_no_strms function (sctp_get_no_strms函数), 290
- sctp_getpaddrs function (sctp_getpaddrs函数), 275, 286, 633–634
 - definition of (sctp_getpaddrs函数定义), 275

- sctp_initmsg structure (sctp_initmsg结构), 228, 299
 definition of (sctp_initmsg结构定义), 228
- sctp_notification structure, definition of (sctp_notification结构定义), 281
- sctp_opt_info function (sctp_opt_info函数), 222, 225–226, 230, 232, 278, 635
- sctp_paddr_change structure, definition of (sctp_paddr_change结构定义), 283
- sctp_paddrinfo structure (sctp_paddrinfo结构), 226
 definition of (sctp_paddrinfo结构定义), 226
- sctp_paddrparams structure (sctp_paddrparams结构), 229, 637
 definition of (sctp_paddrparams结构定义), 229
- sctp_partial_delivery_event member (sctp_partial_delivery_event成员), 226
- sctp_pdapi_event structure, definition of (sctp_pdapi_event结构定义), 286
- sctp_peeloff function (sctp_peeloff函数), 271–272, 286, 638–639, 643, 926–927
- sctp_peer_error_event member (sctp_peer_error_event成员), 226
- sctp_print_addresses function (sctp_print_addresses函数), 633–634
- sctp_recvmsg function (sctp_recvmsg函数), 224–225, 271, 277, 280, 285–286, 288, 623, 927
- sctp_remote_error structure, definition of (sctp_remote_error结构定义), 283
- sctp_rtoinfo structure (sctp_rtoinfo结构), 230
 definition of (sctp_rtoinfo结构定义), 231
- sctp_send_failed structure, definition of (sctp_send_failed结构定义), 284
- sctp_send_failure_event member (sctp_send_failure_event成员), 226
- sctp_sendmsg function (sctp_sendmsg函数), 224–225, 271, 276–277, 286, 288, 293, 295, 298, 301, 927
 definition of (sctp_sendmsg函数定义), 276
- sctp_sendto function (sctp_sendto函数), 271
- sctp_setpeerprim structure (sctp_setpeerprim结构), 231
 definition of (sctp_setpeerprim结构定义), 231
- sctp_setprim structure (sctp_setprim结构), 230
 definition of (sctp_setprim结构定义), 230
- sctp_shutdown_event member (sctp_shutdown_event成员), 226
- sctp_shutdown_event structure, definition of (sctp_shutdown_event结构定义), 285
- sctp_sndrcvinfo structure (sctp_sndrcvinfo结构), 224–225, 271, 277, 280, 288, 290, 292, 300, 642
 definition of (sctp_sndrcvinfo结构定义), 224
- sctp_status structure (sctp_status结构), 232
 definition of (sctp_status结构定义), 232
- sctp_tlv structure, definition of (sctp_tlv结构定义), 280
- sctpstr_cli function (sctpstr_cli函数), 290, 294, 629, 632
- sctpstr_cli_echoall function (sctpstr_cli_echoall函数), 290, 294
- sdl_alen member (sdl_alen成员), 486, 502, 939
- sdl_data member (sdl_data成员), 486–487
- sdl_family member (sdl_family成员), 486
- sdl_index member (sdl_index成员), 486
- sdl_len member (sdl_len成员), 486, 509
- sdl_nlen member (sdl_nlen成员), 486, 502, 939
- sdl_slen member (sdl_slen成员), 486
- sdl_type member (sdl_type成员), 486
- SDP (Session Description Protocol), 571, 573–575
- sdr program (sdr程序), 571
- secure shell (安全shell), 见SSH
- security_association (安全关联), 见SA
 association database (安全关联数据库), 见SADB
 association database, dumping (倾泻安全关联数据库), 514–517
- association, dynamic (动态安全关联), 524–528
- association, static (静态安全关联), 517–523
- parameters index (安全参数索引), 见SPI
- policy database (安全策略数据库), 见SPDB
- SEEK_SET constant (SEEK_SET常值), 834
- segleft member (segleft成员), 727
- segment, TCP (TCP分节), 35
- select function (select函数), 76, 91, 134–135, 141–142, 145, 151, 153–154, 156–157, 160–169, 171–175, 177–185, 188–189, 199, 201–202, 209–210, 248, 262–263, 320, 364, 373, 375–377, 381–382, 385, 400, 402–406, 408–409, 437, 439–440, 445–446, 448–449, 451–452, 456–459, 461–463, 545, 547, 587, 606, 612, 614, 620, 647–648, 651–652, 655, 657, 661–662, 679, 694, 704, 770, 773–774, 777, 780, 817, 819, 831–832, 838, 841, 850, 919, 924, 938–939, 941
- collisions, TCP preforked server (TCP预先派生子进程服务器的select函数冲突), 831–832
- definition of (select函数定义), 161
- maximum number of descriptors (select函数最大描述符数目), 166–167
- TCP and UDP server (TCP与UDP服务器的select函数), 262–264
- when is a descriptor ready (select函数描述字就绪条件), 164–166
- selective acknowledgment (选择性确认), 见SACK
- send function (send函数), 199, 210, 241, 252, 269, 271, 381, 387–389, 391, 395, 399, 408, 432, 435, 646, 648, 660, 662, 736–737, 936
 definition of (send函数定义), 387
- send_all function (send_all函数), 577
- send_dns_query function (send_dns_query函数), 803, 812, 814
- send_v4 function (send_v4函数), 752, 754
- send_v6 function (send_v6函数), 752, 754
- sendmail program (sendmail程序), 349, 363, 377

- sendmsg function (sendmsg函数), 68, 76, 199, 210, 218, 225, 241, 269, 271, 276–277, 282, 300, 381, 389–395, 408, 420–421, 427–430, 435, 588, 601, 603, 615–617, 722, 727, 730–733, 737, 928
 definition of (sendmsg函数定义), 390
- sendto function (sendto函数), 68, 74, 199, 210, 239–241, 243–245, 249–250, 252–253, 255–256, 264–265, 269, 271, 307, 317, 319, 335, 337, 356, 358–359, 382, 390–391, 395, 408, 415, 419, 435, 532, 535–536, 576–577, 599, 601, 611, 669, 736–737, 761, 806, 925
 definition of (sendto函数定义), 240
- SEQ_number member (SEQ_number成员), 865
- sequence number, UDP (UDP序列号), 597
- Sequenced Packet Exchange (有序的分组交换), 见SPX
- Serial Line Internet Protocol (串行线网际网协议), 见SLIP
- SERV_MAX_SCTP_STRM constant (SERV_MAX_SCTP_STRM常值), 294
- SERV_PORT constant (SERV_PORT常值), 122, 125, 189, 242, 288, 599, 608
 definition of (SERV_PORT常值定义), 902
- servent structure (servent结构), 311, 348
 definition of (servent结构定义), 311
- server (服务器)
 concurrent (并发服务器), 15, 114–116
 iterative (迭代服务器), 15, 114, 243, 821–822
 name (名字服务器), 305
 not running, UDP (UDP服务器进程未运行), 248–249
 preforked (预先派生子进程的服务器), 826
 prethreaded (预先创建线程的服务器), 844
 processing time (服务器处理时间), 见SPT
- Services, Differentiated (区分服务), 870–871
- services, standard Internet (标准因特网服务), 61–62, 377, 893
- Session Announcement Protocol (会话声明协议), 见SAP
- session announcements, IP Multicast (IP多播会话声明)
 Infrastructure (IP多播基础设施会话声明), 571–575
- Session Description Protocol (会话描述协议), 见SDP
- session leader (会话头进程), 369
- session, multicast (多播会话), 553
 SSM multicast (SSM多播会话), 559
- setgid function (setgid函数), 373
- setrlimit function (setrlimit函数), 189, 919
- setsid function (setsid函数), 369, 379
- setsockopt function (setsockopt函数), 191–194, 202, 218, 222, 230, 386, 554, 559, 567–570, 710–714, 717–719, 728, 733, 740, 761, 921, 945
 definition of (setsockopt函数), 192
- setuid function (setuid函数), 373, 746, 799
- set-user-ID (SUID), 422, 742, 746, 799
- setvbuf function (setvbuf函数), 402
- Shah, H., 285, 953
- shallow copy (浅度复制), 321
- Sharp, C., 36, 267, 280, 952, 954
- SHUT_RDWR constant (SHUT_RDWR常值), 173, 189, 213, 279, 495, 901
- SHUT_RDWR constant (SHUT_RDWR常值), 173, 189, 280, 901, 919
- SHUT_WR constant (SHUT_WR常值), 173, 175, 205, 279, 901, 919
- shutdown function (shutdown函数), 39, 117, 120, 171–173, 175, 188–189, 205–206, 213, 267, 278–279, 282, 401, 439, 446, 464, 495, 681, 819, 919, 938
 definition of (shutdown函数定义), 173
- shutdown of server host (服务器主机关机), 145
- SHUTDOWN-ACK-SENT state (SHUTDOWN-ACK-SENT状态), 48
- SHUTDOWN-PENDING state (SHUTDOWN-PENDING状态), 47–48
- SHUTDOWN-RECEIVED state (SHUTDOWN-RECEIVED状态), 47–48
- SHUTDOWN-SENT state (SHUTDOWN-SENT状态), 48
- SIG_DFL constant (SIG_DFL常值), 129–130, 935
- SIG_IGN constant (SIG_IGN常值), 129–130, 133, 143
- sig_alarm function(sig_alarm函数), 601, 752, 759, 765, 803
- sig_chld function(sig_chld函数), 133, 138, 263, 823
- sigaction function (sigaction函数), 129–132, 158
- sigaction structure (sigaction结构), 131
- sigaddset function (sigaddset函数), 541, 669
- SIGALRM signal (SIGALRM信号), 131, 342, 381, 383–384, 409, 536, 539, 541, 543, 545, 547, 601, 603, 620, 742, 745, 747, 752, 759, 765, 802–803
- SIGCHLD signal (SIGCHLD信号), 128–130, 132–135, 137–139, 141, 151, 262, 376–377, 446, 614, 823, 946
- sigemptyset function (sigemptyset函数), 541
- Sigfunc datatype (Sigfunc数据类型), 131
- SIGHUP signal (SIGHUP信号), 364, 369–370, 379, 669, 671–672
- SIGINT signal (SIGINT信号), 181–182, 257, 370, 823, 827, 830, 837, 842, 846
- SIGIO signal (SIGIO信号), 129, 157–158, 200, 234–235, 467–468, 663–666, 669–672, 895
- TCP and (TCP与SIGIO信号), 664–665
- UDP and (UDP与SIGIO信号), 664
- SIGKILL signal (SIGKILL信号), 129, 145
- siglongjmp function (siglongjmp函数), 383, 543–545, 601, 603–604, 620, 802–803
- signal (信号), 129–132
 action (信号行为), 129
 blocking (信号阻塞), 131–132, 539, 541, 543, 545, 669–671
 catching (信号俘获), 129
 definition of (信号定义), 129
 delivery (信号递交), 131–133, 137, 539, 541, 545, 669–671, 946
- disposition (信号处置), 129–130, 133, 143, 676
- generation (信号生成), 541
- handler (信号处理函数), 129, 676

- mask (信号掩码), 131, 181–182, 543, 669, 676, 802
queueing (信号排队), 132, 138, 670–671
Signal function (Signal函数), 130
signal function (signal函数), 130–131, 133–134, 137, 383, 664, 935
definition of (signal函数定义), 131
source code (signal函数源代码), 130
signal-driven I/O (信号驱动式I/O), 200, 234–235, 663–673
model (信号驱动式I/O模型), 157–158
SIGPIPE signal (SIGPIPE信号), 142–143, 152, 165, 202, 916–917, 938
SIGPOLL signal (SIGPOLL信号), 129, 663–664
sigprocmask function (sigprocmask函数), 132, 541, 669–670
sigsetjmp function (sigsetjmp函数), 383, 543–545, 601, 603–604, 620, 802–803, 946
SIGSTOP signal (SIGSTOP信号), 129
sigsuspend function (sigsuspend函数), 669
SIGTERM signal (SIGTERM信号), 145, 446–447, 827, 938
SIGURG signal (SIGURG函数), 129–130, 234–235, 467, 647–649, 651, 655, 657–658, 661–662
SIGWINCH signal (SIGWINCH信号), 370
SHT, 880, 952
Simple Mail Transfer Protocol (简单邮件传送协议), 见 SMTP
simple name, DNS (DNS简单名字), 303
Simple Network Management Protocol (简单网络管理协议), 见 SNMP
Simple Network Time Protocol (简单网络时间协议), 见 SNTP
simultaneous (同时)
close (同时关闭), 40–41, 48
connections (同时发起的连接), 452–461
open (同时打开), 40–41
SIN6_LEN constant (SIN6_LEN常值), 69, 71–72
sin6_addr member (sin6_addr成员), 71–72, 102, 480
sin6_family member (sin6_family成员), 71, 254
sin6_flowinfo member (sin6_flowinfo成员), 71–72, 872
sin6_len member (sin6_len成员), 71
sin6_port member (sin6_port成员), 71, 102
sin6_scope_id member (sin6_scope_id成员), 71–72
sin_addr member (sin_addr成员), 68–70, 102, 480
sin_family member (sin_family成员), 68–69, 254
sin_len member (sin_len成员), 68
sin_port member (sin_port成员), 30, 68–69, 102
sin_zero member (sin_zero成员), 68–70
sinfo_assoc_id member (sinfo_assoc_id成员), 224–225
sinfo_context member (sinfo_context成员), 224
sinfo_cumtsn member (sinfo_cumtsn成员), 224
sinfo_flags member (sinfo_flags成员), 224, 300
sinfo_pid member (sinfo_pid成员), 224
sinfo_ppid member (sinfo_ppid成员), 224
sinfo_ssn member (sinfo_ssn成员), 224
sinfo_stream member (sinfo_stream成员), 224, 292
sinfo_timetolive member (sinfo_timetolive成员), 224, 642
sinfo_tsn member (sinfo_tsn成员), 224
sinit_max_attempts member (sinit_max_attempts成员), 228, 639–640
sinit_max_init_timeo member (sinit_max_init_timeo成员), 228, 639–640
sinit_max_instreams member (sinit_max_instreams成员), 228
sinit_max_ostreams member (sinit_max_ostreams成员), 299
sinit_num_ostreams member (sinit_num_ostreams成员), 228
SIOCADDR constant (SIOCADDR常值), 467, 483, 485
SIOCATMARK constant (SIOCATMARK常值), 234, 465–467, 654
SIOCDARP constant (SIOCDARP常值), 467, 482
SIOCDELRT constant (SIOCDELRT常值), 467, 483, 485
SIOC GARP constant (SIOC GARP常值), 467, 482
SIOCGIFADDR constant (SIOCGIFADDR常值), 467, 480, 566, 568
SIOCGIFBRDADDR constant (SIOCGIFBRDADDR常值), 467, 478, 481, 484
SIOCGIFCONF constant (SIOCGIFCONF常值), 234, 467–469, 474–475, 478, 480, 484, 500, 799
SIOCGIFDSTADDR constant (SIOCGIFDSTADDR常值), 467, 478, 481
SIOCGIFFLAGS constant (SIOCGIFFLAGS常值), 467, 477, 480, 792
SIOCGIFMETRIC constant (SIOCGIFMETRIC常值), 467, 481
SIOCGIFMTU constant (SIOCGIFMTU常值), 538
SIOCGIFNETMASK constant (SIOCGIFNETMASK常值), 467, 481
SIOCGIFNUM constant (SIOCGIFNUM常值), 475, 484
SIOCGPGRP constant (SIOCGPGRP常值), 234, 467–468
SIOCGSTAMP constant (SIOCGSTAMP常值), 666
SIOSCARP constant (SIOSCARP常值), 467, 481
SIOCSIFADDR constant (SIOCSIFADDR常值), 467, 480
SIOCSIFBRDADDR constant (SIOCSIFBRDADDR常值), 467, 481
SIOCSIFDSTADDR constant (SIOCSIFDSTADDR常值), 467, 481
SIOCSIFFLAGS constant (SIOCSIFFLAGS常值), 467, 481, 792
SIOCSIFMETRIC constant (SIOCSIFMETRIC常值), 467, 481
SIOCSIFNETMASK constant (SIOCSIFNETMASK常值), 467, 481
SIOCSPGRP constant (SIOCSPGRP常值), 234, 467–468
site-local (网点局部的)
address (网点局部地址), 881
multicast scope (网点局部多播范围), 552–553

- unicast scope (网点局部单播范围), 881
size_t datatype (**size_t**数据类型), 8, 29
sizeof operator (**sizeof**操作符), 9, 412, 862
Sklower, K., 315
sleep function (**sleep**函数), 152, 163, 432, 539, 577, 648, 657, 660, 916, 935
sleep_us function (**sleep_us**函数), 163
SLIP (Serial Line Internet Protocol), 55, 808
slow start (慢启动), 461, 596, 950
Smith, G. P., 325
SMTP (Simple Mail Transfer Protocol), 9, 62, 938
SNA (Systems Network Architecture), 952
sn_header member (**sn_header**成员), 281
sn_type member (**sn_type**成员), 281
SNMP (Simple Network Management Protocol), 57, 62, 239, 496, 597
snoop program (**snoop**程序), 896
snprintf function (**snprintf**函数), 15, 148, 423
SNTP (Simple Network Time Protocol), 579–584, 951
snntp_proc function (**snntp_proc**函数), 582
SO_ACCEPTCON socket option (**SO_ACCEPTCON**套接字选项), 924
SO_ACCEPTCONN socket option (**SO_ACCEPTCONN**套接字选项), 238
SO_ATTACH_FILTER socket option (**SO_ATTACH_FILTER**套接字选项), 792
SO_BROADCAST socket option (**SO_BROADCAST**套接字选项), 193, 198–199, 236, 532, 536, 786, 895, 945
SO_BSDCOMPAT socket option (**SO_BSDCOMPAT**套接字选项), 249
SO_DEBUG socket option (**SO_DEBUG**套接字选项), 193, 198–199, 237, 895, 922
SO_DONTROUTE socket option (**SO_DONTROUTE**套接字选项), 193, 198–199, 388, 617, 895
SO_ERROR socket option (**SO_ERROR**套接字选项), 165, 193, 199–200, 236, 451
SO_KEEPALIVE socket option (**SO_KEEPALIVE**套接字选项), 144–145, 151, 193, 198, 200–202, 236, 238, 895
SO_LINGER socket option (**SO_LINGER**套接字选项), 58, 117, 120, 140, 173, 193, 198, 202–207, 236–237, 282, 462, 895
SO_OOBINLINE socket option (**SO_OOBINLINE**套接字选项), 193, 198, 207, 647–648, 654, 656, 662
SO_RCVBUF socket option (**SO_RCVBUF**套接字选项), 38, 193, 198, 207–209, 236, 243, 260, 623, 895, 925
SO_RCVLOWAT socket option (**SO_RCVLOWAT**套接字选项), 164, 193, 198, 209–210
SO_RCVTIMEO socket option (**SO_RCVTIMEO**套接字选项), 193, 210, 382, 386, 895
SO_REUSEADDR socket option (**SO_REUSEADDR**套接字选项), 103, 193, 203, 210–213, 236–237, 262, 330, 339, 350, 362, 572, 577, 608, 610, 895, 922, 933
SO_REUSEPORT socket option (**SO_REUSEPORT**套接字选项), 103, 193–194, 196, 210–213, 237, 895, 922
SO_SNDBUF socket option (**SO_SNDBUF**套接字选项), 58–60, 193, 198, 207–209, 223, 236, 895, 925
SO SNDLOWAT socket option (**SO SNDLOWAT**套接字选项), 165, 193, 198, 209–210
SO SNDTIMEO socket option (**SO SNDTIMEO**套接字选项), 193, 210, 382, 386, 895
SO_TIMESTAMP socket option (**SO_TIMESTAMP**套接字选项), 666
SO_TYPE socket option (**SO_TYPE**套接字选项), 193, 198, 213
SO_USELOOPBACK socket option (**SO_USELOOPBACK**套接字选项), 173, 193, 213, 509
so_error variable (**so_error**变量), 199–200
so_pgid member (**so_pgid**成员), 235
so_socket function (**so_socket**函数), 892–893
so_timeo structure (**so_timeo**结构), 830
sock program (**sock**程序), 237, 265, 893–895, 925
 options (**sock**程序选项), 895
SOCK_DGRAM constant (**SOCK_DGRAM**常值), 97, 213, 242, 315, 319–320, 414, 791
SOCK_PACKET constant (**SOCK_PACKET**常值), 33, 98, 787, 791–793, 797, 815
SOCK_RAW constant (**SOCK_RAW**常值), 97, 736, 791
SOCK_SEQPACKET constant (**SOCK_SEQPACKET**常值), 97–98, 319
SOCK_STREAM constant (**SOCK_STREAM**常值), 7, 97–98, 198, 213, 319–320, 327, 330, 414–415
sock_bind_wild function (**sock_bind_wild**函数), 86–88, 772, 779
 definition of (**sock_bind_wild**函数定义), 87
sock_cmp_addr function (**sock_cmp_addr**函数), 86–88
 definition of (**sock_cmp_addr**函数定义), 87
sock_cmp_port function (**sock_cmp_port**函数), 86–88
 definition of (**sock_cmp_port**函数定义), 87
sock_get_port function (**sock_get_port**函数), 86–88
 definition of (**sock_get_port**函数定义), 87
sock_masktop function (**sock_masktop**函数), 493–494
sock_ntop function (**sock_ntop**函数), 86–88, 110, 120, 331, 340, 350, 482, 593, 933, 935, 941
 definition of (**sock_ntop**函数定义), 86
 source code (**sock_ntop**函数源代码), 87
sock_ntop_host function (**sock_ntop_host**函数), 86–88, 493, 536
 definition of (**sock_ntop_host**函数定义), 87
sock_opts structure (**sock_opts**结构), 194
sock_set_addr function (**sock_set_addr**函数), 86–88, 932
 definition of (**sock_set_addr**函数定义), 87
sock_set_port function (**sock_set_port**函数), 86–88, 761, 932
 definition of (**sock_set_port**函数定义), 87
sock_set_wild function (**sock_set_wild**函数), 88, 581
 definition of (**sock_set_wild**函数定义), 87

- sock_str_flag function** (sock_str_flag函数), 197
sockaddr structure(sockaddr结构), 9, 71–72, 193, 315, 477, 519, 522
 definition of (sockaddr结构定义), 70
sockaddr_dl structure(sockaddr_dl结构), 489, 508, 591
 definition of (sockaddr_dl结构定义), 486
 picture of (sockaddr_dl结构图示), 73
sockaddr_in structure(sockaddr_in结构), 8, 10, 68, 76, 227, 322, 358, 361, 477, 492, 494, 519, 772, 860, 915
 definition of (sockaddr_in结构定义), 68
 picture of (sockaddr_in结构图示), 73
sockaddr_in6 structure(sockaddr_in6结构), 32, 72, 76, 322, 477, 519, 617, 772, 872
 definition of (sockaddr_in6结构定义), 71
 picture of (sockaddr_in6图示), 73
sockaddr_storage structure (sockaddr_storage结构), 72–73, 120, 330, 561, 567, 772, 779
 definition of (sockaddr_storage结构定义), 72
 picture of (sockaddr_storage结构图示), 73
sockaddr_un structure (sockaddr_un结构), 74, 76, 412, 416, 418–419
 definition of (sockaddr_un结构定义), 412
 picture of (sockaddr_un结构图示), 73
sockargs function (sockargs函数), 68
socketmark function (socketmark函数), 234, 465, 467, 654–660, 662
 definition of (socketmark函数定义), 654
 source code (socketmark函数源代码), 654
socket (套接字)
 active (主动套接字), 104
 address structures (套接字地址结构), 67–74
 address structure, comparison of (套接字地址结构比较), 73–74
 address structure, generic (通用套接字地址结构), 70–71
 address structure, IPv4 (IPv4套接字地址结构), 68–70
 address structure, IPv6 (IPv6套接字地址结构), 71–72
 address structure, new generic (新的通用套接字地址结构), 72–73
 address structure, routing socket, datalink (路由套接字数据链路套接字地址结构), 486–487
 address structure, Unix domain (Unix域套接字地址结构), 412–414
datagram (数据报套接字), 33
 definition of (套接字定义), 8, 52
 introduction (套接字简介), 67–93
 key management (密钥管理套接字), 511–528
 owner (套接字属主), 234–236, 649, 664, 669
 pair, definition of (套接字对定义), 52
 passive (被动套接字), 104
raw (原始套接字), 18, 31, 62, 97, 214–216, 411, 485, 492, 495, 735–786, 788, 791, 793–794, 805–807, 809, 884, 945
receive buffer (套接字接收缓冲区), UDP, 260–261
routing (路由套接字), 485–509
SCTP (SCTP套接字), 267–286, 621–643
stream (字节流套接字), 33
TCP (TCP套接字), 95–120
timeout (套接字超时), 210, 381–386
UDP (UDP套接字), 239–265, 587–620
Unix domain (Unix域套接字), 411–433
Socket wrapper function, source code (Socket包裹函数源代码), 11
socket function (socket函数), 7–8, 10–11, 13, 30, 37–38, 45, 95–99, 101, 104, 109, 115, 120, 126, 140, 178, 210, 236, 242, 272, 275–276, 288, 314, 317, 319–320, 324, 327, 330, 361, 379, 416, 418–419, 421, 717, 736–739, 791–792, 829–831, 891–893, 913, 924, 941
 definition of (socket函数定义), 96
socket option (套接字选项), 191–238
 generic (通用套接字选项), 198–213
ICMPv6 (ICMPv6套接字选项), 216
IPv4 (IPv4套接字选项), 214–215
IPv6 (IPv6套接字选项), 216–218
 multicast (多播套接字选项), 559–564
 obtaining default (获得默认套接字选项), 194–198
SCTP (SCTP套接字选项), 222–233
socket states (影响套接字状态的套接字选项), 198
TCP (TCP套接字选项), 219–221
socketpair function (socketpair函数), 414–415, 420–421, 423, 545
 definition of (socketpair函数定义), 414
sockets and standard I/O (套接字与标准I/O), 399–402
sockets API (套接字API), 8
sockfd_to_family function (sockfd_to_family函数), 119, 569
 source code (sockfd_to_family函数源代码), 119
sockfs filesystem (sockfs文件系统), 892
socklen_t datatype (socklen_t数据类型), 29, 69, 75, 915
sockmod STREAMS module (sockmod流模块), 853, 858
sockproto structure (sockproto结构), 98
sofree function (sofree函数), 140
soft error (软错误), 100
software interrupts (软件中断), 129
SOL_SOCKET constant (SOL_SOCKET常值), 395, 397
 Solaris, 20, 22, 51, 78, 100, 108, 111, 133–134, 169, 248, 257, 262, 306, 343, 345–346, 378, 380, 414, 444, 447, 451, 475, 477, 486, 536, 538, 564, 694–697, 700–701, 705, 718, 735, 772, 774, 793, 806, 818, 832, 836, 841, 892, 896–897, 913, 916, 919–920, 922
 solutions to exercises (习题解答), 913–946
soo_select function (soo_select函数), 165
soreadable function (soreadable函数), 165
sorwakeup function (sorwakeup函数), 664
source address (源地址)
 IPv4 (IPv4源地址), 871
 IPv6 (IPv6源地址), 873
 source code (源代码)

- availability (源代码可获得性), xxii
 conventions (源代码约定), 7
 portability, interoperability (源代码可移植性、互操作性), 361
 source quench, ICMP (ICMP源熄灭), 771–772, 883
 source routing (源路由)
 IPv4 (IPv4源路由), 711–719
 IPv6 (IPv6源路由), 725–731
 source-specific multicast (源特定多播), 见SSM
 sowriteable function (sowriteable函数), 165
 sp_family member (sp_family成员), 98
 sp_protocol member (sp_protocol成员), 98
 Spafford, E. H., 15, 949
 spc_state member (spc_state成员), 283
 SPDB (security policy database), 512
 Spero, S., 294, 953
 SPI (security parameters index), 518
 spinfo_address member (spininfo_address成员), 226–227
 spinfo_assoc_id member (spininfo_assoc_id成员), 226
 spinfo_cwnd member (spininfo_cwnd成员), 226–227
 spinfo_mtu member (spininfo_mtu成员), 226–227
 spinfo_rto member (spininfo_rto成员), 226–227
 spinfo_srtt member (spininfo_srtt成员), 226–227
 spinfo_state member (spininfo_state成员), 226–227
 spoofing, IP (IP欺骗), 108, 948
 spp_address member (spp_address函数), 229
 spp_assoc_id member (spp_assoc_id成员), 229–230
 spp_hbinterval member (spp_hbinterval成员), 229
 spp_hbpathmaxrxt member (spp_hbpathmaxrxt成员), 230
 spp_pathmaxrxt member (spp_pathmaxrxt成员), 229, 639
 sprintf function (sprintf函数), 15
 SPT (server processing time), 595
 SPX (Sequenced Packet Exchange), 952
 Srinivasan, R., 150, 953
 srto_assoc_id member (srto_assoc_id成员), 231
 srto_initial member (srto_initial成员), 231, 639
 srto_max member (srto_max成员), 231, 639
 srto_max_init_timeo member (srto_max_init_timeo成员), 640
 srto_min member (srto_min成员), 231, 639
 ss_family member (ss_family成员), 72–73
 ss_len member (ss_len成员), 72–73
 sscanf function (sscanf函数), 148–149
 SSH (secure shell), 22, 62
 SSM (source-specific multicast), 558–559, 950
 SSM multicast session (SSM多播会话), 559
 SSN (stream sequence number), 224
 ssp_addr member (ssp_addr成员), 230
 ssp_assoc_id member (ssp_assoc_id成员), 230
 sspp_addr member (sspp_addr成员), 230–231
 sspp_assoc_id function (sspp_assoc_id函数), 231
 sspp_assoc_id member (sspp_assoc_id成员), 231
 SSRR (strict source and record route), 710–712
 sstat_assoc_id member (sstat_assoc_id成员), 232
 sstat_fragmentation_point member (sstat_fragmentation_point成员), 232–233
 sstat_instrms member (sstat_instrms成员), 232
 sstat_outstrms member (sstat_outstrms成员), 232–233
 sstat_penddata member (sstat_penddata成员), 232
 sstat_primary member (sstat_primary成员), 232–233
 sstat_rwnd member (sstat_rwnd成员), 232–233
 sstat_state member (sstat_state成员), 232
 sstat_unackdata member (sstat_unackdata成员), 232–233
 Stallman, R. M., 26
 standard Internet services (标准因特网服务), 61–62, 377, 893
 standard I/O (标准I/O), 168, 344, 399–402, 409, 437, 935, 952
 sockets and (套接字与标准I/O), 399–402
 stream (标准I/O流), 399
 stream, fully buffered (完全缓冲的标准I/O流), 401
 stream, line buffered (行缓冲的标准I/O流), 402
 stream, unbuffered (不缓冲的标准I/O流), 402
 standards, Unix (Unix标准I/O标准), 25–28
 start_connect function (start_connect函数), 457–458
 state transition diagram, SCTP (SCTP状态转换图), 47–49
 TCP (TCP状态转换图), 40–41
 static qualifier (static限定词), 92, 342
 stderr constant (stderr常值), 365
 <stdio.h> header (<stdio.h>头文件), 402
 stealing, port (端口盗用), 212, 350
 Stevens, W. R., 28, 35, 71, 208, 216, 346–347, 361, 397, 504, 719, 732, 738, 744, 948–949, 953–954
 Stewart, R. R., 36, 46, 49–50, 61, 203, 227, 280, 285, 641, 927, 953–954
 sticky options, IPv6 (IPv6粘附的选项), 731–732
 Stone, J., 36
 str_cli function (str_cli函数), 125–126, 128, 136, 141–142, 147–148, 167, 169, 171–173, 175, 189, 401, 403, 406, 416, 437, 441–443, 446–447, 463, 679–681, 717
 str_echo function (str_echo函数), 122–123, 126, 128, 147, 149, 263, 400–402, 416, 430, 638–639, 681, 707

- strbuf structure (strbuf结构), 856, 866
 definition of (strbuf结构定义), 856
- strcat function (strcat函数), 15
- strcpy function (strcpy函数), 15
- Stream Control Transmission Protocol (流控制传输协议),
 见 SCTP
- stream (流)
 fully buffered standard I/O (完全缓冲的标准I/O流), 401
 line buffered standard I/O (行缓冲的标准I/O流), 402
 pipe, definition of (流管道定义), 415
 sequence number (流序号), 见 SSN
 socket (字节流套接字), 33
 standard I/O (标准I/O流), 399
 unbuffered standard I/O (不缓冲的标准I/O流), 402
- STREAMS (流), 851–868
 driver (流驱动程序), 851
 head (流头), 852
 ioctl function (流ioctl函数), 857–858
 message, high-priority (高优先级流消息), 183, 854
 message, normal (普通流消息), 183, 854
 message, priority band (优先级带流消息), 183, 854
 message types (流消息类型), 854–855
 modules (流模块), 852
 multiplexor (流多路复用器), 852–853
 queue (流队列), 854
- strerror function (strerror函数), 774–775, 910
- strict source and record route (严格的源与记录路由), 见
 SSRR
- <string.h> header (<string.h>头文件), 80
- strlcat function (strlcat函数), 15
- strlcpy function (strlcpy函数), 15
- strlen function (strlen函数), 916
- strncat function (strncat函数), 15
- strncpy function (strncpy函数), 15, 413
- strong end system model (强端系统模型), 103, 533
 definition of (强端系统模型定义), 247
- strtok function (strtok函数), 685
- strtok_r function (strtok_r函数), 685
- subnet (子网)
 address (子网地址), 875–876, 951
 ID (子网ID), 878
 mask (子网掩码), 875
- sum.h header (sum.h头文件), 148
- Sun RPC, 9, 62
- SUN_LEN macro (SUN_LEN宏), 412–413, 902
- sun_family member (sun_family成员), 412, 414
- sun_path member (sun_path成员), 412–414, 416
- SunOS 4, 131, 788, 793
- SunOS 5, 22
- superuser (超级用户), 111, 120, 213, 330, 363, 480,
 482–483, 486, 492, 496, 498, 511, 579, 617, 736, 746,
 759, 799, 862, 938
- SVR3 (System V Release 3), 161, 182–183, 851
- SVR4 (System V Release 4), 20, 34, 133, 140, 161–162, 164,
 182–183, 262, 336, 415, 420, 463, 545, 594, 663–664,
 772, 779, 787, 790, 815, 830, 832, 834, 836, 852, 855,
 857, 868, 891–892
- SYN (synchronize sequence numbers flag, TCP
 header), 37–38, 44, 57, 99–100, 102, 104–105, 107, 208,
 213, 219, 354–355, 357, 362, 416, 436, 710, 717–718,
 789, 896, 917, 921
 flooding (SYN泛滥), 108, 948
- SYN_RCVD state (SYN_RCVD状态), 41, 104, 106
- SYN_SENT state (SYN_SENT状态), 40–41, 101
- synchronize sequence numbers flag, TCP header (TCP首部
 同步序列号标志), 见 SYN
- synchronous, I/O (同步I/O), 160
- sysconf function (sysconf函数), 186, 189
- sysctl function (sysctl函数), 77, 482, 484–486,
 495–500, 502, 509
 definition of (sysctl函数定义), 496
- sysctl operations, routing socket (路由套接字sysctl
 操作), 495–499
- <sys/errno.h> header (<sys/errno.h>头文件),
 13, 436, 677, 913
- <sys/event.h> header (<sys/event.h>头文件),
 405
- <sys/ioctl.h> header (<sys/ioctl.h>头文件),
 466
- syslog function (syslog函数), 312, 340, 364–367, 369,
 379–380, 718, 910, 934
 definition of (syslog函数定义), 365
- syslogd program (syslogd程序), 363–367, 370, 379
- <sys/param.h> header (<sys/param.h>头文件),
 590
- <sys/select.h> header (<sys/select.h>头文件),
 163, 189
- <sys/signalf.h> header (<sys/signalf.h>头文件),
 664
- <sys/socket.h> header (<sys/socket.h>头文件),
 69–70, 98, 202, 238, 396–397, 429, 497
- <sys/stropts.h> header (<sys/stropts.h>头文
 件), 184
- <sys/sysctl.h> header (<sys/sysctl.h>头文件),
 497
- system call (系统调用)
 interrupted (被中断的系统调用), 131, 134–135, 139
 slow (慢系统调用), 134
 tracing (系统调用跟踪), 891–893
 versus function (对比系统调用与函数), 891
- System V Release 3 (System V版本3), 见 SVR3
- System V Release 4 (System V版本4), 见 SVR4
- Systems Network Architecture (系统网络体系结构), 见
 SNA
- <sys/tihdr.h> header (<sys/tihdr.h>头文件),
 858, 860
- <sys/types.h> header (<sys/types.h>头文件),
 69, 166, 189
- <sys/uio.h> header (<sys/uio.h>头文件), 389–390
- <sys/un.h> header (<sys/un.h>头文件), 412
- T_BIND_ACK constant (T_BIND_ACK常值), 860, 862

- T_bind_ack structure, definition of (T_bind_ack结构定义),** 862
- T_BIND_REQ constant (T_BIND_REQ常值),** 860
- T_bind_req structure (T_bind_req结构),** 860
- definition of (T_bind_req结构定义),** 860
- T_CONN_CON constant (T_CONN_CON常值),** 865
- T_conn_con structure, definition of (T_conn_con结构定义),** 865
- T_conn_req structure (T_conn_req结构),** 863
- definition of (T_conn_req结构定义),** 863
- T_DATA_IND constant (T_DATA_IND常值),** 866
- T_data_ind structure (T_data_ind结构),** 866
- definition of (T_data_ind结构定义),** 867
- T_DISCON_IND constant (T_DISCON_IND常值),** 865, 867
- T_discon_ind structure, definition of (T_discon_ind结构定义),** 865
- T_ERROR_ACK constant (T_ERROR_ACK常值),** 860, 862, 864
- T_error_ack structure, definition of (T_error_ack结构定义),** 862
- T_OK_ACK constant (T_OK_ACK常值),** 864–865
- T_ok_ack structure, definition of (T_ok_ack结构定义),** 865
- T_ORDREL_IND constant (T_ORDREL_IND常值),** 867
- T_ordrel_ind structure, definition of (T_ordrel_ind结构定义),** 867
- T_ordrel_req structure (T_ordrel_req结构),** 867
- definition of (T_ordrel_req结构定义),** 867
- T_primitives structure (T_primitives结构),** 865
- t_info structure (t_info结构),** 29
- t_ophdr structure (t_ophdr结构),** 29
- t_scalar_t datatype (t_scalar_t数据类型),** 29
- t_uscalar_t datatype (t_uscalar_t数据类型),** 29
- TACCES error (TACCES错误),** 862
- TADDRBUSY error (TADDRBUSY错误),** 862
- Tanenbaum, A. S., 8, 94
- tar program (tar程序),** 26
- Taylor, T., 36, 280, 954
- tcflush function (tcflush函数),** 465
- tcgetattr function (tcgetattr函数),** 465
- TCP (Transmission Control Protocol),** 33, 35–36
- and SIGIO signal (TCP与SIGIO信号), 664–665
- checksum (TCP校验和), 753
- client alternatives, (TCP客户程序设计范式) 819–820
- concurrent server, one child per client (每个客户一个子进程的TCP并发服务器), 822–825
- concurrent server, one thread per client (每个客户一个线程的TCP并发服务器), 842–843
- connection establishment (TCP连接建立), 37–43
- connection termination (TCP连接终止), 37–43
- MSS option (TCP MSS选项), 38
- options (TCP选项), 38–39
- out-of-band data (TCP带外数据), 645–653, 661–662
- output (TCP输出), 58–59
- preforked server (TCP预先派生子进程的服务器), 826–842
- preforked server, distribution of connections to children (TCP预先派生子进程的服务器中, 连接在子进程中的分布), 830–831, 835
- preforked server, select function collisions (TCP预先派生子进程的服务器中的select函数冲突), 831–832
- preforked server, too many children (TCP预先派生子进程的服务器中过多子进程的影响), 830, 834
- prethreaded server (TCP预先创建线程的服务器), 844–849
- SCTP versus (SCTP与TCP对比), 641–642
- segment (TCP分节), 35
- socket (TCP套接字), 95–120
- socket, connected (已连接TCP套接字), 109
- socket option (TCP套接字选项), 219–221
- state transition diagram (TCP状态转换图), 40–41
- three-way handshake (TCP三路握手), 37–38
- timestamp option (TCP数据戳选项), 39, 219, 950
- UDP, and SCTP, introduction (TCP、UDP与SCTP简介), 31–64
- urgent mode (TCP紧急模式), 645
- urgent offset (TCP紧急偏移), 646
- urgent pointer (TCP紧急指针), 646
- versus UDP (TCP与UDP对比), 594–597
- watching the packets (TCP观察分组), 42–43
- window (TCP窗口), 35
- window scale option (TCP窗口规模选项), 38, 208, 950
- TCP_MAXSEG constant (TCP_MAXSEG常值), 229
- TCP_MAXSEG socket option (TCP_MAXSEG套接字选项), 38, 194, 198, 219, 229, 236, 269, 895
- TCP_NODELAY socket option (TCP_NODELAY套接字选项), 194, 198, 219–221, 236–237, 269, 390, 895, 923
- tcp_close function (tcp_close函数), 140
- tcp_connect function (tcp_connect函数), 10, 319, 326–330, 337, 456, 696, 717
- definition of (tcp_connect函数定义), 326
- source code (tcp_connect函数源代码), 327
- tcp_listen function (tcp_listen函数), 330–335, 338–339, 378, 681, 823
- definition of (tcp_listen函数定义), 330
- source code (tcp_listen函数源代码), 331
- tcpdump program (tcpdump程序), 32, 101, 142, 144, 189, 248, 256–257, 265, 443, 547, 566, 585, 661, 711, 718, 787, 789, 793, 800, 815, 893, 896–897, 921, 925–926
- TCP/IP big picture (TCP/IP总图),** 32–34
- TCP/IP Illustrated, Volume 1, 见TCPv1**
- Volume 2, 见TCPv2
- Volume 3, 见TCPv3
- TCPv1 (TCP/IP Illustrated, Volume 1),** 953
- TCPv2 (TCP/IP Illustrated, Volume 2),** 954
- TCPv3 (TCP/IP Illustrated, Volume 3),** 953
- Telnet (remote terminal protocol),** 61–62, 151, 219–220, 662, 916
- telnet program (telnet程序),** 93, 350

- termcap file (termcap文件), 169
 termination of server process (服务器进程终止), 141–142
 test networks and hosts (测试网络与主机), 22–25
 test programs (测试程序), 896
 test_udp function (test_udp函数), 799, 801
 TFTP (Trivial File Transfer Protocol), 57, 62, 213, 253, 587, 596–597, 613–614
 Thaler, D., 551, 949
 Thomas, M., 28, 216, 397, 719, 732, 738, 744, 936, 953
 Thomas, S., 551, 949
 Thomson, S., 28, 71, 216, 304, 346–347, 361, 504, 949, 954
 thr_join function (thr_join函数), 695–697, 701, 705
 Thread structure (Thread结构), 844, 846
 thread_main function (thread_main函数), 845, 847
 thread_make function (thread_make函数), 845, 847
 <thread.h> header (<thread.h>头文件), 694
 threads (线程), 675–707
 - argument passing (线程参数传递), 682–685
 - attributes (线程属性), 677
 - detached (已脱离的线程), 678
 - ID (线程ID), 677
 - joinable (可汇合线程), 678
 thread-safe (线程安全的), 86, 92, 346, 685–686, 691, 843
 thread-specific data (线程特定数据), 92, 343, 346, 686–694
 three-way handshake (三路握手), 37, 99, 104–109, 198, 208, 252, 256, 383, 436, 448–449, 451, 649, 656, 717–719, 826, 938
 - TCP (TCP三路握手), 37–38
 thundering herd (惊群), 830, 834, 846
 Thyagarajan, A., 564, 948
 time (时间)
 - absolute (绝对时间), 704
 - delta (增量时间), 704
 - exceeded, ICMP (ICMP超时错误), 755, 761, 764, 771, 883–884
 - port (流逝时间获取服务端口), 61
 TIME_WAIT state (TIME_WAIT状态), 41, 43–44, 62, 128, 151, 203, 207, 236–237, 339, 820, 897, 915–916, 921
 time function (time函数), 14–15
 time program (time程序), 447
 time_t datatype (time_t数据类型), 182
 timeout (超时)
 - BPF, receive (BPF接收超时), 789
 - connect function (connect函数超时), 382–383
 - recvfrom function with a (带超时的recvfrom函数), 383–386
 - socket (套接字超时), 210, 381–386
 - UDP (UDP超时), 597
 timespec structure (timespec结构), 181–182, 405, 704–705, 903
 - definition of (timespec结构定义), 181
 timestamp option, TCP (TCP时间戳选项), 39, 219, 950
 timestamp request, ICMP (ICMP时间戳请求), 739, 883
 time-to-live (存活时间), 见 TTL
 timeval structure (timeval结构), 161–162, 181–182, 193, 210, 385–386, 405, 449, 606, 666, 704, 747, 941
 - definition of (timeval结构定义), 161
 timod STREAMS module (timod流模块), 853, 858
 tirdwr STREAMS module (tirdwr流模块), 853–854
 TLI_error member (TLI_error成员), 862
 TLV (type, length, value), 720
 tmpnam function (tmpnam函数), 419, 685
 token ring (令牌环), 34, 63, 199, 550–551, 914
 Torek, C., 213, 954
 TOS (type-of-service), 215, 870, 883, 948
 total length field, IPv4 (IPv4总长度字段), 870
 Touch, J., 294, 954
 TPI (Transport Provider Interface), 854, 858–868, 954
 tpi_bind function (tpi_bind函数), 859–860, 863
 tpi_close function (tpi_close函数), 860, 867
 tpi_connect function (tpi_connect函数), 860, 863
 tpi_daytime.h header (tpi_daytime.h头文件), 858
 tpi_read function (tpi_read函数), 866
 trace.h header (trace.h头文件), 755
 traceloop function (traceloop函数), 757, 759, 765
 traceroute program (traceroute程序), 33, 62, 214–215, 218, 617, 619
 - implementation (traceroute程序实现), 755–768
 traffic class (流通类别), 618, 871
 transaction time (事务处理时间), 595
 transient multicast group (临时多播组), 551
 Transmission Control Protocol (传输控制协议), 见 TCP
 transport sequence number (传输序号), 见 TSN
 Transport Layer Interface (传输层控制), 见 TLI
 Transport Provider Interface (传输提供者接口), 见 TPI
 Trivial File Transfer Protocol (简化文件传送协议), 见 TFTP
 trpt program (trpt程序), 199
 truncation, UDP, datagram (UDP数据报截断), 594
 truss program (truss程序), 892–893
 TRY AGAIN constant (TRY AGAIN常值), 308
 ts member (ts成员), 809
 TSN (transport sequence number), 224–225
 TTL (time-to-live), 43, 215, 217–218, 552–553, 559, 563, 566, 575, 749, 755, 757, 759, 761–762, 772, 871, 873, 883, 886
 ttynname function (ttynname函数), 685
 ttynname_r function (ttynname_r函数), 685
 Tuexen, M., 285, 953
 tunnel (隧道), 885–889
 - automatic (自动形成的隧道), 880
 tv_nsec member (tv_nsec成员), 181, 903
 tv_sec member (tv_sec成员), 161–162, 181, 903
 tv_sub function (tv_sub函数), 747
 - source code (tv_sub函数源代码), 747
 tv_usec member (tv_usec成员), 161, 181
 type field, ICMP (ICMP类型字段), 882
 type, length, value (类型、长度、值), 见 TLV
 type-of-service (服务类型), 见 TOS
 typo (排印或打字错误), 51
 typographical conventions (排版约定), 7

- u_char datatype** (*u_char*数据类型), 69, 559
u_int datatype (*u_int*数据类型), 69, 559
u_long datatype (*u_long*数据类型), 69
u_short datatype (*u_short*数据类型), 69
userdata member (*userdata*成员), 405
UDP (User Datagram Protocol), 33–34
 adding reliability to application (给UDP应用增加可靠性), 597–608
 and SCTP, introduction, TCP (TCP、UDP与SCTP简介), 31–64
 and SIGIO signal (UDP与SIGIO信号), 664
 binding interface address (UDP捆绑接口地址), 608–612
 checksum (UDP校验和), 259, 497–499, 753, 793–814
 concurrent server (UDP并发服务器), 612–614
 connect function (适合UDP的*connect*函数), 252–255
 datagram truncation (UDP数据报截断), 594
 determining outgoing interface (UDP确定外出接口), 261–262
 lack of flow control (UDP缺乏流控), 257–261
 lost datagrams (UDP丢失的数据报), 245–246
 output (UDP输出), 59–60
 sequence number (UDP序号), 597
 server not running (UDP服务器进程未运行), 248–249
 socket (UDP套接字), 239–265, 587–620
 socket, connected (已连接UDP套接字), 252
 socket receive buffer(UDP套接字接收缓冲区), 260–261
 socket, unconnected (未连接的UDP套接字), 252
 TCP versus (TCP与UDP对比), 594–597
 timeout (UDP超时), 597
 verifying received response (UDP验证接收到的响应), 246–248
udp_check function (*udp_check*函数), 808–809
udp_client function (*udp_client*函数), 334–337, 572, 577, 580–582, 620, 935, 941
 definition of (*udp_client*函数定义), 334
 source code (*udp_client*函数源代码), 335
udp_connect function (*udp_connect*函数), 337, 935
 definition of (*udp_connect*函数定义), 337
 source code (*udp_connect*函数源代码), 337
udp_read function (*udp_read*函数), 803, 806, 815
udp_server function (*udp_server*函数), 338–339, 933
 definition of (*udp_server*函数定义), 338
 source code (*udp_server*函数源代码), 338
udp_server_reuseaddr function (*udp_server_reuseaddr*函数), 933
udp_write function (*udp_write*函数), 805, 814
udpcksum.h header (*udpcksum.h*头文件), 795
udpiphdr structure (*udpiphdr*结构), 805
ui_len member (*ui_len*成员), 806
ui_sum member (*ui_sum*成员), 806
uint16_t datatype (*uint16_t*数据类型), 69
uint32_t datatype (*uint32_t*数据类型), 69, 75
uint8_t datatype (*uint8_t*数据类型), 68–69
umask function (*umask*函数), 414–415
uname function (*uname*函数), 577
unbuffered standard I/O stream (不缓冲的标准I/O流), 402
unconnected UDP socket (未连接的UDP套接字), 252
unicast (单播)
 broadcast versus (广播与单播对比), 532–535
 multicast versus (多播与单播对比), 553
 scope, global (全球单播范围), 878
 scope, link-local (链路局部单播范围), 881
 scope, site-local (网点局部单播范围), 881
uniform resource identifier (统一资源标识符), 见URI
uniform resource locator (统一资源定位符), 见URL
<unistd.h> header (*<unistd.h>*头文件), 466, 516
Unix 95, 27
Unix 98, 30, 133, 184, 346, 685, 919, 952
 definition of (Unix定义), 27
Unix domain (Unix域)
 differences in socket functions (Unix域在套接字函数上的差异), 415–416
 socket (Unix域套接字), 411–433
 socket address structure (Unix域套接字地址结构), 412–414
Unix International (Unix国际), 790, 854, 954
Unix I/O, definition of (Unix I/O定义), 399
/unix service (/unix服务), 936
Unix standard services (Unix标准服务), 52
Unix standards (Unix标准), 25–28
UNIX_error member (*UNIX_error*成员), 862
UNIXDG_PATH constant (*UNIXDG_PATH*常值), 419
 definition of (*UNIXDG_PATH*常值定义), 902
UNIXSTR_PATH constant (*UNIXSTR_PATH*常值), 416
 definition of (*UNIXSTR_PATH*常值定义), 902
Unix-to-Unix Copy (Unix到Unix复制), 见UUCP
UnixWare, 20, 257
unlink function (*unlink*函数), 413–414, 416, 419, 432, 777, 834, 935
unordered data, SCTP (SCTP无序的数据), 629
unp_in_pktinfo structure (*unp_in_pktinfo*结构), 588, 590, 901
 definition of (*unp_in_pktinfo*结构定义), 588
unp.h header (*unp.h*头文件), 7–9, 13, 71, 86, 122, 125, 131, 242, 416, 419, 491, 588, 592, 679, 795, 899–904
 source code (*unp.h*头文件源代码), 899
unpicmpd.h header, source code (*unpicmpd.h*头文件源代码), 771
unpifi.h header (*unpifi.h*头文件), 469
 source code (*unpifi.h*头文件源代码), 471
unproute.h header (*unproute.h*头文件), 491
unprtt.h header (*unprtt.h*头文件), 601, 604, 606
 source code (*unprtt.h*头文件源代码), 604
unpthread.h header (*unpthread.h*头文件), 679
unspecified address (未指定地址), 876, 881
URG (urgent pointer flag, TCP header), 646–647, 661
urgent (紧急的)
 mode, TCP (紧急模式), 645
 offset, TCP (TCP紧急偏移), 646
 pointer flag, TCP header (TCP首部紧急指针标志), 见URG
 pointer, TCP (TCP紧急指针), 646

- URI (uniform resource identifier), 575
 URL (uniform resource locator), 947
 User Datagram Protocol (用户数据报协议), 见 UDP
 user ID (用户ID), 350, 374, 429, 431, 676, 746, 759, 799
 UTC (Coordinated Universal Time), 15, 61, 575, 582, 606, 704
 UUCP (Unix-to-Unix Copy), 366
- value-result argument (值-结果参数), 74–77, 109–111, 164, 183, 192, 197, 246, 389, 391, 394, 414, 469, 496, 499, 590, 710, 717, 856–857, 915, 932
 Varadhan, K., 874, 949
`/var/adm/messages` file (`/var/adm/messages`文件), 379
`/var/log/messages` file (`/var/log/messages`文件), 370
`/var/run/log` file (`/var/run/log`文件), 364, 367
 verifying received response, UDP (UDP验证接收到的响应), 246–248
 version number field, IP (IP版本号字段), 869, 871
 vi program (vi程序), 26
 virtual network (虚拟网络), 885–889
 virtual private network (虚拟专用网络), 见 VPN
 Vixie, P. A., 308, 954
`void datatype` (`void`数据类型), 9, 70–71, 88, 131, 677, 679, 681, 915
 volatile qualifier (volatile限定词), 802
 VPN (virtual private network), 22
- `wait` function (`wait`函数), 132–133, 135–139, 151, 613, 820, 827
 definition of (`wait`函数定义), 135
`waitpid` function (`waitpid`函数), 132–133, 135–139, 151, 376, 423, 677–678
 definition of (`waitpid`函数定义), 135
`wakeup_one` function (`wakeup_one`函数), 830
 WAN (wide area network), 5, 35, 219, 448, 549, 556–558, 596–597
 wandering duplicate (漫游的重复分组), 43
 weak end system model (弱端系统模型), 103, 533, 608, 666, 916
 definition of (弱端系统模型定义), 247
`web_child` function (`web_child`函数), 825, 829, 843, 847
`web_client` function (`web_client`函数), 842
`web.h` header (`web.h`头文件), 454
 Webstone benchmark (Webstone基准测试程序), 820
 well-known (众所周知的)
 address (众所周知的地址), 52
 multicast group (众所周知的多播组), 551, 571
 port (众所周知的端口), 50
 WEXITSTATUS constant (WEXITSTATUS常值), 135, 423
 Whelan, E., 571, 949
 wide area network (广域网), 见 WAN
`WIFEXITED` constant (`WIFEXITED`常值), 135
 wildcard address (通配地址), 53, 87, 102, 122, 126, 147, 211, 322, 354–355, 357, 362, 373, 560, 562, 568, 581–582, 608, 610–611, 772, 779, 876, 881
 window scale option, TCP (TCP窗口规模选项), 38, 208, 950
 window, TCP (TCP窗口), 35
 Wise, S., 315
 WNOHANG constant (WNOHANG常值), 136, 138
 World Wide Web (万维网), 见 WWW
 wrapper function (包裹函数), 11–13
- source code, Listen (`Listen`包裹函数源代码), 107
 source code, Pthread_mutex_lock (`Pthread_mutex_lock`包裹函数源代码), 12
 source code, Socket (`Socket`包裹函数源代码), 11
 Wright, G. R., 954
`writable_timeo` function (`writable_timeo`函数), 385
`write` function (`write`函数), 15, 29–30, 58, 60, 88, 117, 135, 143, 152, 174, 200, 210, 221, 237, 240, 252–253, 255–256, 269, 271, 337, 344, 381–382, 387, 389–390, 395, 399, 403, 408, 432, 435, 437, 440, 442, 445, 458, 492, 495, 509, 648, 660, 665, 736–737, 790, 841, 854–856, 914, 916, 919, 923, 935, 938, 945
`write_fd` function (`write_fd`函数), 427–428, 773, 841
 source code (`write_fd`函数源代码), 428
`write_get_cmd` function (`write_get_cmd`函数), 457–459, 697
`written` function (`written`函数), 88–93, 121, 123, 125–126, 141–144, 149–151, 168–169, 175, 288, 400, 437, 458
 definition of (`written`函数), 88
 source code (`written`函数源代码), 89
`writev` function (`writev`函数), 210, 221, 237, 381, 389–391, 395, 408, 435, 601, 737, 924
 definition of (`writev`函数定义), 389
 WWW (World Wide Web), 3, 106, 310, 448, 452–461, 818, 820, 822, 834
- XDR (external data representation), 150
 Xerox Network Systems (Xerox网络系统), 见 XNS
 Xie, Q., 36, 46, 49–50, 61, 203, 227, 280, 285, 641, 927, 953–954
`xinetd` program (`xinetd`程序), 377
 XNS (Xerox Network Systems), 28, 98
 XNS (X/Open Networking Services), 27, 952
 X/Open, 27
 Networking Services (X/Open网络服务), 见 XNS
 Portability Guide (X/Open移植性指南), 见 XPG
 Transport Interface (X/Open传输接口), 见 XTI
 XPG (X/Open Portability Guide), 27
 XTI (X/Open Transport Interface), 27, 29
`yacc` program (`yacc`程序), 26
 Yoakum, J., 36, 952
 Yu, J. Y., 874, 949
- Zhang, L., 36, 44, 280, 950, 954
 zombie (僵死), 129, 132–134, 137, 139

函数和宏定义索引表

(下面所列页码均指页边栏中标注的页码, 加粗的页码指示源代码实现所在之处)

函数或宏	页码	函数或宏	页码
accept	109	getservbyname	311
bcmp	81	getservbyport	312
bcopy	81	getsockname	118
bind	101	getsockopt	192
bzero	81	gf_time	442
		host_serv	325, 326
close	117	htonl	79
closelog	367	htons	79
CMSG_xxx	397		
connect	99	ICMP6_FILTER_xxx	740
connect_nonb	450	if_freenameindex	504, 508
connect_timeo	382	if_indextoname	504, 506
		if_nameindex	504, 507
daemon_inetd	377	if_nametoindex	504, 505
daemon_init	368	IN6_IS_ADDR_xxx	360
dg_send_recv	602	in_cksum	753
		inet6_opt_xxx	723
err_doit	910	inet6_rth_xxx	727
err_dump	910	inet_addr	82
err_msg	910	inet_aton	82
err_quit	910	inet_ntoa	82
err_ret	910	inet_ntop	83
err_sys	910	inet_pton	83
execxxx	113	ioctl	466, 857
fcntl	235	kevent	405
fork	111	kqueue	405
freeaddrinfo	321		
free_ifi_info	480	listen	104
gai_strerror	321	mcast_block_source	565
getaddrinfo	315	mcast_get_if	565
gethostbyaddr	310	mcast_get_loop	565
gethostbyaddr_r	345	mcast_get_ttl	565
gethostbyname	307	mcast_join	565, 567
gethostbyname2	347	mcast_join_source_group	565
gethostbyname_r	345	mcast_leave	565
get_ifi_ifno	474, 501	mcast_leave_source_group	565
getipnodebyname	347	mcast_set_if	565
getmsg	856	mcast_set_loop	565, 570
getnameinfo	340	mcast_set_ttl	565
getpeername	118	mcast_unblock_source	565
getpmsg	857	memcmp	81

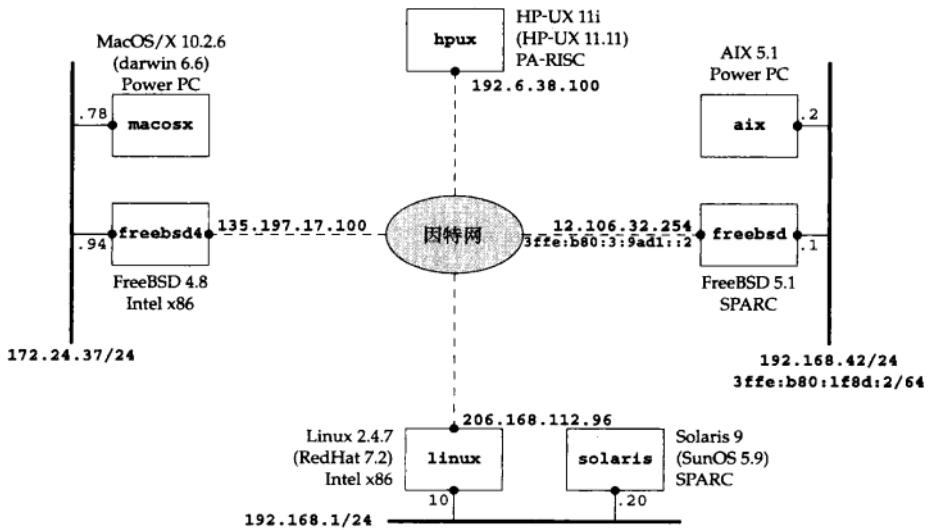
(续)

函数或宏	页码	函数或宏	页码
Memcpy	81	sctp_bindx	272
memset	81	sctp_connectx	274
		sctp_freeladdrs	276
ntohl	79	sctp_freepaddrs	275
ntohs	79	sctp_getladdrs	275
		sctp_getpaddrs	275
openlog	367	sctp_sendmsg	276
		select	161
poll	182	send	387
pselect	181, 543	sendmsg	390
pthread_cond_broadcast	704	sendto	240
pthread_cond_signal	702	setsockopt	192
pthread_cond_timedwait	704	shutdown	173
pthread_cond_wait	702	signal	130
pthread_create	677	socketmark	654, 654
pthread_detach	678	sock_bind_wild	87
pthread_exit	678	sock_cmp_addr	87
pthread_getspecific	691	sock_cmp_port	87
pthread_join	677	socket	96
pthread_key_create	690	socketpair	414
pthread_mutex_lock	700	sockfd_to_family	119
pthread_mutex_unlock	700	sock_get_port	87
pthread_once	690	sock_ntop	86, 87
pthread_self	678	sock_ntop_host	87
pthread_setspecific	691	sock_set_addr	87
putmsg	856	sock_set_port	87
putpmsg	857	sock_set_wild	87
		sysctl	496
readable_timeo	385	syslog	365
read_fd	426		
readline	88, 90, 91, 693	tcp_connect	326, 327
readn	88, 89	tcp_listen	330, 331
readv	389	tv_sub	747
recv	387		
recvfrom	240	udp_client	334, 335
recvmsg	390	upd_connect	337, 337
rtt_init	605	udp_server	338, 338
rtt_minmax	605		
rtt_newpack	606	wait	135
rtt_start	606	waitpid	135
rtt_stop	607	write_fd	428
rtt_timeout	607	written	88, 89
rtt_ts	606	writev	389

结构定义索引表

(下面所列页码均指页边栏中标注的页码)

结构	页码	结构	页码
addrinfo	315	sadb_address	519
arpreq	481	sadb_alg	524
		sadb_key	519
cmsgcred	429	sadb_lifetime	523
cmsghdr	396	sadb_msg	513
		sadb_sa	518
group_req	560	sadb_supported	524
group_source_req	562	sctp_adaption_event	285
		sctp_assoc_change	281
hostent	307	sctp_assocparams	222
		sctp_event_subscribe	226
ifa_msghdr	488	sctp_initmsg	228
if_announce_msghdr	488	sctp_notification	281
ifconf	469	sctp_paddr_change	283
ifma_msghdr	488	sctp_paddrinfo	226
if_msghdr	488	sctp_paddrparams	229
if_nameindex	504	sctp_pdapi_event	286
ifreq	469	sctp_remote_error	283
in6_addr	71	sctp_rtinfo	231
in6_pktnfo	616	sctp_send_failed	284
in_addr	68	sctp_setpeerprim	231
iovec	389	sctp_setprim	230
ip6_mtuinfo	619	sctp_shutdown_event	285
ip_mreq	560	sctp_sndrcvinfo	224
ip_mreq_source	562	sctp_status	232
ipoption	714	sctp_tlv	280
ipv6_mreq	560	servent	311
		sockaddr	70
kevent	405	sockaddr_dl	486
		sockaddr_in	68
linger	202	sockaddr_in6	71
		sockaddr_storage	72
msghdr	390	sockaddr_un	412
		strbuf	856
pcap_pkthdr	809		
pollfd	183	timespec	181
		timeval	161
rt_msghdr	488	unp_in_pktnfo	588



用于正文中大多数示例的主机和网络





手机号码: _____ (此为会员编号)

姓 名: _____ 性别: 男 女 出生年月: ____ 年 ____ 月

通信地址: _____

邮政编码: _____

电子邮件: _____

您购买的图书是: 22840 / 《UNIX网络编程 卷1：套接字联网API（第3版）》 (129.00元)

您获得的会员积分是: 12.9 分

欢迎参加“**有奖DEBUG**”活动。提交本书勘误，每确认一处即可获赠积分5分。详情见图灵网站。

请沿虚线剪下此页，寄回图灵公司，即可成为图灵读者俱乐部的一员（复印无效）。**积分累计，可获赠书**（赠书清单见图灵网站）。

邮政编码: 100107

通信地址: 北京市朝阳区北苑路 13 号院 1 号楼 C603

北京图灵文化发展有限公司 图灵读者俱乐部



[General Information]

书名=UNIX网络编程 卷1 套接字联网API

作者=(美)W.Richard Stevens著

页数=809

出版社=人民邮电出版社

出版日期=2010.06

SS号=12586890

DX号=000006907086

URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=000006907086&d=055D0A2638ECBDEFF3A46D127866DC5D