套接字联网API

大佬说这本书只用看前8章，相信大佬好了，多了其实也看不完。。

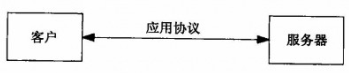
看了《UNIX高级环境编程》之后，知道了套接字其实也是一种IPC的方式，而且套接字可以与文件描述符关联。也可以实现通过文件描述符来输入和输出内容，因此，我对这本书《套接字编程》的理解是详细介绍套接字编程，讲解IPC中套接字的具体实现和使用。所以看来先看《高级环境编程》是正确的。

第一章：简介

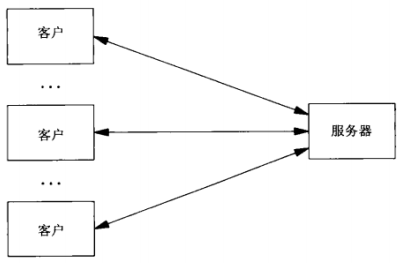
1、要编写通过计算机网络通信的程序，首先要确定这些程序相互通信所使用的的协议

2、一般认为Web服务器程序是一个长时间运行的程序（守护程序，记忆中好像只用设置一些方式就可以了），它只在响应来自网络的请求时才发送网络消息。协议的另一端是Web客户程序，如某种浏览器，与服务器进程的通信总是由客户进程发起的。

3、一般按照客户端和服务器来组织

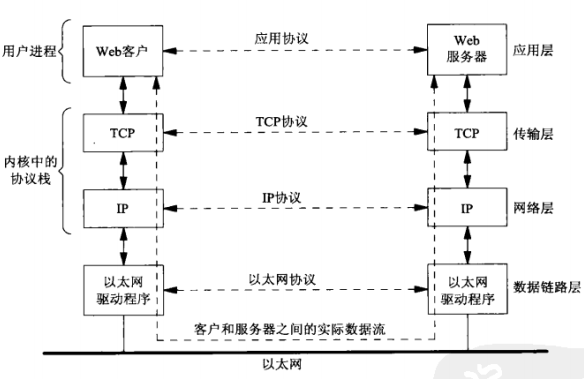


通常客户每次只与一个服务器通信，但一个服务器会与多个客户进行通信

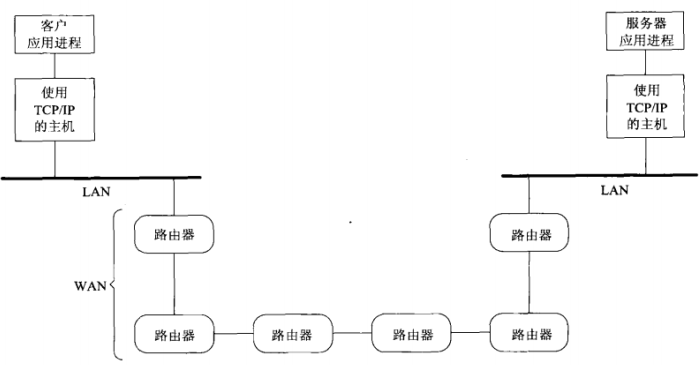


客户机和服务器使用某个网络协议通信的，本书的焦点是TCP/IP。实际上会一层一层的进行通讯

一层一层的逐渐下降



同一网络应用的客户和服务器无需处于同一个局域网。如图所示



4、接下来介绍了一个获取服务器时间的客户程序：

包含头文件

命令行参数

创建TCP套接字

指定服务器的IP地址和端口

建立与服务器的连接

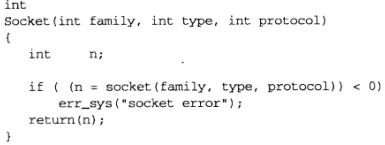
读入并输出服务器的应答

终止程序

协议无关性：程序兼容不同协议，例如上面的程序可以使用IPv4或IPv6

5、错误处理：程序必须检查每个函数调用是否返回错误。发生错误时就用我们的err\_quit或err\_sys函数输出一个出错消息并终止程序的运行。既然发生错误时终止程序的运行是非常普遍的情况，我们可以定义包裹函数

如下，Socket就是socket的包裹函数



本书中首字母大写的函数名，就是定义的包裹函数，调用的实际函数的名字与包裹函数名相同，只是以对应的小写字母开头。

本书的例子中，除非必须检查某个确定的错误是否发生，并以不同于终止进程的其他某种方式处理它，否则就是用这些包裹函数。（看UNIX高级环境编程时，发现对错误的处理确实会让代码很乱，包裹函数确实让代码简洁了很多。）

7、UNIX errno值：只要一个Unix函数中有错误发生，全局变量errno就被设置为一个指明该错误类型的正值，函数本身则返回-1，全书中使用例如“connect函数返回ECONNREFUSED”这样的句子简明表达以下意思：该函数返回一个错误，同时errno被置为指定的常值。

8、服务器程序：（socket bind listen这三步是任何TCP服务器准备所谓的监听描述符的正常步骤）

创建TCP套接字（socket）

把服务器的众所周知端口捆绑到套接字（时间获取服务是13）（bind）

把套接字转换成监听套接字（listen）

接受客户连接，发送应答（accept，服务器在这一步等待，TCP采用三次握手来建立连接，握手完毕返回的是一个已连接描述符的新描述符）

终止连接（调用close关闭与客户的连接，引发正常的TCP连接终止序列。 三次握手，四次挥手。）

（与版本无关的程序在11章，这是一个迭代式的服务器，一次只能响应一个客户端的请求，并发式的最简单的是使用Unix的fork函数。为每个客户创建一个子进程，其他技术包括线程代替fork，或在服务器启动时预先fork一定数量的子进程

9、如果在shell命令行启动本例这样的一个服务器，我们也许想要它运行很久，这要求我们在服务器中添加代码，让它作为一个Unix守护进程（daemon）

10、OSI网络七层模型需要知道：

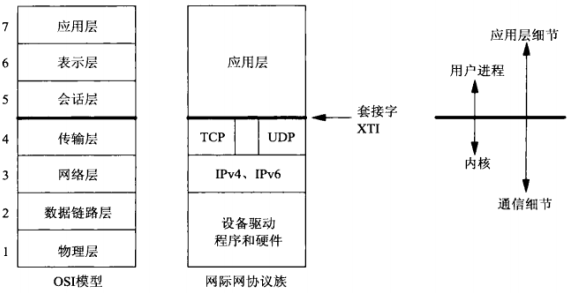
底下两层是随系统提供的设备驱动程序和网络硬件。通常除了需要知道数据链路的某些特性外（如MTU为1500字节），不必关心这两层的具体情况

网络层由IPv4和IPv6这两个协议处理，传输层有UDP和TCP，网络应用绕过传输层直接使用IPv4或IPv6是可能的。也就是所谓的原始套接字

OSI顶上3层被合并为一层，称为应用层。

11、本书的焦点是如何使用套接字编写使用TCP或UDP的网络应用程序。

为什么套接字提供从OSI模型的顶上三层进入传输层的接口？：顶上三层处理具体网络应用的所有细节，对通信细节了解很少；底下四层对具体网络应用了解不多，却处理所有的通信细节；其次是顶上三层构成所谓用户进程，底下四层却通常作为操作系统内核的一部分提供。UNIX与其他现代操作系统都提供分隔用户进程与内核的机制，由此可见，第四层和第五层之间的接口是构建API的自然位置



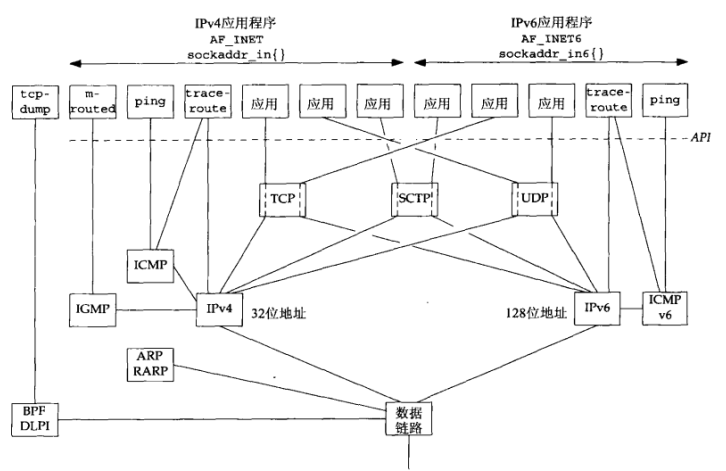
12、网络拓扑：大多数UNIX系统都提供了可用于发现某些网络细节的两个基本命令，netstat和ifconfig。Netstat -i提供网络接口的信息，-n标志指定以输出数值地址，netstat -r展示路由表，也是另一种确定接口的方法。有了各个网络接口的名字，执行ifconfig就可以获得每个接口的详细信息。对本地接口的广播地址执行ping就可以获得本地网络中众多主机的IP地址

13、size\_t在32位系统中是32位，但在64位中就会变为64位，同样还有long ，有滴地方不需要变化，因此在不需要变化的地方设计特有的数据类型

第二章：传输层 TCP UDP SCTP

本章介绍传输层的协议的概貌。从网络编程角度提供足够的细节理解如何使用这些协议

1、总图：右边的网络应用使用IPv6，随sockaddr\_in6结构讲解AF\_INET6常值，左边的使用IPv4，使用sockaddr\_in结构，值为AF\_INET



（应该记住各种协议的名称）

2、用户数据报协议（UDP）：应用程序向UDP套接字中写入一个消息，该消息随后被封装到一个UDP数据报，该UDP数据报随后又被封装到一个IP数据报，然后发送到目的地。UDP不保证UDP数据报回到达其最终目的地，不保证每个数据报的先后顺序跨网络保持不变，也不保证每个数据报只到达一次。UDP缺乏可靠性，如果一个数据报到达了最终目的地，但是校验和检测有错误，或者数据报被中途丢弃了，它就无法被投递给UDP套接字，也不会被源端自动重传。如果想要确保一个数据报到达目的地，可以往应用程序中添加一大堆的特性：来自对端的确认，本端的超时和重传等。每个UDP数据报都有一个长度，如果数据报正确的到达目的地，该数据报的长度将随数据一道传递给接收端应用进程，而TCP是一个字节流协议，没有任何记录边界。UDP提供无连接的服务，因为UDP客户和服务器之间不必存在任何长期的关系，例如一个UDP客户可以创建一个套接字发送一个数据报给一个给定的服务器，然后立即用同一个套接字发送给另一个数据报给另一个服务器，同样地，一个UDP服务器可以用同一个UDP套接字从若干个不同的客户接受数据报，每个客户一个数据报。

3、传输控制协议（TCP）：

TCP提供客户与服务器之间的连接，TCP客户先与某个给定服务器建立一个连接，再跨该连接与那个服务器交换数据，然后终止这个连接。

TCP提供了可靠性，当TCP向另一端发送数据时，它要求对端返回一个确认，如果没有收到确认，TCP就自动重传数据并等待更长时间。在数次重传失败后，TCP才放弃。重传一般花费4~10分钟

TCP有动态估算客户和服务器之间的往返时间RTT的算法，以便它知道等待一个确认要多少时间。

TCP通过每个字节关联一个序列号对所发送的数据进行排序。如果TCP接收到来自对端的重复数据，它可以判定数据是重复的，从而丢弃重复数据

TCP提供流量控制，TCP总是告知对端在任何时刻它一次能够从对端接受多少字节的数据，这称为通告窗口，窗口指出接受缓冲区中当前可用的空间量，从而确保发送端发送的数据不会使接受缓冲区溢出。该窗口动态变化，当接受到来自发送端的数据时，窗口大小就减小。

TCP是全双工的，在一个给定的连接上应用可以在任何时刻在进出两个方向上既发送数据又接收数据。

4、流控制传输协议（SCTP ）

SCTP提供类似UDP和TCP的服务

SCTP在客户端和服务器之间提供关联，并像TCP那样给应用提供可靠性、排序、流量控制以及全双工的数据传送。一个关联指代两个系统之间的一次通信，它可能因为SCTP支持多宿而涉及不止两个地址。

SCTP面向消息，提供各个记录的按序递送服务。和UDP一样，由发送端写入的每条记录的长度随数据一道传递给接收端应用。

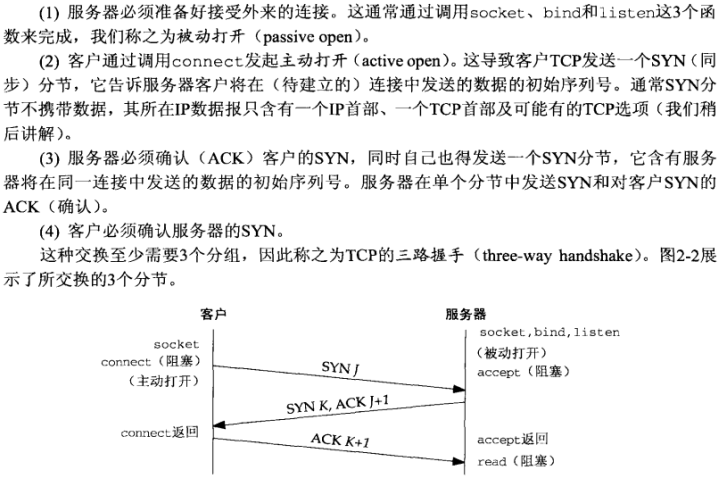
SCTP能够在所连接的端点之间提供多个流，每个流各自可靠地按序递送消息。一个流上某个消息的丢失不会阻塞同一关联其他流上消息的投递。与TCP正好相反

SCTP还提供多宿特性，使得单个SCTP端点能够支持多个IP地址。该特性可以增强应对网络故障的健壮性。一个端点可能有多个冗余的网络连接，每个网络又可能有各自介入因特网基础设施的连接。SCTP可以通过切换到使用已与该关联相关的另一个地址来规避所发生的的故障。

5、TCP的建立和终止

这有助于理解connect、accept、close这三个函数并使用netstat程序调试TCP应用

三次握手：建立一个TCP连接时发生下述情形。



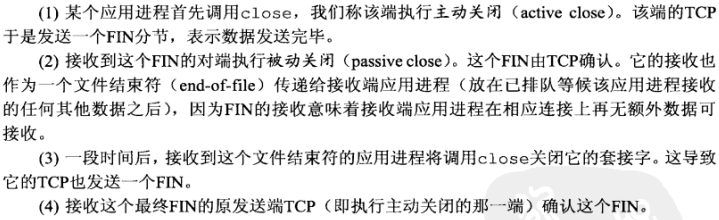
SYN报文可以含有多个TCP选项，

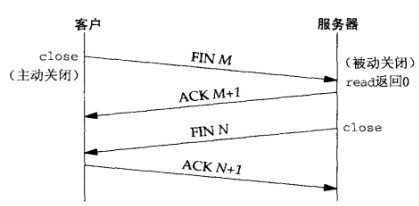
如MSS选项：最大分节大小，MSS，也就是它在本连接的每个TCP分节中愿意接受的最大数据量。

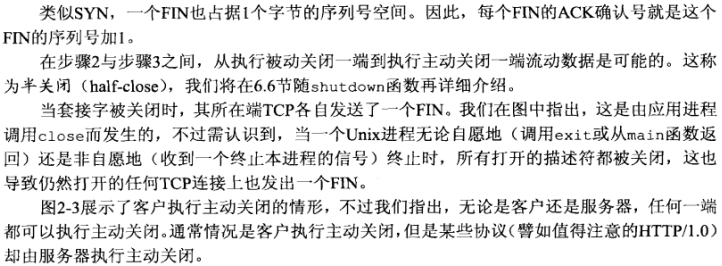
窗口规模选项：TCP连接任何一端能够通告对端的最大窗口大小是65535，因为在TCP首部中相应的字段占16位。新选项指定TCP首部中的通告窗口必须扩大的位数是0~14，因此所提供的最大窗口65535\*214 约为1GB

时间戳选项：防止失而复现的网络分组造成数据损坏。网络编程人员可以不考虑

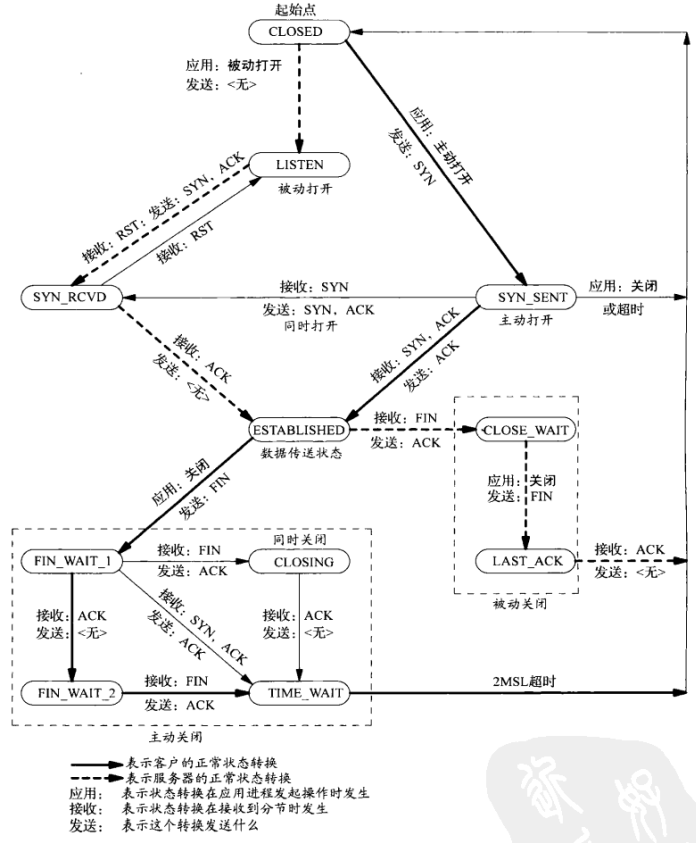
TCP终止连接的4次挥手：







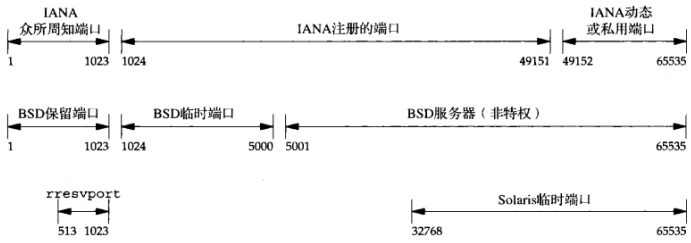
6、TCP状态转换图：TCP涉及连接建立和连接终止的操作可以用状态转换图来说明，提供有11种状态。这些状态可以使用netstat显示。



7、TIME\_WAIT状态存在的理由：可靠地实现TCP全双工连接的终止；允许老的重复分节在网络中消逝。2MSL

8、任何时候，多个进程使用TCP UDP SCTP传输层协议，3个协议都使用16位整数的端口号来区分这些进程。

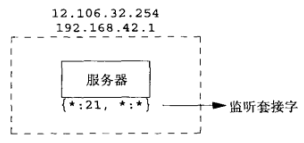
众所周知的端口为0~1023。已登记的端口为1024-49151，49151~65535是动态的或私用的端口。



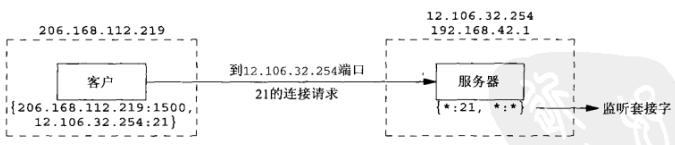
9、套接字对：一个TCP连接的套接字对是一个定义该连接的两个端点的四元组：本地IP地址、本地TCP端口号、外地IP地址、外地TCP端口号。

10、TCP端口号与并发服务器：过程图解

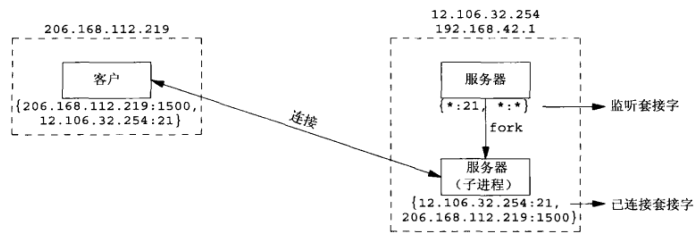
服务器本地任意接口的端口21上等待监听，然后监听套接字对为｛\*：21，\*：\*｝



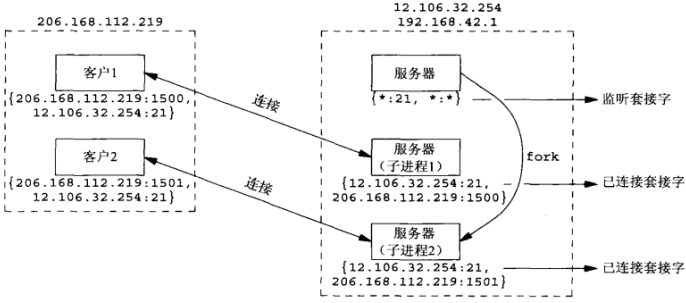
客户端主动打开，



客户和服务器进行连接，服务器将fork一个自身的副本，让子进程来处理该客户的请求。



假设另一个客户请求连接到同一个服务器，则产生第二个套接字对



TCP无法仅仅通过查看目的端口号来分离外来的分节到不同的端点。它必须查看套接字对的所有4个元素才能确定哪个端点接受某个到达的分节。

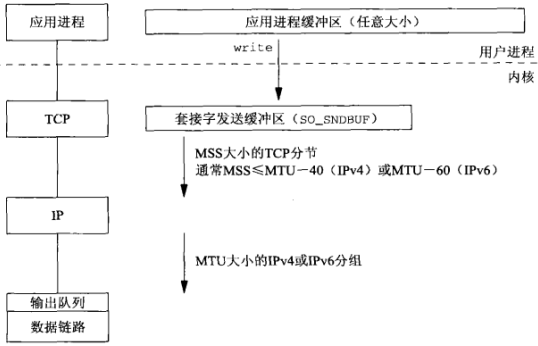
11、缓冲区大小和限制

IPv4 65535字节 包含首部

以太网 MTU 1500字节

当一个IP数据报从某个接口送出时，如果大小超过相应链路MTU，IPv4和IPv6都将执行分片

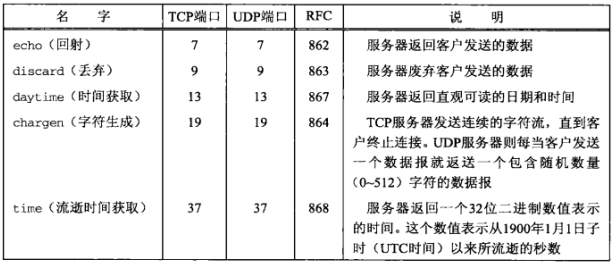
12、TCP输出：设定套接字发送缓冲区选项SO\_SNDBUF来设定缓冲区大小，然后应用进程调用write时，内核从该应用进程缓冲区中复制所有数据到所写套接字的发送缓冲区，



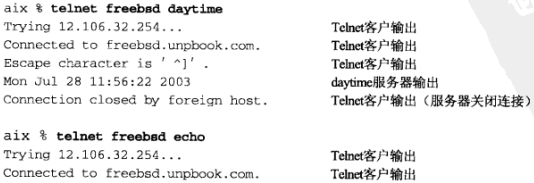
UDP输出：



13、标准因特网服务：如下图



提供标准的Telnet客户程序完成简易的测试机制，如：





14、常见互联网应用的协议使用：



第二部分：基本套接字编程

第三章 套接字编程简介

套接字地址结构可以在两个方向上传递，从内核到进程，从进程到内核。

从内核到进程方向的传递是值-结果参数的一个例子

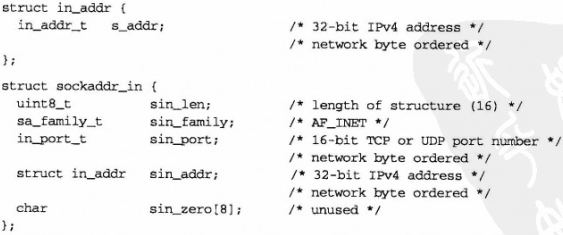
地址转换函数在地址的文本表达和他们存放在套接字地址结构中的二进制值之间进行转换。多数现存的IPv4代码使用inet\_addr和iner\_ntoa这两个函数，同时适用IPv4和IPv6两种代码。

1. 套接字地址结构

大多数套接字函数需要一个指向套接字地址结构的指针作为参数。每个协议族都定义它自己的套接字地址结构。

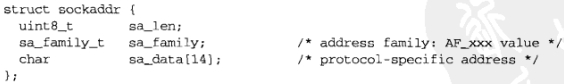
这些结构的名字均以sockaddr\_开头，并以对应每个协议族的唯一后缀结尾。

IPv4套接字地址结构



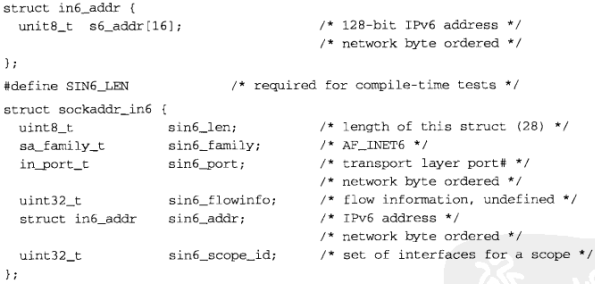
POSIX只需要这个结构中的3个字段：sin\_family、sin\_addr 和 sin\_port。

通用套接字地址结构

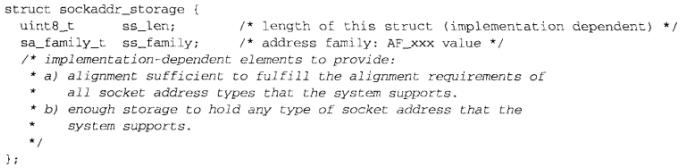


用于强制类型转换，因为套接字函数接收参数为任何协议类型的套接字地址结构的指针，因此要有一个通用的套接字地址结构，这样解决如何声明所传递指针的数据类型的问题。（本来可以用void\*，但是在ANSI C之前定义的UNIX函数通过一个通用的套接字地址结构进行类型强制转换来实现。 ）

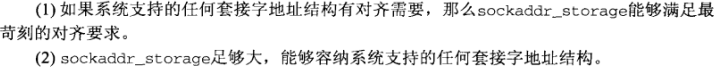
IPv6套接字地址结构



新的通用套接字地址结构：



与sockaddr相比存在以下两点差别：



1. 套接字地址结构的比较（本书中遇到5种套接字）



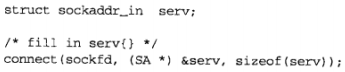
1. 值-结果参数：

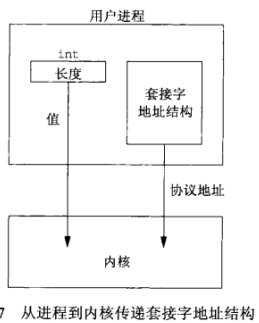
当往一个套接字函数传递一个套接字地址结构时，该结构总是以引用形式（也就是指针，是因为多态的原因吗？？？为了实现多态所以用指针？）来传递，该结构的长度也是作为一个参数来传递。

但是传递方式取决于该结构的传递方向：从进程到内核，还是从内核到进程。

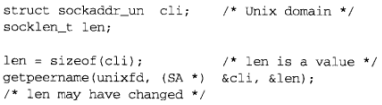
从进程到内核 connect bind sendto

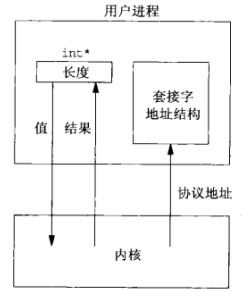
例如：





从内核到进程 accept recvfrom getsockname和getpeername

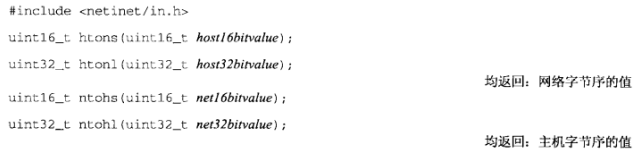




1. 字节序函数：

字节存储分为大端小端，因此网络协议必须指定一个网络字节序，而还有一个主机字节序。网际协议一般按照大端字节序。

两种字节序的转换用以下函数

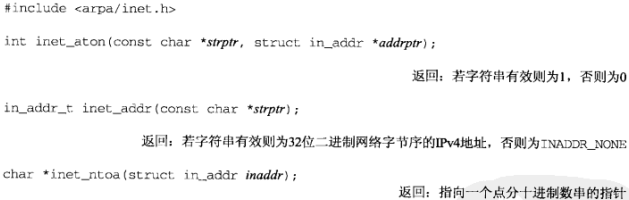


1. 字节操纵函数

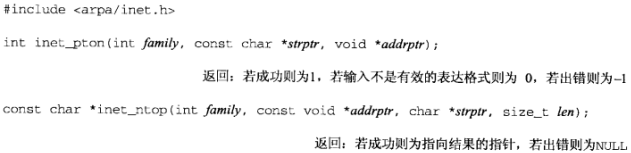
bzero bcopy bcmp

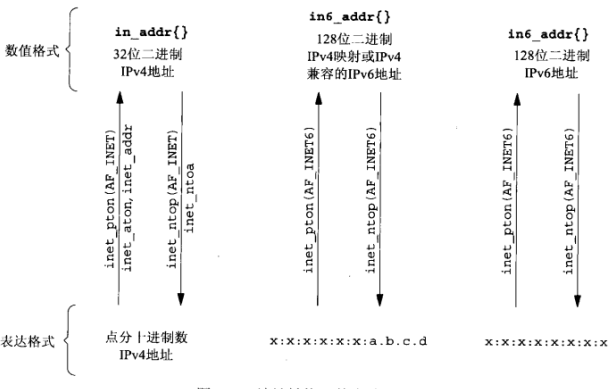
memcmp memcpy memset

1. 地址转换函数（把点分十进制的字符串转换为32位的二进制数据）



inet\_pton和inet\_ntop函数

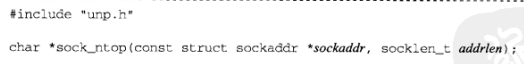




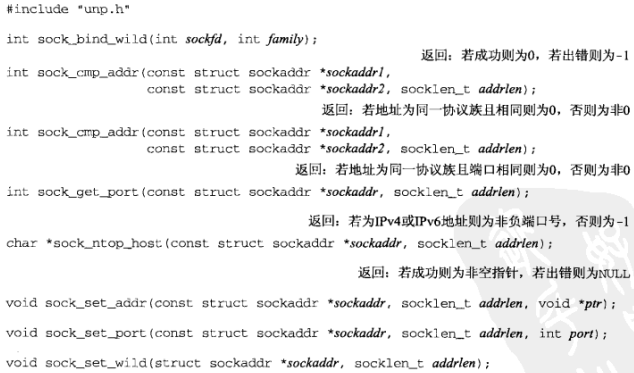
上面地址转换函数的基本问题是要求调用者传递一个指向二进制地址的指针， 同时需要套接字结构的格式和地址族，这样代码

就与协议相关了。

解决这个问题，自行编写一个函数，以指向套接字地址结构的指针作为参数



其中sockaddr指向一个长度为addlen的套接字地址结构，下面是一些类似的函数

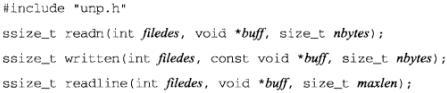


1. readn,writen和readline函数

字节流套接字如TCP套接字上调用read和write不同于通常的文件IO，字节流套接字上调用read或write输入或输出

的字节数可能比请求的数量少，这是因为内核中用于套接字的缓冲区可能以达到了极限。

为以防万一，不让实现返回一个不足的字节计数值，我们总是改为调用writen函数来取代write函数

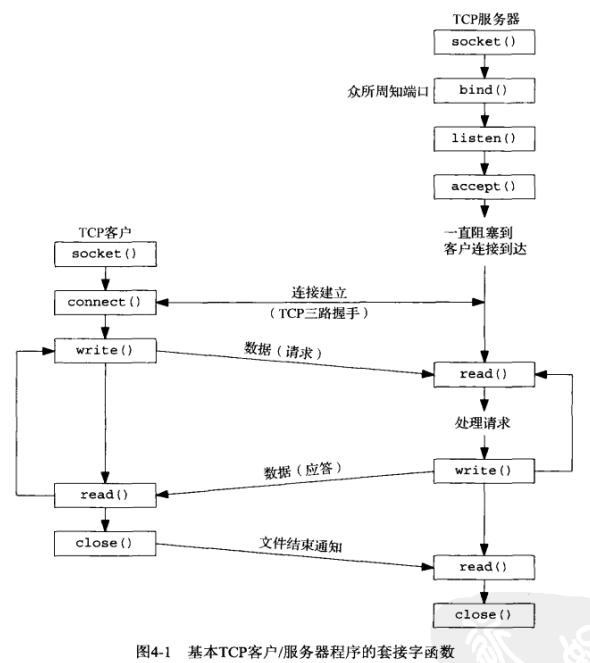


第4章：基本TCP套接字编程

本章讲解一个完整的TCP客户/服务器程序需要的基本套接字函数。上一章是套接字地址结构，下一章就开始用这些来开发这个客户/服务器程序。围绕之前的客户/服务器程序展开本书，多次对它改进。

还要讲解并发式的服务器，大量客户连接到同一服务器上时用于提供并发性的一种常用的UNIX技术

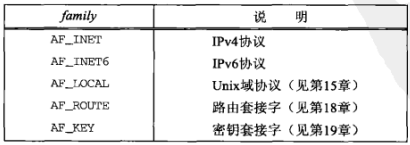
用到的函数：



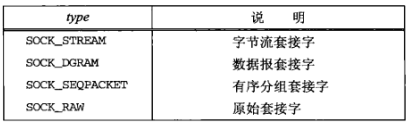
1. socket函数：首先要创建套接字，指定期望的通信协议类型。



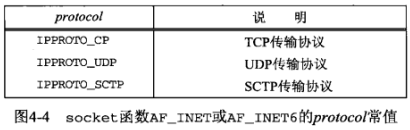
family参数常值选项



type参数常值选项：



以及



Socket函数成功时返回一个小的非负整数值，它与文件描述符类似，称为套接字描述符。

得到这个套接字我们只指定了协议族和套接字类型，没有指定协议地址和远程协议地址

1. connect函数



无须先用bind函数，内核会确定源IP地址，并选择临时端口。

调用connect将激发TCP的三路握手过程，连接成功或者出错才返回，出错返回情况如下：

未收到SYN分节响应：若无响应等待24s后再发送一个，最终一直没有响应返回ETIMEDOUT错误。

收到SYN响应为RST：表明服务器主机在我们指定的端口上没有进程在等待与之连接。立刻返回

客户发出的SYN在中间的某个路由器上引发了一个“目的地不可达”ICMP错误，认为是一种软错误

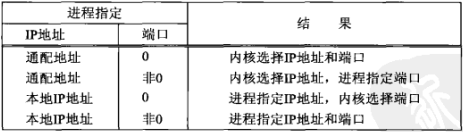
1. bind函数



将一个本地地址赋给一个套接字

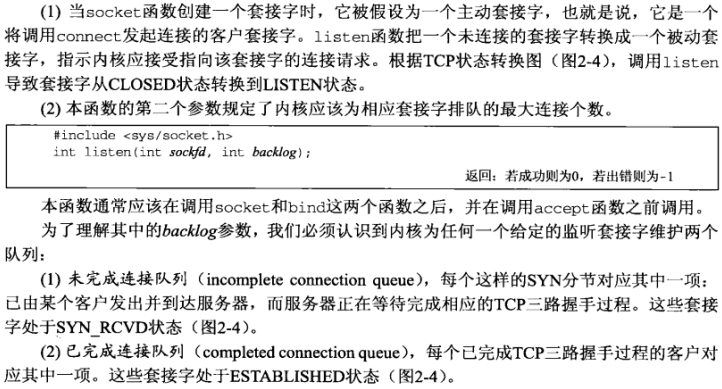
返回的常见错误是EADDRINUSE，为地址已使用

可以指定地址和端口的方式：

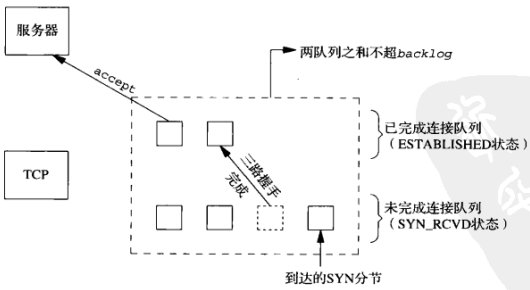


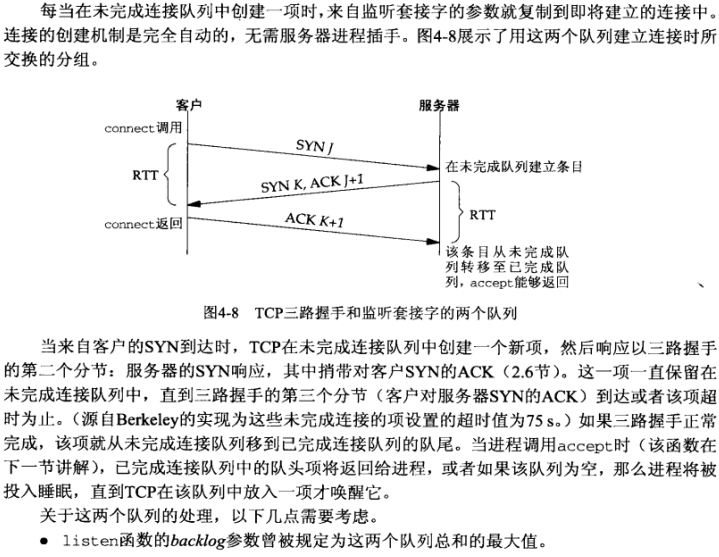
1. listen函数

仅TCP服务器调用，做两件事情。



监听套接字的两个队列如下图：



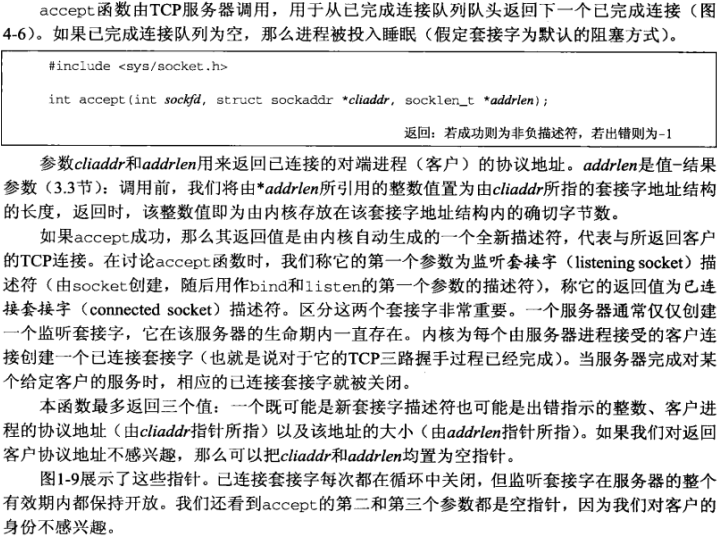


存在模糊因子，如1.5，放大1.5倍

不能把backlog定义为0

三路握手正常完成的前提下，未完成连接的任何一项在其中的留存时间就是一个RTT

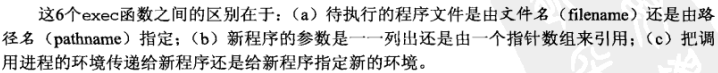
1. accept函数

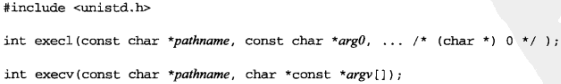


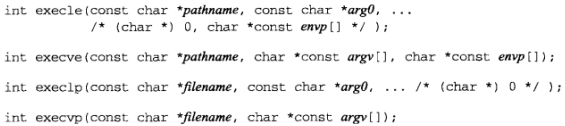
值-结果参数：一个参数的作用是返回函数的值。

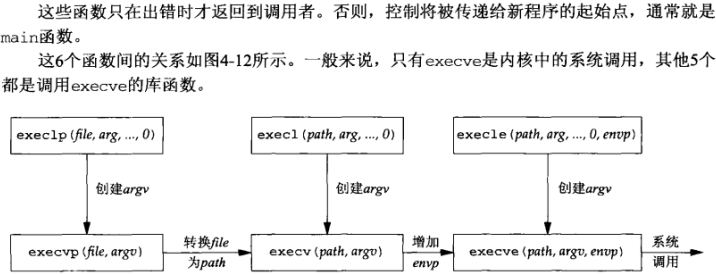
1. fork和exec函数：

存放在硬盘上的可执行程序文件能够被UNIX执行的唯一方法是：有一个现有的进程调用6个exec函数中的一个。







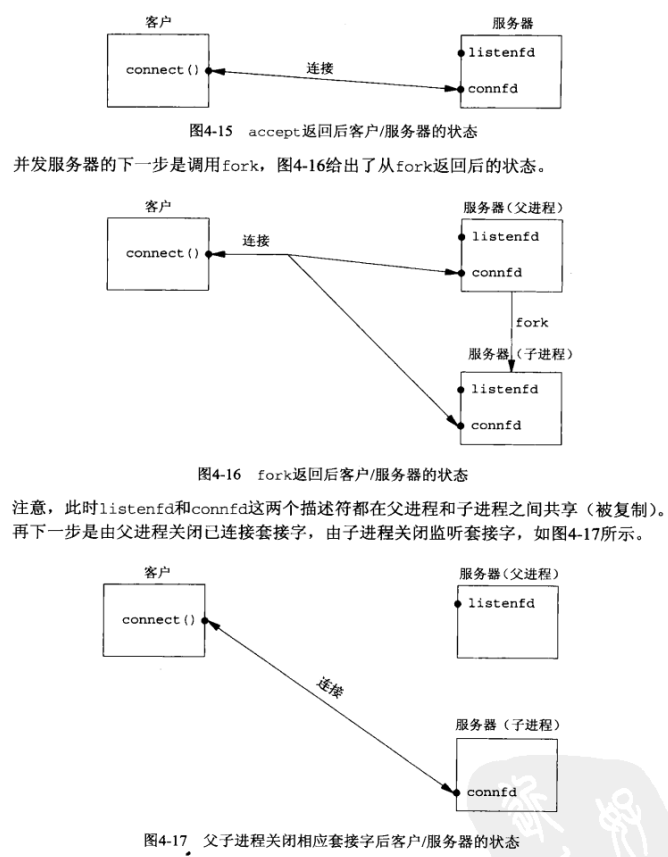


1. 并发服务器

一个简单的并发服务器的轮廓

此处要注意accept函数返回后，listenfd对应的套接字都已经不再使用，accept函数返回新的套接字connfd用于建立连接。

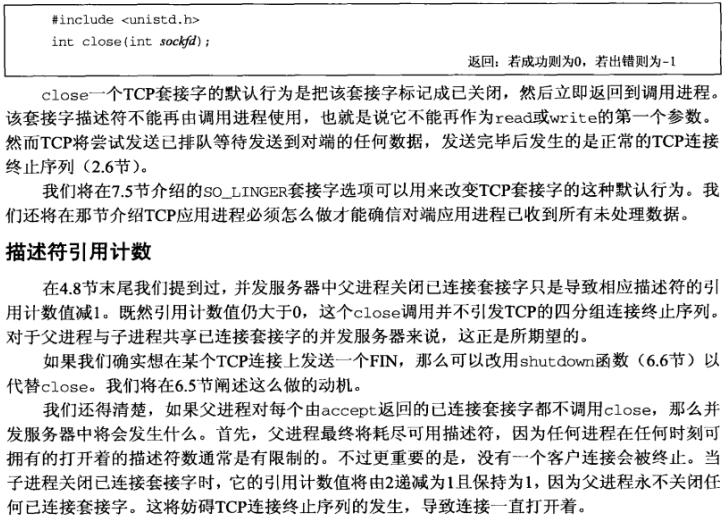
此时fork后，父子进程都有这个套接字，因此connfd引用计数为2，在父进程中关闭这个套接字，引用计数为1，套接字真正的清理和释放要当引用计数为0时才发生。



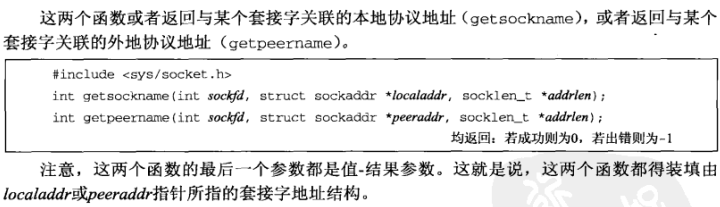
两个套接字的最终状态，子进程处理与客户的连接，父进程则监听套接字上再次调用accept来处理下一个客户连接。

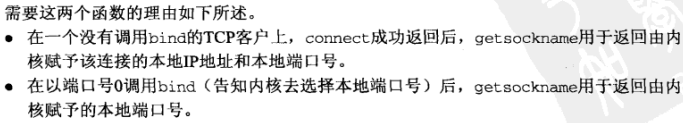
1. close函数

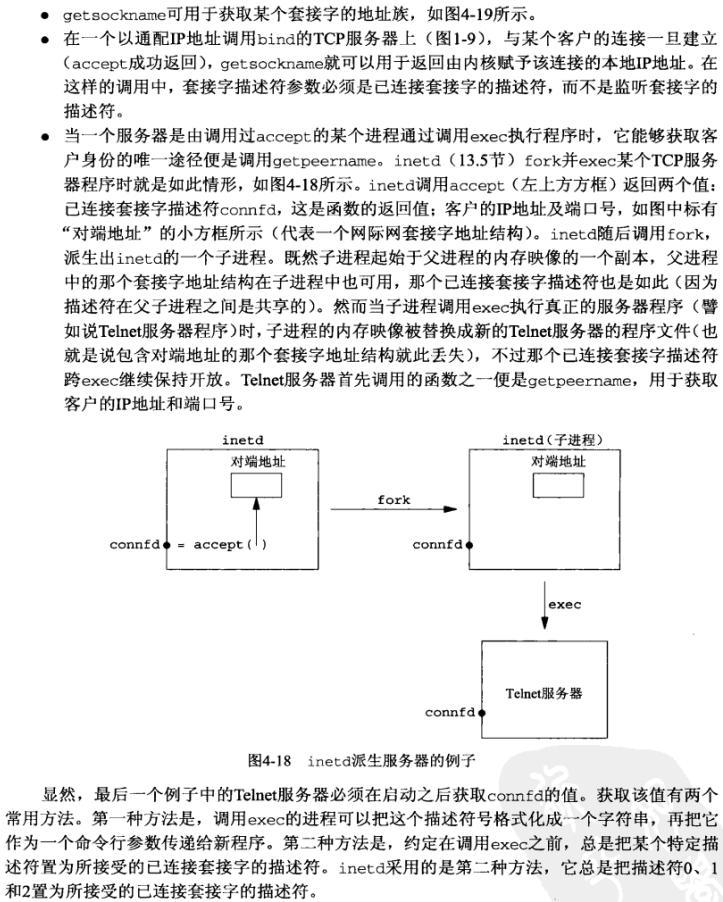
通常的UNIX close函数也用来关闭套接字，并终止TCP连接。



1. getsockname和getpeername函数

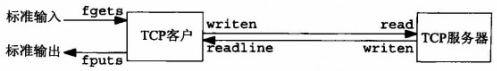




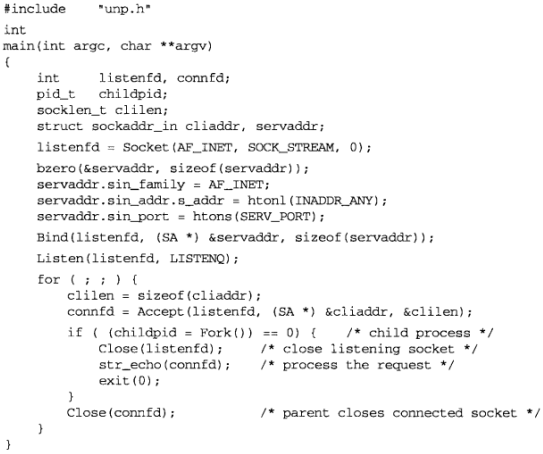


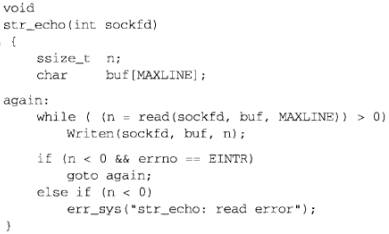
第五章：TCP客户/服务器程序示例

本章写一个TCP回射客户/服务器

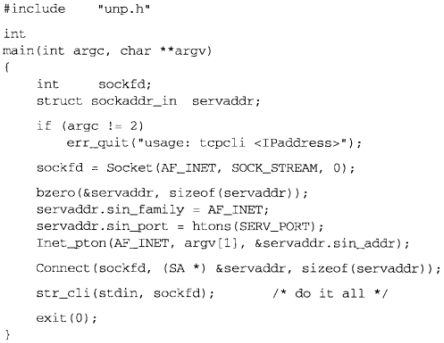


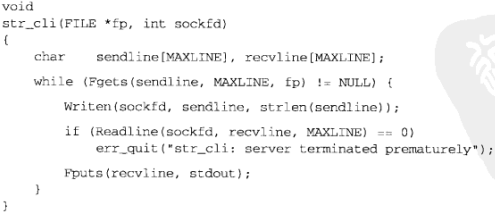
1. 服务器代码：





客户端代码：





1. 重点：虽然这个TCP的例子程序很小，但是对于我们要弄清楚客户和服务器如何启动，如何终止，更为重要的是当发生某些错误（例如客户主机崩溃，客户进程崩溃，网络连接断开等等）时会发生什么，本例子极端重要。

只有搞清这些边界条件以及他们与TCP/IP协议的相互作用，才能写出能处理这些情况的健壮的客户和服务器程序。

1. 用netstat -a命令可以看到套接字对的状态

ps命令展示各个进程目前的状态。

网络进程的STAT列是“S”时，表明进程在为等待某些资源而睡眠。

网络进程的WCHAN值，wait\_for\_connect表示进程阻塞于accept或connect，进程阻塞于套接字输入或输出时，为tcp\_data\_wait，阻塞于终端IO时，输出read\_chan。

1. 服务器正常终止：客户端发送FIN，服务器接收到后进行进程终止，调用exit，服务器子进程终止时，给父进程发送一个SIGCHLD

信号，如果没在代码中捕获该信号，就会使子进程进入僵死状态

1. POSIX信号处理：

信号：软件中断，异步发生

可以由一个进程发给另一个进程，由内核发给进程

处理信号的行为有3种：信号处理函数、设置SIG\_IGN进行忽略、信号的处置设定为SIG\_DFL来启用它的默认处置。

用signal函数来绑定一个返回void，参数为int的信号处理函数进行绑定。

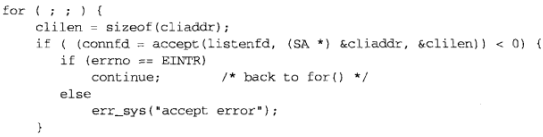
1. 慢系统调用：可能永远阻塞的系统调用。多数网络支持函数属于这一类。

当阻塞慢系统调用的一个进程捕获某个信号且相应信号处理函数返回时，该系统调用可能返回一个EINTR错误。我们对慢系统调用返回EINTR有所准备。

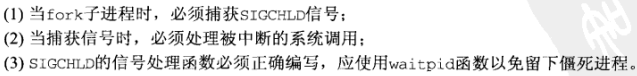
编写捕获信号的程序时，要对慢系统调用返回EINTR有所准备。

有些UNIX系统的实现会自动重启，有些则不会重启。

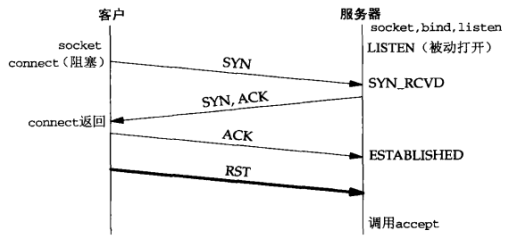
例如对慢系统调用accept，要对捕获信号后accept返回EINTR有所处理、



1. wait waitpid 返回已终止子进程的进程ID号，并且statloc指针是一个值结果参数，返回子进程终止状态，并且有三个宏可以用来检查子进程终止状态。
2. wait对于一起发生的多个SIGCHLD信号只处理一次，而waitpid则可以处理多次。一般多使用waitpid。避免多个子进程只处理一个而其他的僵死掉。
3. 需要避免的情况：



10、accept返回前连接中止：



类似于这种，，在accept返回前客户中止了连接，此时一般的UNIX系统都会返回一个EPROTO的errno值，而POSIX会返回ECONNABORT的errno值给别人，因为不与其他的错误重合，服务器可以忽略它，再次调用accept

11、服务器进程终止怎么办：

上面的例子程序中，客户有两个输入---套接字和用户输入，当客户进程阻塞在用户输入上时，套接字输入就会被忽视。如果在

客户进程阻塞在用户输入时，服务器进程终止，FIN到达套接字时，客户就正阻塞在fgets调用上。

12、SIGPIPE信号：当一个进程向某个已经收到RST的套接字执行写操作，内核向该进程发送一个SIGPIPE信号，该信号的默认行为

是终止进程，因此进程必须捕获它以免不情愿地被终止。不论该进程是捕获了该信号并从其信号处理函数返回，还是简单地忽略该信号，写操作都将返回EPIPE错误。

处理SIGPIPE建议方法取决于它发生时进程想做什么，如果没有特殊的事情要做，就将信号处理办法设置为SIG\_IGN。并假设后续的输出操作将捕捉EPIPE的错误并终止。

13：服务器崩溃怎么办：客户端会阻塞在接受服务器数据处，并且在超时后，返回错误ETIMEDOUT。如果某个中间路由器判断服务

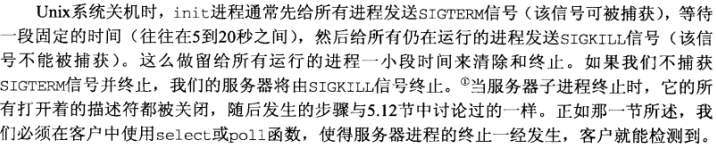
主机不可达，从而响应一个ICMP消息，那么返回的错误是EHOSTUNREACH或ENETUNREACH

14、服务器崩溃后重启



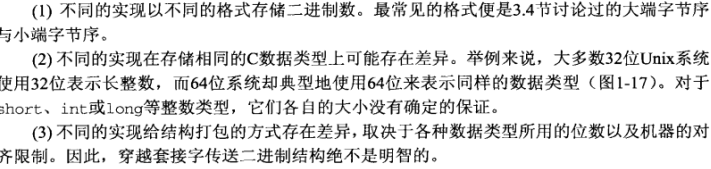
SO\_KEEPALIVE套接字选项或某些客户/服务器心博函数这些技术可以让客户即使不主动发送数据也能检测处服务器是否崩溃。

15、服务器主机关机：

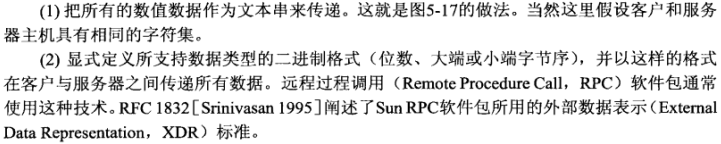


16、数据格式：

服务器和客户端传递数据的数据格式有字符串和二进制格式两种方式，用二进制传递要考虑以下问题：



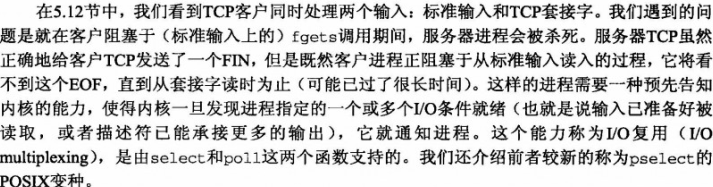
解决这种数据格式问题一般用两个方法：



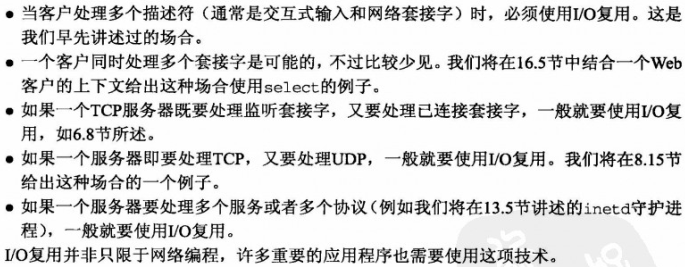
第六章：select和poll函数

一个进程处理两个输入，如果阻塞于一个输入，另一个输入有内容也看不到，直到读到这个输入才可以。

例如：



1. IO复用典型使用在下列网络应用场合：



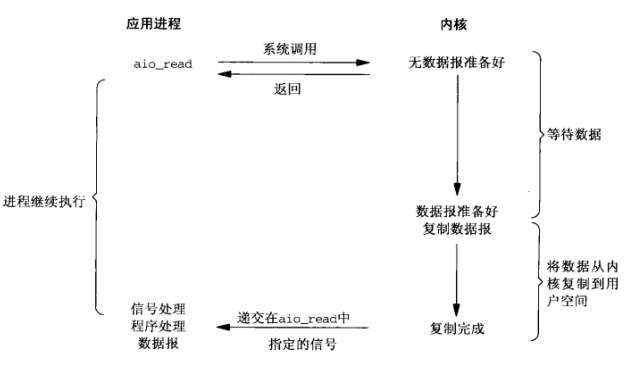
（一点思考：关于‘apue’中IO多路复用的内容是整体的系统的讲解，而‘unp’中这部分强调IO多路复用在网络编程中的使用情景。）

1. 输入操作通常包含两个不同的阶段：

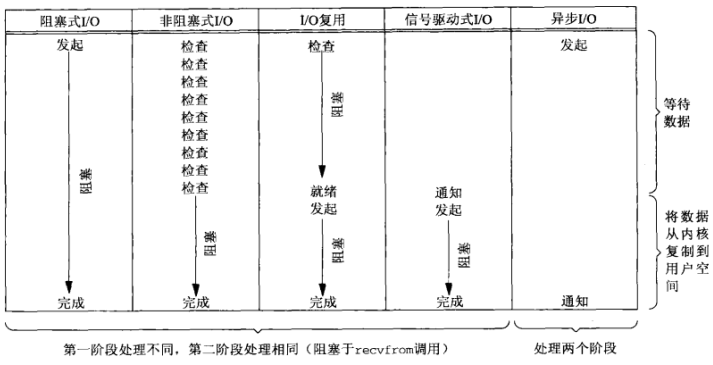
等待数据准备好，从内核向进程复制数据

IO模型看apue笔记：阻塞 非阻塞 IO复用 信号驱动式（SIGIO） 异步IO

异步IO图解



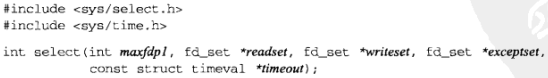
1. 各种IO模型的比较



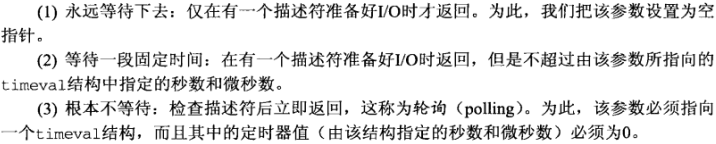
同步IO操作导致进程阻塞，直到IO操作完成

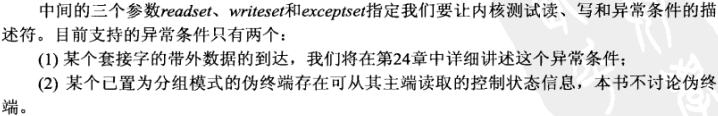
异步IO操作不导致请求进程阻塞

1. select函数



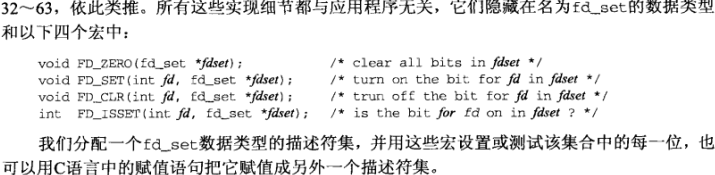
timeval这个参数设置要等待的时间，有三种可能

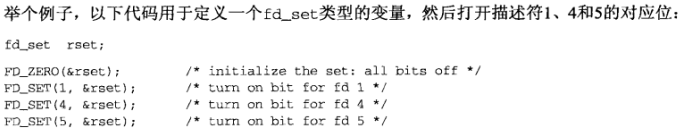




其中描述符集通常是一个整数数组， 每个整数的每一位对应一个描述符，

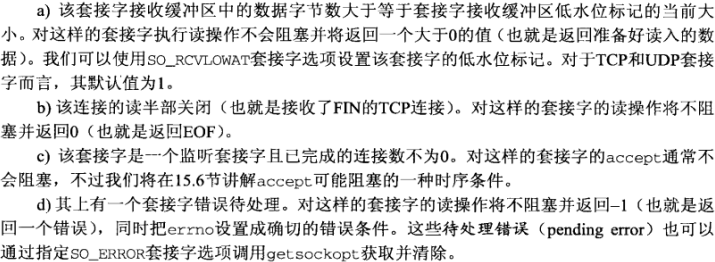




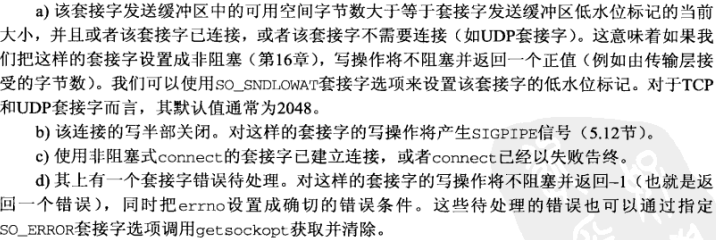


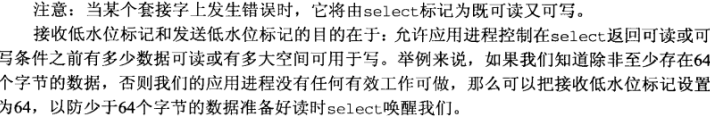
描述符就绪条件：

满足下列四个条件中的任何一个时，一个套接字准备好读：

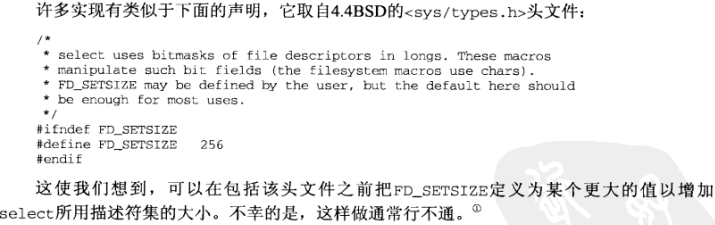


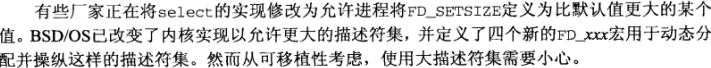
满足下列四个条件中的任何一个时，一个套接字准备好写：





描述符数限制：

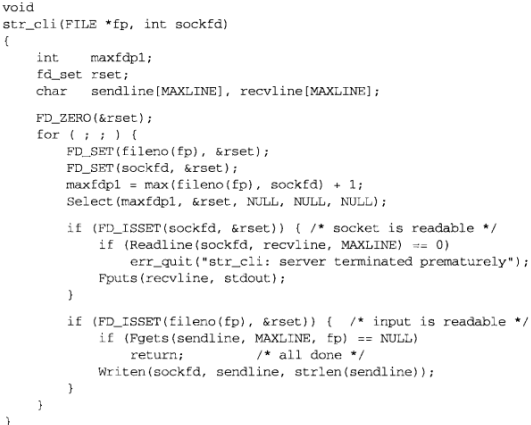




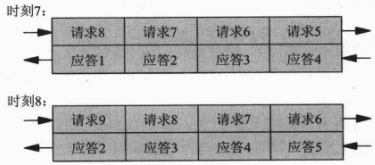
1. 混合stdio和select被认为是非常容易犯错误的，这样做必须小心。

比如输入了很多，但是select一次后读入一次后又阻塞在select。

如下例子22行

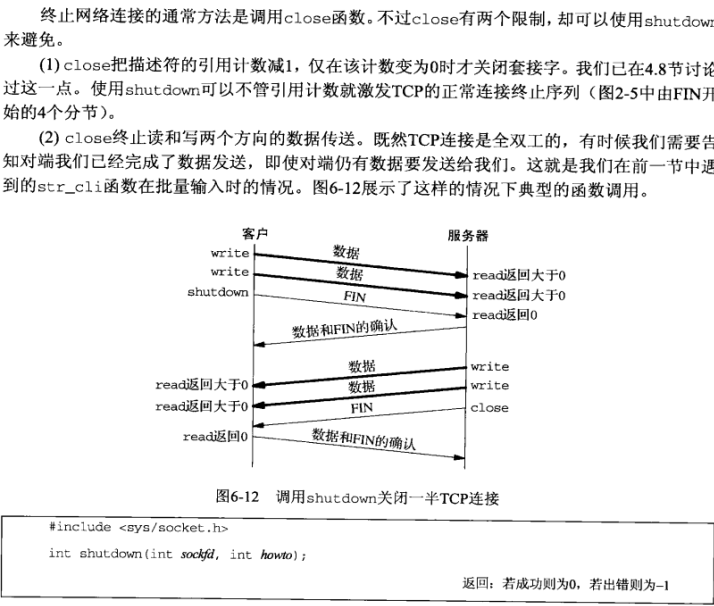


连续发送8个请求和应答后，客户端直接返回main然后调用close，随后main终止，但是还有请求在去往服务器的路上，仍有应答在返回客户的路上。



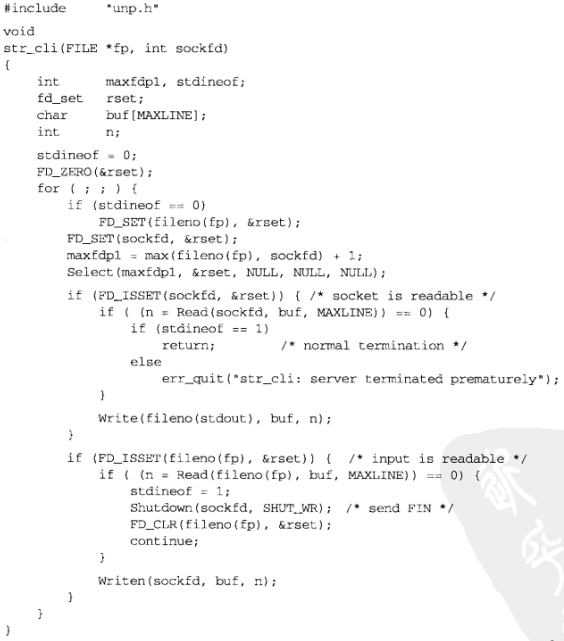
因此我们需要一种关闭TCP连接一半的方法，我们给服务器发送一个FIN，告诉它我们完成了数据发送，但是仍然保持套接字描述符打开以便读取。

1. shutdown函数。



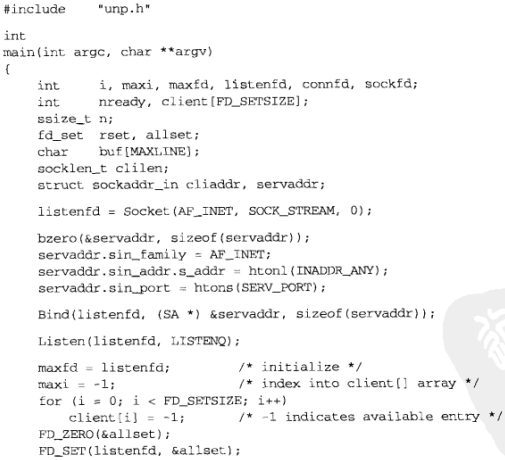
参数howto有SHUT\_RD和SHUT\_WR和SHUT\_RDWR

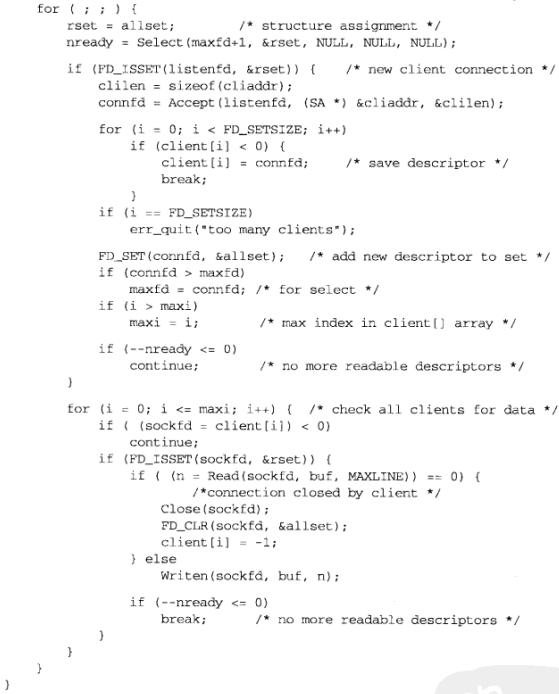
7、使用了select和shutdown的正确的客户端



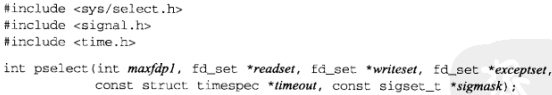
stdineof是一个标志，当为0时标准输入有可能有输入，每次主循环都会select标准输入

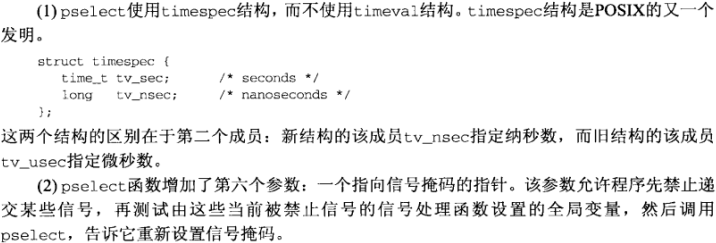
1. 使用了select的服务器，避免了创建新的进程，节省了开销。



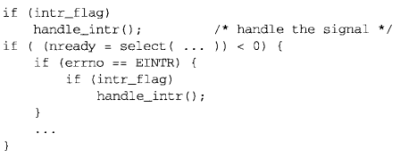


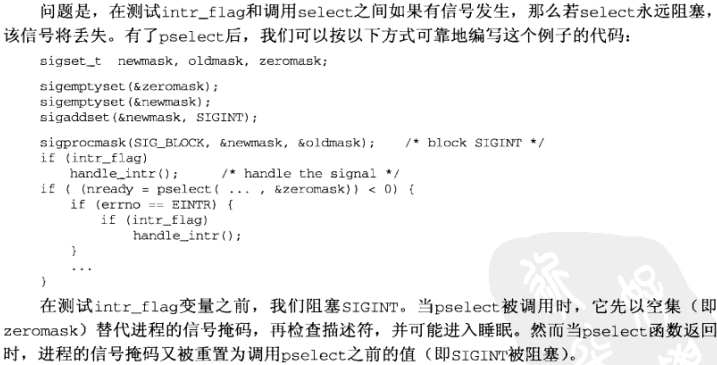
8、pselect函数，



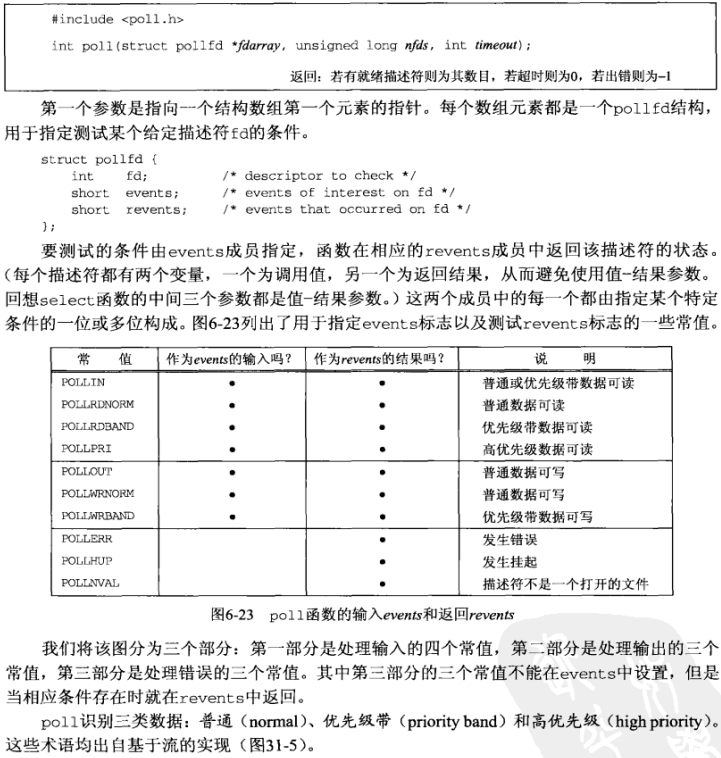


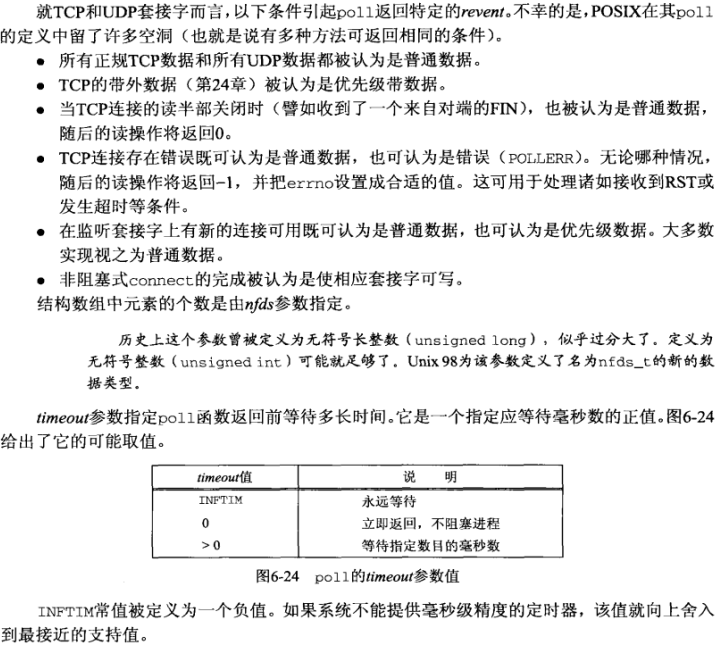
关于第二点：

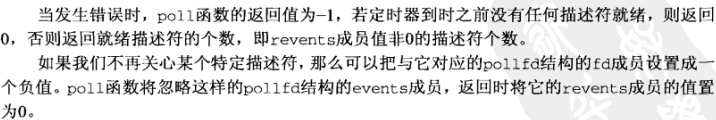




1. poll函数

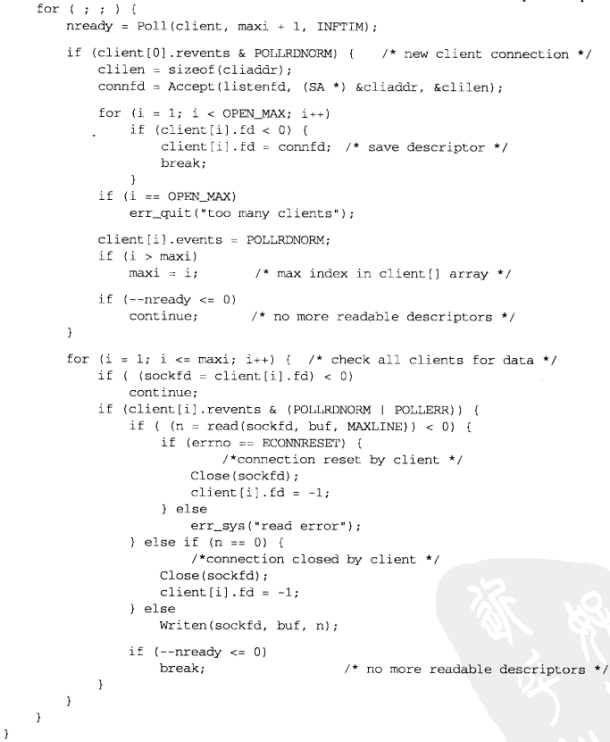






1. 使用了poll的服务器代码：





第七章：套接字选项

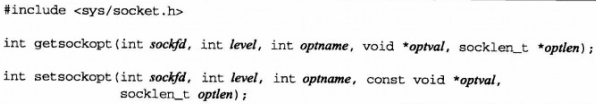
本章讲解获取和设置影响套接字选项的方法

如getsockopt setsockopt

fcntl函数

ioctl函数

1. getsockopt函数和setsockopt函数



仅用于套接字

