第三部分　Unix域协议

第16章　Unix域协议：概述

16.1　概述

Unix域协议是进程间通信 (IPC)的一种形式，可以通过与网络通信中使用的相同插口 API来访问它们。图 16-1的左边表示使用插口写成的客户程序和服务器程序，它们在同一台主机上利用Internet协议进行通信。图 16-1的右边表示用插口写的利用 Unix域协议进行通信的客户程序和服务器程序。

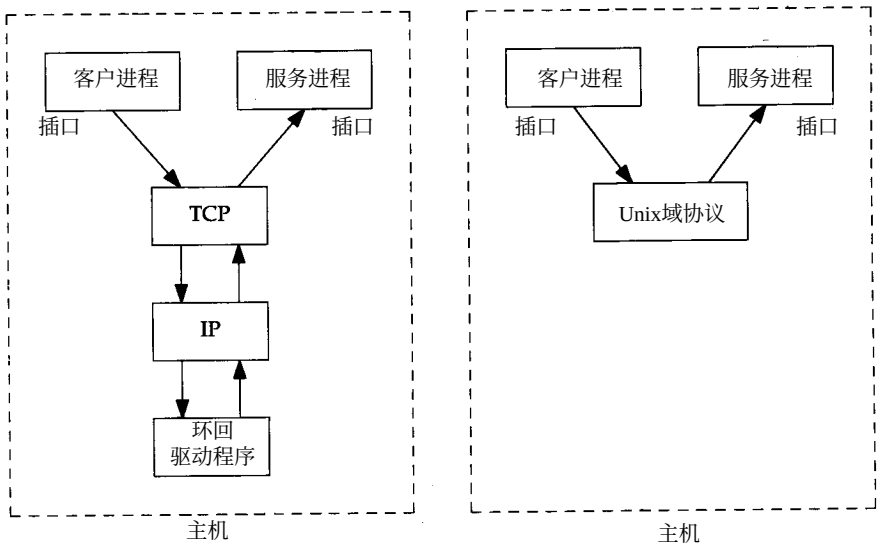

图16-1　使用Internet协议和Unix域协议的客户程序与服务器程序

当客户进程通过TCP往服务器进程发送数据时，数据首先由 TCP输出处理，然后再经过 IP输出处理，最后发往环回驱动器 (见卷2的5.4节)，在环回驱动器中，数据首先被放到 IP输入队列，然后经过 IP输入和TCP输入处理，最后传送到服务器。这样工作得很好，并且对于在相同主机上的对等端 (客户进程和服务器进程)来说是透明的。然而，在 TCP/IP协议栈里需要大量的处理过程，当数据没有离开主机时，这些处理过程实际上是不需要的。

Unix域协议由于知道数据不会离开主机，所以只需要较少的处理过程 (这样数据传送就快多了)。不需要进行检验和的计算和验证，数据也不会失序，由于内核能控制客户进程和服务器进程的执行过程，流量控制也被大大简化了，等等。虽然 IPC的其他形式也有这些优点 (消息队列、共享内存、命名管道，等等)，但是，Unix域协议的优点在于它们使用的接口与网络应用程序使用的插口接口完全一样：客户程序调用 `connect`，服务器程序调用 `listen`和

accept，两者都调用read和write，等等。而其他形式的IPC使用完全不同的API，其中有一些不能与插口以及其形方式的 I/O较好地交互 (例如，在系统 V消息队列中我们不能使用select函数)。

一些TCP/IP实现努力通过优化去提高性能，例如当目的地址是环回接口时可以忽略TCP检验和的计算和验证。

Unix域协议既提供一个流插口 (SOCK_STREAM，与TCP字节流相似)，又提供一个数据报插口(SOCK_DGRAM，与UDP数据报相似)。Unix域插口的地址族是AF_UNIX。在Unix域协议中用于标识插口的名字是文件系统的路径名 (Internet协议使用IP地址和端口号的组合来标识TCP和UDP插口)。

网络编程API标准IEEE POSIX 1003.1g 也支持Unix域协议，它使用的名称是“local IPC”。其地址族是AF_LOCAL，协议族是PF_LOCAL。因而使用术语“Unix”来描述这些协议也许已成为历史。

Unix域协议还能提供在不同机器之间进程间通信时所没有的功能。这一功能就是描述符传递，即通过Unix域协议在互不相关的进程间传送描述符的能力，这个例子我们将在第 18章讨论。

16.2 用途

许多应用程序使用Unix域协议：

1) 管道。在一个源于伯克利的内核里，使用 Unix域流插口来实现管道。在17.13节里我们将讨论pipe系统调用的实现。
2) X Window系统。当与X11服务器相连时， X11客户进程通常基于DISPLAY环境变量的值或基于-display命令行参数值来决定使用什么协议。这个值的形式是 *hostname:display.screen*，这里hostname是可选的，默认值是当前主机，使用的协议是最有效的通信方式，其中典型的是 Unix域流协议。值unix强制使用Unix域流协议。服务器绑定到Unix插口上的名字类似于/tmp/.X11-unix/X0。
   由于X服务器进程通常处理在同一台主机或者网络上的客户进程的请求，这就意味着服务器进程需要等待一个连接请求到达TCP插口或者Unix流插口。
3) BSD打印假脱机系统(lpr客户进程和lpd服务器进程，在[Stevens 1990]的第13章详细描述)使用一个名为/dev/lp的Unix域流插口在相同的主机上进行通信。像 X服务器一样， lpd服务器使用Unix插口处理在相同主机上客户进程的连接，或者使用 TCP插口处理网络上客户进程的连接。
4) BSD系统记录器(syslog库函数，可以被任何应用程序调用)和syslogd服务器程序(使用一个名为/dev/log的Unix域数据报插口在相同的主机上进行通信)。客户进程写一个消息到插口上，服务器进程读出来并进行处理。服务器进程也处理来自其他主机上使用UDP插口的客户进程的消息，关于这种机制的详细介绍见 [Stevens 1992]的13.4.2节。
5) InterNetNews守护程序(innd)创建一个Unix 数据报插口来读取控制报文，一个 Unix流插口来读取本地新闻阅读器上的文章。这两个插口分别是 control和nntpin，通常

是在/var/news/run目录里。

以上内容并不全面，还有其他的应用程序使用 Unix插口。

16.3 性能

比较Unix域插口与TCP插口的性能是非常有趣的。除了 TCP和UDP插口，修改公共域ttcp程序的一个版本，使之使用一个 Unix域流插口。我们在同一台主机上运行的两个程序副本之间传送16 777 216个字节的数据，结果如图16-2所示。

内 核	最快的TCP (字节/秒)	Unix域 (字节/秒)	增长 百分比
DEC OSF/1 V3.0	14 980 000	32 109 000	114 %
SunOS 4.1.3	4 877 000	11 570 000	137
BSD/OS V1.1	3 459 000	7 626 000	120
Solaries 2.4	2 829 000	3 570 000	26
AIX 3.2.2	1 592 000	3 948 000	148

图16-2 Unix域插口与TCP插口吞吐量的比较

我们感兴趣的是从一个TCP插口到一个Unix域插口速度的增长率，而不是绝对速度 (这些测试运行在五个不同系统上，覆盖了不同的处理器速度，在不同的行上进行速度比较毫无意义)。所有的内核都是源于伯克利，而不是 Solaris 2.4。我们可以看到，在源于伯克利内核上的Unix插口比TCP插口要快两倍多，在Solaris下增长率要慢得多。

SVR4以及源于它的Solaris，采用了完全不同的方法来实现 Unix域插口。[Rago 1993]的7.5节描述了基于流的SVR4中实现Unix域插口的方法。

在这些测试里，术语“ Fastest TCP(最块的TCP) ”意味着这些测试是在下列情况下进行的：将发送缓存和接收缓存都设置为 32 768(这个值要比一些系统中的默认值大)，直接指定环回地址而不是主机自己的 IP地址。在早期的BSD实现中，如果指定了主机的 IP地址，那么在ARP码执行之前分组不会发送到环回接口 (见卷1图2-4)，这稍微降低了性能 (这就是为什么定时测试运行时要指定环回地址)。这些主机都有一个本地子网的网络入口，其接口就是网络的设备驱动程序，卷1第87页中间网络入口140.252.13.32就是一个例子(SunOS 4.1.3)。较新的BSD内核有一条到主机本身 IP地址的路由，其接口就是环回驱动程序，卷 2图18-2中入口140.252.13.35就是一个例子(BSD/OS V2.0)。

我们将在讨论Unix域协议的实现后，在18.11节再返回到性能问题。

16.4 编码举例

为了说明如何缩小一个 TCP客户-服务器与一个Unix域客户-服务器之间的差别，我们修改了图1-5和图1-7中的客户-服务器，使它们利用 Unix域协议通信。图16-3表示Unix域客户程序，与图1-5的不同之处用黑体字表示。

2-6 我们包含了<sys/un.h>头文件，客户和服务器的插口地址结构现在是 sockaddr_un类型。

11-15 socket调用的协议族是PF_UNIX，调用strncpy将与服务器相联系的路径名(从命令行参数得到)写入插口的地址结构。

unixcli.c

```
 1 #include    "cliserv.h"
 2 #include    <sys/un.h>

 3 int
 4 main(int argc, char *argv[])
 5 {                                   /* simple Unix domain client */
 6     struct sockaddr_un serv;
 7     char    request[REQUEST], reply[REPLY];
 8     int     sockfd, n;

 9     if (argc != 2)
10         err_quit("usage: unixcli <pathname of server>");

11     if ((sockfd = socket(PF_UNIX, SOCK_STREAM, 0)) < 0)
12         err_sys("socket error");

13     memset(&serv, 0, sizeof(serv));
14     serv.sun_family = AF_UNIX;
15     strncpy(serv.sun_path, argv[1], sizeof(serv.sun_path));

16     if (connect(sockfd, (SA) &serv, sizeof(serv)) < 0)
17         err_sys("connect error");

18     /* form request[] ... */

19     if (write(sockfd, request, REQUEST) != REQUEST)
20         err_sys("write error");
21     if (shutdown(sockfd, 1) < 0)
22         err_sys("shutdown error");

23     if ((n = read_stream(sockfd, reply, REPLY)) < 0)
24         err_sys("read error");

25     /* process "n" bytes of reply[] ... */

26     exit(0);
27 }
```

unixcli.c

图16-3 Unix域事务客户程序

当我们在下一章讨论具体实现时，就会看到导致这些差别的原因。

图16-4为Unix域服务器程序，与图1-7的不同之处用黑体字表示。

unixserv.c

```
 1 #include    "cliserv.h"
 2 #include    <sys/un.h>

 3 #define SERV_PATH   "/tmp/tcpipiv3.serv"

 4 int
 5 main()
 6 {                                   /* simple Unix domain server */
 7     struct sockaddr_un serv, cli;
 8     char    request[REQUEST], reply[REPLY];
 9     int     listenfd, sockfd, n, clilen;

10     if ((listenfd = socket(PF_UNIX, SOCK_STREAM, 0)) < 0)
11         err_sys("socket error");

12     memset(&serv, 0, sizeof(serv));
13     serv.sun_family = AF_UNIX;
14     strncpy(serv.sun_path, SERV_PATH, sizeof(serv.sun_path));
```

图16-4 Unix域事务服务器程序

```
15      if (bind(listenfd, (SA) &serv, sizeof(serv)) < 0)
16          err_sys("bind error");

17      if (listen(listenfd, SOMAXCONN) < 0)
18          err_sys("listen error");

19      for (;;) {
20          clilen = sizeof(cli);
21          if ((sockfd = accept(listenfd, (SA) &cli, &clilen)) < 0)
22              err_sys("accept error");

23          if ((n = read_stream(sockfd, request, REQUEST)) < 0)
24              err_sys("read error");

25          /* process "n" bytes of request[] and create reply[] ... */

26          if (write(sockfd, reply, REPLY) != REPLY)
27              err_sys("write error");

28          close(sockfd);
29      }
30  }
```

unixserv.c

图16-4 (续)

2-7 我们包含了<sys/un.h>头文件，并且定义了与服务器相联系的路径名 (通常路经名应在客户程序和服务器程序都包含的头文件中定义，为了简单，我们在这里定义)。现在的插口地址结构是sockaddr_un类型。

13-14 调用strncpy将路径名填入到服务器的插口地址结构。

16.5 小结

Unix域协议提供了进程间通信的一种形式，它使用同网络通信相同的编程接口 (插口)。Unix域协议既提供类似于 TCP的流插口，又提供类似于 UDP的数据报插口。从Unix域协议能获得的优点是速度：在一个源于伯克利的内核上， Unix域协议要比TCP/IP大约快两倍。

Unix域协议的最大用户是管道和X Window系统。如果X客户进程发现X服务器进程与X客户进程在同一台主机上，它就会使用 Unix域流连接来代替TCP连接， TCP客户-服务器程序和Unix域客户-服务器程序代码变化是很小的。

下面的两章描述Net/3内核中Unix域插口的实现。