UNIX 下 TCP/IP 网络编程技术

吴湘宁 （湖北武昌吴家湾 430074）

本文将介绍 UNIX 环境下套接字的基本概念及编程技术，并结合实例说明在 UNIX 下如何用套接字实现客户机/服务器方式的进程通信。

一、套接字简介

套接字(Socket)是网络通信的基本操作单元，它提供了不同主机间进程双向通信的端点，这些进程在通信前各自建立一个 Socket，并通过对 Socket 的读/写操作实现网络通信功能。

套接字分为以下三种类型：

1. **字节流套接字**(Stream Socket)：是最常用的套接字类型，TCP/IP 协议簇中的 TCP(Transport Control Protocol)协议使用此类接口，它提供面向连接的(建立虚电路)、无差错的、发送先后顺序一致的、包长度不限和非重复的网络信包传输。

2. **数据报套接字**(Datagram Socket)：TCP/IP 协议簇中的 UDP(User Datagram Protocol)协议使用此类接口，它是无连接的服务，它以独立的信包进行网络传输，信包最大长度为 32KB，传输不保证顺序性、可靠性和无重复性，它通常用于单个报文传输或可靠性不重要的场合。

3. **原始数据报套接字**(Raw Socket)：提供对网络下层通信协议(如 IP 协议)的直接访问，它一般不是提供给普通用户的，主要用于开发新的协议或用于提取协议较隐蔽的功能。

二、套接字系统调用

下表是 UNIX 中套接字系统调用的简单说明：

系统调用名	功 能
accept	接收套接字上的一次连接
adjtime	修正时间与系统时钟同步
bind	给套接字捆扎一个地址
close	撤消一个套接字
connect	对一个套接字进行连接初始化
getpeername	获得连接的对等端点的名字
getsockname	获得套接字名字
getsockopt	获得套接字上的选择
listen	在套接字上收听连接请求
recv	在已连接的套接字上接收信息
recvfrom	在套接字上接收信息
select	同步 I/O 多路复用
send	向已连接的套接字发送信息
sendto	向套接字发送信息
setsockopt	设置套接字上的选择
shutdown	关闭全双工连接的部分功能
socket	建立一个通信端点

三、套接字编程方法

这里将分别介绍面向连接协议的字节流套接字与非连接协议的数据报套接字的编程方法，因原始数据报套接字在实际工作中使用较少，在此不作讨论。

不论何种套接字编程均采用客户机/服务器的协作模式，即由客户进程向服务器进程发出请求，服务器进程执行被请求的任务并将结果返回给客户进程。

字节流套接字的服务进程和客户进程在通信前必须建立连接。建立连接及通信的步骤如下：

1. 服务进程首先调用 socket()创建一个字节流套接字，并调用 bind()将服务器地址捆扎在该套接字上，接着调用 listen()监听连接请求，随后调用 accept()做好与客户进程建立连接的准备，无连接请求时，服务进程被阻塞；

2. 客户进程调用 socket()创建字节流套接字，然后调用 connect()向服务进程发出连接请求；

3. 当连接请求到来后，服务进程被唤醒，生成一个

新的字节流套接字，并用新套接字同客户进程的套接字建立连接，而服务进程最早生成的套接字则继续用于监听网络上的服务请求；

4. 服务进程和客户进程通过调用 read()和 write()交换数据；

5. 服务进程或客户进程调用 close()撤消套接字并中断连接。

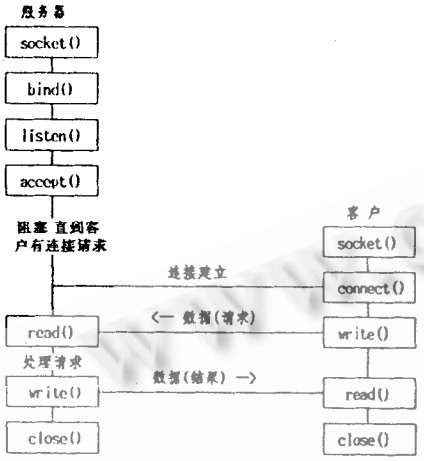

图 1　面向连接协议的字节流套接字系统调用

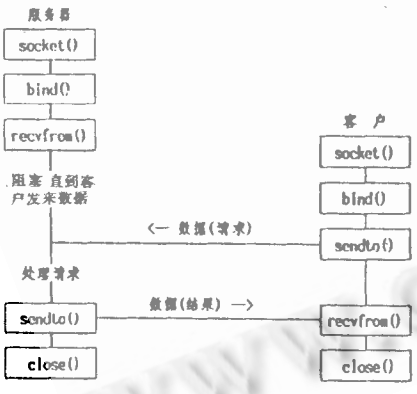

图 2　非连接协议的数据报套接字系统调用

数据报套接字的服务进程和客户进程通信前不必建立连接，通信的步骤为：

(1) 服务进程首先调用 socket()创建一个数据报套接字，并调用 bind()将服务器地址捆扎在该套接字上，然后调用 recvfrom()等待客户进程发来的请求；

(2) 客户进程在调用 socket()创建一个数据报套接字后，调用 bind()将客户机地址捆扎在此套接字上，接着调用 sendto()向服务进程发送请求，然后调用 recvfrom()等待服务进程返回该请求的处理结果；

(3) 服务进程在执行客户进程所请求的任务后，调用 sendto()将处理结果返回给客户进程；

(4) 服务进程和客户进程调用 close()撤消套接字。

四、套接字编程示例

下面给出一个运用字节流套接字在 TCP/IP 网络上实现客户机/服务器方式进程通信的实例。

在此例中，服务进程先于客户进程运行，当双方建立连接后，服务进程通过该连接向客户进程不断发送一个连续增长的序列数，客户进程每接收到 50 个序列数就在屏幕上显示一个'.'，显示至 20 个点后换行，直至任意一方进程被中断为止。

```
/* * * * * * * * * * * * * * * * * * * * * * server.c * * * * * * * * * * * * * * * * * * * * * * */
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>
# include <stdio.h>
main()
{
    int sock, namelen, seq, netint;

    struct sockaddr-in server; //存服务器的 internet 地址
    char msgsock;
    char buf[1024];

    //创建 internet 域的 TCP 协议的字节流套接字
    sock = socket(AF-INET, SOCK-STREAM, IPPRO-TO-TCP);
    if(sock<0){
      perror("socket");
      exit(1);
    }
    //将本地主机(服务器)的地址捆扎到创建的套接字上
    server.sin-family = AF-INET; //internet 域
    server.sin-addr.s-addr = INADDR-ANY; //使用任意合法地址
    server.sin-port = 1032; //公认的服务端口号
    if(bind(sock, &server, sizeof(server))<0){
```

```
        perror("bind");
        exit(2);
    }
    //建立长度为 5 的监听队列,从套接字上收听连接请求
    if(listen(sock, 5)<0){
        perror("listen");
        exit(3);
    }
    //阻塞至客户方有连接请求到来,建立一新套接字用于通信
    namelen = sizeof(server);
    if((msgsock = accept(sock, &server, &namelen))<0){
        perror("accept");
        exit(4);
    }
    //此时连接已建立,可以进行通信
    seq = 0;
    for(;;){
        netint = htonl(seq); //主机字节顺序转为网络字节顺序
        write( msgsock, &netint, 4); //向客户方写序列数
        seq++;
    }
}

/********************* client.c *********************/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
main(argc, argv)
    int argc;
    char *argv[];
{
    int sock, myseq, recvseq;
    struct sockaddr-in server; //存服务器的 internet 地址
    struct hostent *h; //存主机信息

    //命令行必须跟参数:服务器的主机名,该主机
    //必须在/etc/hosts 文件中定义,例如:
    // 192.7.100.31 hp486
    if(argc! =2){
        printf("Usage : %s servername \n", argv[0]);
        exit(1);
    }
    //创建 internet 域 TCP 协议的字节流套接字
    sock = socket(AF-INET, SOCK-STREAM, IPPRO-TO-TCP);
    if(sock<0){
        perror("socket");
        exit(2);
    }
    //根据命令行参数提供的服务器主机名,取得服务器的地址
    if(!(h=gethostbyname(argv[1]))){
        perror(argv[1]);
        exit(3);
    }
    bzero(&server, sizeof(server)); //先将服务器地址清 0
    server.sin-family = AF-INET; //internet 域
    //将取到的主机地址填入服务器的地址
    bcopy(h->h-addr, &server.sin-addr, h->h-length);
    server.sin-port = 1032; //填入公认的服务端口号
    //与服务进程建立连接
    if(connect(sock, &server, sizeof(server))<0){
        perror("connect");
        exit(4);
    }
    //此时,连接已建立,可通过对套接字的读/写实现通信
    myseq = 0;
    while (read (sock, &recvseq, 4) == 4){ //读序列数
        recvseq = ntohl(recvseq); //网络字节顺序转换为
    //主机字节顺序
        if(myseq! = recvseq){
            printf("sented = %d wanted = %d \n", recvseq, myseq);
            myseq = recvseq;
        }
        else
            myseq++;
```

```
        if(! (recvseq%50))
            printf(".");
        if(! (recvseq%1000))
            printf("
n");
    }
}
```

五、结束语

虽然 Sockets 最早是作为 BSD 规范提出来的，并已成为 UNIX 操作系统下 TCP/IP 网络编程标准，但是，随着网络技术的不断进步，Sockets 的应用范围已不再局限于 UNIX 操作系统和 TCP/IP 网络。目前，Windows、Windows NT、Windows95、OS/2 、Sun OS、Netware 等诸多的操作系统都开始提供套接字接口，它们在兼容 4.3BSD UNIX Sockets 的基础上附加了一些适应自身操作系统特性的扩充内容，这些新版的 Sockets 以操作系统内置或外挂的形式提供给程序员。Winsock(Windows Sockets) 便是一个用于 Windows 系列操作系统的 Sockets 版本。同时，套接字所支持的网络协议种类也不断增加，例如，Winsock 不仅支持 TCP/UDP 协议，而且支持 IPX/SPX 、AppleTalk、Decnet、NetBEUI 等网络协议，Netware 的套接字支持 TCP/UDP 及 IPX/SPX 协议。另外，套接字还增加了非 C 语言支持：如 C++、BASIC、Pascal 等。

由此可见，Sockets 的开放性能正逐步完善，可以说已经成为网络编程的通用接口，有了这个强有力的工具，我们可以构造任意跨操作系统、跨网络协议的分布式处理系统。

（来稿时间：1997 年 1 月）