国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第18卷第1期 1996年3月 Vol. 18 No. 1

利用TLI进行无连接方式应用开发*

张钦伍 黄瑞芳 张卫民
(国防科技大学计算机系 长沙 410073)

摘 要 TLI是ISO—OSI模型的传输层的一个编程界面，支持网络中两个用户进程间的数据传送。TLI为应用开发提供了两种服务方式：连接方式和无连接方式。前者适宜于要求较大数据量，面向数据流交互的应用程序；后者适合于含有短小快速请求/响应交互的应用程序。本文详细讨论了TLI的无连接方式服务和在应用开发过程中必须解决的几个问题。

关键词 TLI，传送端点，无连接方式服务
分类号 TP393

Developing Connectionless-Mode Service Using TLI

Zhang Qinwu Huang Ruifang Zhang Weimin
(Department of Computer Science, NUDT, Changsha, 410073)

Abstract TLI is a programming interface to the transport layer of ISO-OSI model, it supports the transfer of data between two user processes in network. TLI provides two modes of service for application and development: connection mode and connectionless mode. The former is appropriate for applications that requires relatively long-lived, datastream-oriented interactions; the latter is appropriate for application that involves short-term request/response interactions. In this paper, the auther discussed connetionless mode service of TLI and several problems that must be solved.

Key words TLI, transport endpoint, connectionless mode service

在计算机网络通信中，由于近年已经出现了网络协议向标准化方案OSI发展的趋势，所以TLI (The Transport Layer Interface) 界面越来越受到重视。它是ISO—OSI模型传输层的一个编程界面，支持网络中两个用户进程间的数据传送。该界面对介质和协议都是独立的，允许应用程序垮任何支持该界面的协议运行。与套接字 (Socket) 界面不同，它遵从MIPS ABI。因此，通常把它作为直接访问传输层的标准方法[1,2,3,4]。

* 1995年7月15日收稿

TLI 为应用程序的开发提供了两种服务方式:连接方式和无连接方式。连接方式是面向电路的,类似于 BSD 的“流式套接字”,以数据流代替孤立的数据单位。这种方式适宜于要求传送的数据量较大,面向数据流交互的应用程序。无连接方式是面向消息的,支持自包含的数据单位的传送。它相当于通过 BSD 数据报套接字传送的数据报。这种方式适宜于含有短小快速请求/响应式交互的应用程序。

1 通用数据结构和基本函数

TLI 把网络上相互通信的两个进程称为传送端点。在两个端点之间数据传送的提供者不是 TLI,而是它下层的各种协议及相应的例程。它只是用户进程和传送提供者之间支持这种传送的界面。该界面向编程用户提供的两种服务方式都是用函数实现的,所以对编程用户而言,TLI 实际上是一个函数库。它也为用户使用这些函数定义了一系列通用数据结构,以便在用户代码和 TLI 函数之间进行信息传递。

1.1 与无连接方式有关的通用数据结构

① struct netbuf {

```
    unsigned int maxlen;
    unsigned int len;
    char *buf;
}
```

这是一个最基本的结构,其它结构大多以该结构为核心构成。其中 buf 指向存放数据的缓冲区;len 是该缓冲区内容的实际大小的字节数;maxlen 是缓冲区的最大长度,填写缓冲区时不得超过该字节数。

② struct t_bind {

```
    struct netbuf addr;
    unsigned int qlen;
}
```

该结构用于向 t_bind () 函数提供本地地址或由 t_bind () 函数分配的本地地址返回在该结构中。其中 qlen 为队列长度,在无连接方式中无用,其值为0。

③ struct t_unitdata {

```
    struct netbuf addr;
    struct netbuf opt;
    struct netbuf udata;
}
```

这是一个自包含的数据单位。在发送时,udata 存放要发送的数据,addr 存放要发送到哪里的目的地址,opt 标识协议专用任选,通常可不关心。在接收时,udata 存放收到的数据,addr 存放源地址,即发送者的地址。

1.2 与无连接方式有关的基本函数

① int t_open (char *pathname, int oflag, struct t_info *info)

该函数的功能是打开标识特定传送提供者的 UNIX 文件。参数 pathname 指向该文

件的全路径名；oflag 与系统调用 open（）的相应参数一样，通常为 O_RDWR；info 所指的结构中存放的是该函数返回给调用者的与传送提供者有关的信息。不关心这些信息时，可指定该参数为 NULL。

正常情况，该函数返回一个文件描述符（一个小整数）标志该传送端点，供其它 TLI 函数引用。

② char *t_alloc (int fd, int structtype, int fields)

提供这个函数是为了简化 TLI 结构的动态分配。fd 标识传送端点；structtype 指明数据结构的类型，例如为 T_BIND，F_UNITDATA，即为相应的结构分配空间；fields 指明给定结构中相关的域如何分配。通常为 T_ALL（分配和初始化给定结构的所有相关域）或 T_ADDR（分配和初始化给定结构的 addr 域）。

③ int t_bind (int fd, struct t_bind *request, struct t_bind *return)

该函数给传送端点分配本地地址。一个端点通过 t_open（）建立之后还不能进行通信，只有分配了本地地址才能通信。所以 t_bind（）总是紧接在 t_open（）之后被调用。其中 fd 指定端点；request 是调用者指定的由传送提供者分配给该端点的地址；若该参数为 NULL，则由传送提供者隐式分配该端点地址，调用者不关心。不论哪种情况，传送提供者总是把实际分配给端点的地址返回在参数 return 中。当调用者用 request 为端点规定了特定地址时，必须在该调用返回后与返回地址 return 进行比较，以确认传送提供者分配了指定的地址。

④ int t_sndudata (int fd, struct t_unitdata *unitdata)

int t_rcvudata (int fd, struct t_unitdata *unitdata, int *flags)

这两个函数用于端点之间数据的发送和接收。fd 标志端点，t_sndudata（）中的 unitdata 参数指明要发送的数据（udata 域）及数据发送到哪里（addr 域）。t_rcvudata（）中的 unitdata 参数指明接收的数据放在哪里（udata 域）及该数据从哪里来，即发送者的地址(addr 域)。flags 是在返回时被设置的，用以指示接受的数据单位是否完整。这两个函数正常情况下都返回0，出错为－1.

2 无连接通信的基本模型和步骤

不论要开发的应用程序的相互通信有多复杂，最终都要落实到两个进程间的通信，而且两个进程之间的通信又可归结为如图1所示的模型中。

将这一过程用 TLI 函数反映出来，便如图2所示。

P_A，P_B 的关系总是一主一次，处于典型的 client/server 关系。比如 P_A 为 client，P_B 为 server，那么首先由 P_A 向 P_B 发送一个请求（可能是请求处理的数据、命令等），P_B 接收这个请求，按要求进行必要的服务处理之后，将处理的结果发送给 P_A。P_A 收到结果之后，再去做自己应做的事（很可能又向 P_B 发送新的请求）。于是 P_A、P_B 之间便完成了一个 client/server 过程。

这就是利用 TLI 介面进行无连接通信的基本步骤：前两步称为本地管理，后两步称为数据传送。

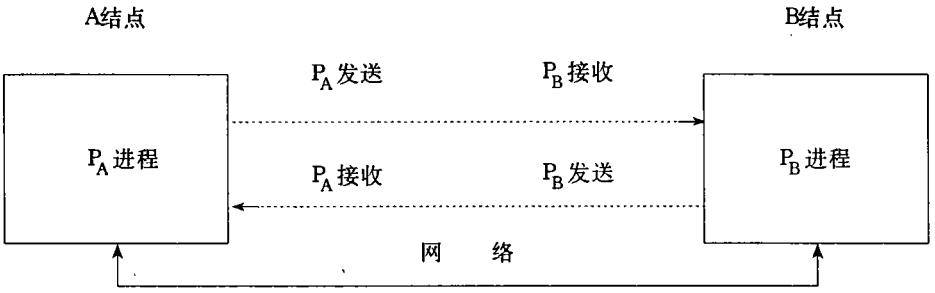

图1　无连接通信的模型

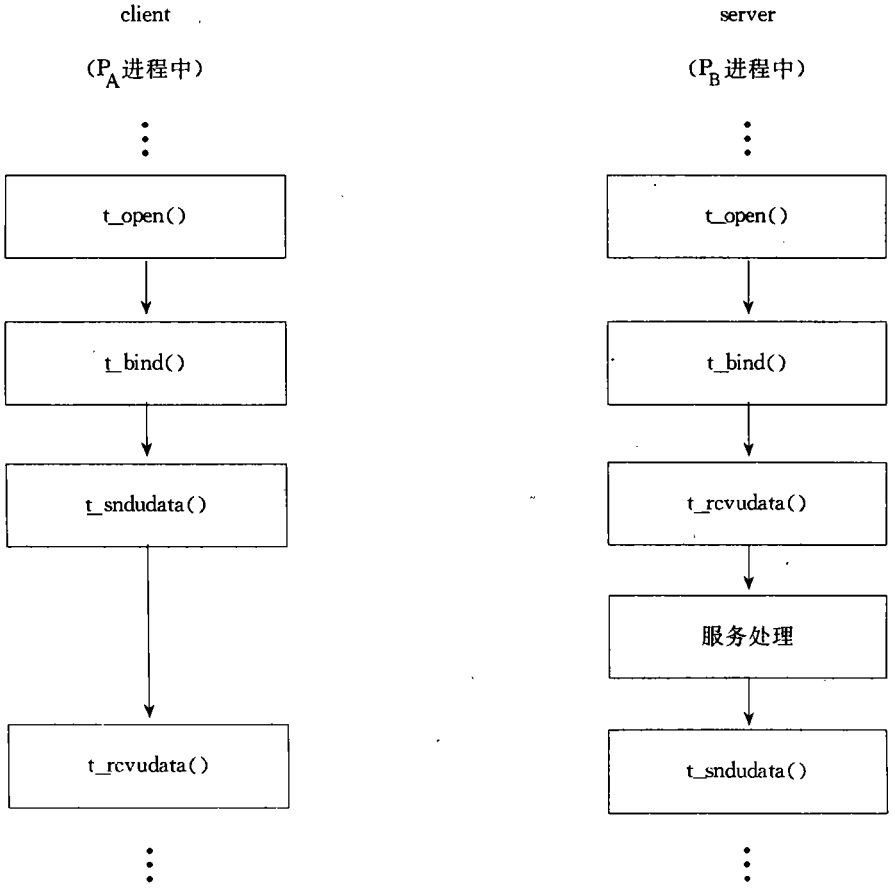

图2　无连接通信的步骤

3　必须解决的几个问题

实践证明只了解基本步骤还不足以进行应用开发，因为这些步骤与所在系统的网络配置是紧密相关的。经验告诉我们，要进行实际的应用开发还必须弄清下面几个问题。

3.1　选择传送提供者

本地管理的第一步就是用t_open()函数打开标识特定传送提供者的UNIX文件。这

个文件名是t_open ()的主要参数。哪里来?这要查看所在系统的网络配置文件/etc/netconfig。可以直接从该文件中查出语义值为tpi_clts(标志无连接方式)的网络的设备文件名，比如是/dev/udp，此时就用“/dev/udp”作为t_open ()的第一参数。也可通过程序的手段来查/etc/netconfig文件。此时，要在t_open ()调用之前增加下面几行语句：

```
if ( (handlep=setnetconfig ()) ==NULL) {
    nc_perror ( "setnetconfig failed");
    exit (1);
}
while ( (nconf=getnetconfig (handlep))!=NULL) {
    if (nconf→nc_semantics==NC_TPI_CLTS) break;
}
if (nconf==NULL) {
    fprintf (stderr, "no tpi_clts device file");
    exit (1);
}
```

第一条if是用setnetconfig ()函数打开或反绕/etc/netconfig文件并获得头指针。第二条while是用该指针查/etc/netconfig文件并将查到的一行内容放到nconf所指的结构中。在接着的t_open ()调用中，用nconf→nc_device作为第一参数。

3.2 为server端分配端口号

局部管理的第二步是用函数t_bind ()为传送端点分配地址。这一步在client端，为简单起见可用：t_bind (fd, NULL, NULL)，即让传送提供者隐式分配。原因是server方在t_unitdata结构中收到client发送的数据的同时，也收到了client这个地址。因此server在往回发送响应数据时，可直接用这个地址，不用现填。

然而，这一步在server端，调用者就必须显式地为端点分配地址，而且应该让client端知道。原因是client向server发送数据时要用到这个地址。因此，在server端，这步略微复杂些。下面就udp地址概括了这一步要做的事。

```
if ( (bind= (struct t_bind *) t_alloc (fd, T_BIND, T_ADDR)) ==NULL) {
    t_error ( "t_alloc of t_bind structure failed");
    exit (1);
}
bind→addr. len=16;

* (short int *) bind→addr. buf=2;
* (short int *) (bind→addr. buf+2) =2048;
bind→qlen=0;

if (t_bind (fd, bind, bind) <0) {
    t_error ( "t_bind failed");
```

```
        exit (1);
    }
    if (! ( ( * (short int * ) bind→addr. buf==2) &&
          ( * (short int * ) (bind→addr. buf+2) ==2048))) {
        fprintf (stderr, "t_bind bound wrong, address\n");
        exit (1);
    }
```

udp 地址格式为16个字节，前两个字节放该网络的语义值（这里是2，是固定的），接着的两个字节是分配给该端点的端口号（这里是2048，是用户任选的，通常系统中较小的值都已被占用，选个大一点的，以避免重复），再接着的4个字节是结点的网络地址，在/etc/hosts 文件中可查到，这是固定的，在分配本地地址时不必填。最后8个字节与对等进程有关，这里也不填。可以看出，这里主要是为该端点分配一个端口号，通常 server 都是通用的，为使所有请求的 client 都能知道该端口号，必须把它放到/etc/services 文件中，即在该文件中增加一行：

〈该服务名字〉　2048/udp

以便 client 自动查找。

3.3 填目的地址

在数据传送阶段一个是发送，一个是接收。要特别引起注意的是 client 端的发送。因为 client 在结构 t_unitdata 的 udata 域填发送数据的同时，还要在 addr 域填上目的地址，即 server 的地址，告诉传送提供者数据发往哪里。

这16个字节地址的前8个字节都要填。可用类似于4.2节中那段程序第5－8行的办法，直接填。在/usr/include/netconfig. h 文件中可查到 udp 的语义值；在/etc/services 文件中可查到 server 的端口号；在/etc/hosts 文件中可查到 server 所在结点的点分十进制网络地址。

也可以用程序的手段自动地查填，办法之一如下：

```
Struct nd_hostserv hostserv;
Struct nd_addrlist * addrs;

hostserv·h_host= "server 所在的结点名";
hostserv·h_serv= "server 名";

/ * 查目的地址 * /
if ( (handlep=setnetconfig ()) ==NULL) {
      nc_perror ( "setnetconfig failed");
      exit (1);
}
while ( (nconf=getnetconfig (handlep))!=NULL) {
      if (netdir_getbyname (nconf, &hostserv, &addrs) ==0) break;
}
```

```
    if (nconf==NULL) {
        fprint (stderr, "no address for hostserv\n");
        exit (1);
    }
        ......
    /* 分配 T_UNITDATA 空间 */
    if ( (ud=(struct t_unitdata *) t_alloc (fd, T_UNITDATA, T_ALL)) ==
NULL) {
t_error ( "t_alloc of t_unitdata struct failed");
exit (1);
    }
    /* 填目的地址和数据 */
    ud→addr= * (addrs→n_addrs);
    ud→udata. len=data_buf_len;
    strcpy (ud→udata. buf, data_buf);
    ud→opt. len=0;
    /* 发送 */
    if (t_sndudata (fd, ud) <0) {
        t_error ( "t_sndudata falied");
        exit (1);
    }
```

不难看出上面以 while 为中心的三个语句与4.1中的三个语句类似，实际上把选择传送提供者和查目的地址放在一个 while 中一次实现更简单。

4 结束语

TLI 内容丰富，对于连接方式的应用和一些高级用法，由于篇幅所限本文均未涉及。这里讨论的都是无连接方式应用中碰到的最重要的问题，提出的方法都是经实践证明了的，望能对从事这方面研究和开发的读者有所补益。

参 考 文 献

1 Silicon Graphics. Inc. IRIX Network Programming Guide. SGI, 1993

2 蔡传俊. UNIX/TCP/IP/NFS 网络编程与应用开发. 北京：海洋出版社，1993

3 朱三元，杨明等. 网络通信软件设计指南. 北京：清华大学出版社，1994

4 王劲松. 网络互联协议 TCP/IP 详解. 北京：科学技术文献出版社，1993

（责任编辑　朱宝龙）