基于微内核的网络系统界面的实现*

胡宁宁 金志权 谢 立

(南京大学计算机科学与技术系 南京 210093)
(南京大学计算机软件新技术国家重点实验室 南京 210093)

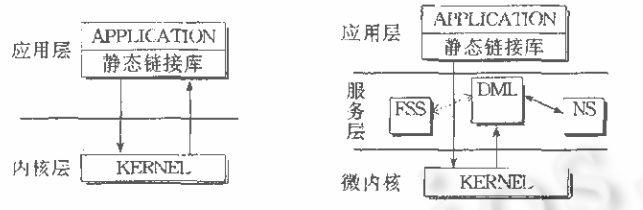
摘要 CONET V2.0 是一种在微内核环境下实现的网络系统软件,由于微内核在系统构造方法上与传统的方法不同,所以该环境下的系统模块在实现上也具有其特殊之处.文章介绍了 CONET V2.0 的体系结构与实现思想,并着重阐述了作为该系统应用程序接口之一——SOCKET 的设计与实现.

关键词 网络系统,应用程序接口,套接字,微内核.

中图法分类号 TP393

网络系统软件是在一定操作系统环境下实现的一组通信协议,为用户的网络应用提供支持.不同的操作系统环境势必会造成网络系统具体实现方法的不同.本文在简要介绍了微内核机制下实现的网络系统软件 CONET V2.0 的总体结构与实现思想之后,着重介绍了作为该系统应用程序接口之一——SOCKET 的设计与实现.

在微内核机制出现以前,通常,整个操作系统分为内核层与应用层,在内核层实现各种系统服务,在应用层,应用程序通过系统调用来使用这些服务.但在软硬件迅猛发展、各种不同的操作系统不断出现的情况下,人们对不同平台上系统软件的可移植性、可构造性提出了更高的要求,进而促使软件开发者更加重视操作系统构造的灵活性与重用性,而不再像以往那样仅局限于高效性.传统操作系统的构造方法很难达到这点要求,虽然诸如分层法、虚拟机法等构造方法在一定程度上改善了系统模块的可移植性,但却不能让用户为实现某种特定需要而动态地构造服务系统[1].



(a) 传统操作系统的体系结构 (b) 基于微内核的操作系统的体系结构

图1 两种操作系统的体系结构

微内核机制的引入在很大程度上解决了这一问题,其主要设计思想为:将整个操作系统分为 3 层,即内核层、服务层与应用层[2].在内核层实现操作系统最基本的功能,如进程管理、内存管理、进程间通信以及 I/O 管理等;在服务层以服务器的形式实现操作系统中面向应用程序的其他功能(如图 1(b)所示,其中 NS 即指以服务器形式实现的 CONET V2.0,FSS 指文件系统服务器);应用层的应用程序通过在运行时链入的动态映射库 DML(dynamic mapped library)来调用各种服务器中的系统服务.

由此可见,在微内核机制下实现的各种服务器最重要的就是使其所提供的应用程序接口能适应这种系统调用方式.对于网络服务器 CONET V2.0 来说,就要求 SOCKET 与 TLI(transport layer interface)模块具备这种能力.

---

* 本文研究得到国家"九五"科技攻关项目基金资助.
作者胡宁宁,1976 年生,硕士生,主要研究领域为分布与并行计算机系统,网络系统.金志权,1941 年生,教授,主要研究领域为分布与并行计算机系统,网络系统.谢立,1942 年生,教授,博士生导师,主要研究领域为分布与并行计算机系统,智能操作系统.

本文通讯联系人:胡宁宁,南京 210093,南京大学计算机科学与技术系
本文 1998-03-24 收到原稿,1998-06-22 收到修改稿

1 CONET V2.0 系统概述

1.1 CONET V2.0 的层次结构

CONET V2.0 按照标准实现了 TCP/IP 协议簇,其层次结构以及在系统中的位置如图 2 所示.其中 SOCKET/TLI 是应用程序接口,TCP,UDP,IP,ICMP,ARP,RARP 都是根据标准实现的传输层与网间网层的协议模块;VNET 用于管理各物理网络协议(如 Ethernet,X.25)的多路复用;ETH 与 ETHDRV 负责与网卡的交互.

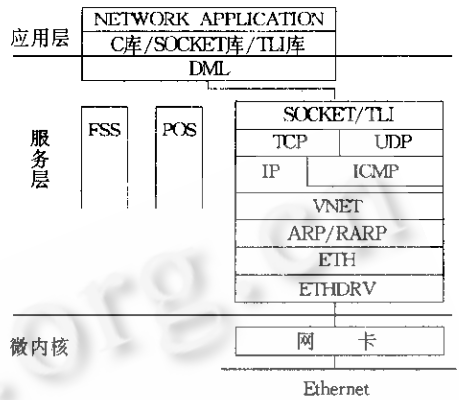


图2　系统层次结构图

1.2 CONET V2.0 中的面向对象机制

CONET V2.0 的整体设计采用了面向对象的思想[3],利用面向对象理论在各层协议之间构造了模块化的统一界面,大大简化了协议模块间接口的处理.CONET V2.0 将系统中的各种主要数据分成 3 类:

• 协议对象(protocol object):图 2 中的各层协议模块皆以协议对象的形式存在于网络系统中,它们在网络系统初启时创建一次,在整个系统运行过程中一直存在.它们的主要功能是,协调与管理会话对象与消息对象.下文以 $P(x)$ 表示 x 协议对象.

• 会话对象(session object):会话对象对应于网络通信中的传输端点(end point),由它来解释消息对象中的数据并维护与数据通信相关的状态信息.在 CONET V2.0 中,会话对象是作为协议对象的实例(instance),由协议对象动态创建与释放的,每次连接(连接方式)或数据传输(非连接方式)时,由系统生成一组相互连接的会话对象.下文以 $S(x,y)$ 表示 $P(x)$ 与 $P(y)$ 两协议对象之间的会话对象.

• 消息对象(message object):对应于传输的数据,无论是应用程序要发送到网络上的数据,还是从网络上接收到的数据,都以消息对象的形式在会话对象之间或会话对象与协议对象之间传递.

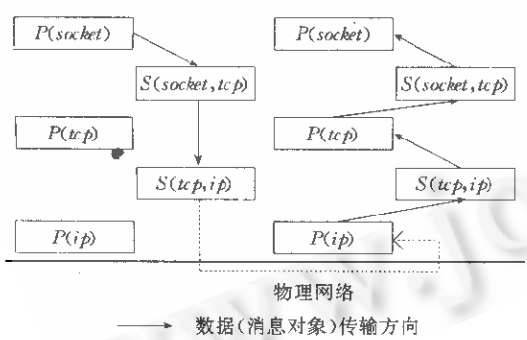


图3　CONET V2.0中3种对象间的关系

协议对象与会话对象、消息对象之间的关系集中体现在通信时网络系统内部数据通路的建立上(如图 3 所示).对连接方式而言,这条通路在连接建立时创建(对非连接方式而言,在第 1 次传送数据时创建).对于连接请求方,这是一个主动创建的过程,$P(socket)$ 将用户的连接请求交给 $P(tcp)$ 协议对象,后者就创建 $P(socket)$ 与 $P(tcp)$ 之间的会话对象 $S(socket,tcp)$.同时,将该请求再交给 $P(ip)$.依此类推,这样就建立了一条协议-会话对象链,作为通信时网络系统内部的数据处理通路.对于接收连接方,则存在两个问题:(1) 这种通路在没有连接请求时是不应该建立的;(2) 连接请求最先是由最下层协议对象获得,而在未建立这条通路前是无法将该请求给 $P(socket)$,然后再像连接请求方那样由上而下地创建.为解决这两个问题,我们采用协议对象之间的授权机制:当应用进程准备就绪可以接收连接时,$P(socket)$ 对 $P(tcp)$ 授权,$P(tcp)$ 再对 $P(ip)$ 授权.依此类推,使得下层协议对象可以自下而上地创建与上层对象之间的会话对象.

可以看出,通过面向对象机制,能够较好地将网络系统中静态成分与动态成分分离,从而提高了系统的模块化,增强了可移植性与可扩充性.

2 CONET V2.0 中 SOCKET 的设计与实现

Socket(套接字)作为通信协议应用程序接口,相对于 OSI 的 7 层网络模型,是处于会话层的地位,它作为传

输端点(end point)是网络系统内部的数据发送者与接收者,同时也是应用程序与网络系统交互的界面.

2.1 SOCKET与DML的交互及多线程的实现

2.1.1 SOCKET与DML的交互

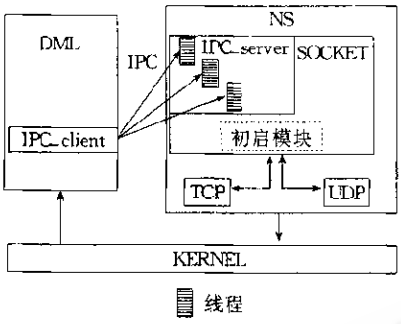


图4 SOCKET与DML的交互及CONET V2.0多线程的实现

CONET V2.0与传统UNIX系统(如BSD),在SOCKET模块的实现上最突出的不同点在于系统调用过程中用户参数的获取.在BSD中,系统调用经内核进入SOCKET模块后,后者获得的就是应用程序的原始参数.而在CONET V2.0所处的微内核环境中,系统调用经内核后首先由DML获取,再由DML通过IPC(interprocess call)将这些用户参数传送给SOCKET模块.由于IPC本身的限制,它不能很好地处理某些(诸如链表)数据结构(这种限制实际上与远程过程调用(RPC)所遇到的问题具有相同之处).具体实现时,我们在SOCKET与DML两个模块中分别加入一个内部模块IPC_client与IPC_server(如图4所示),其中IPC_client将用户参数改变成IPC可以处理的数据形式,而IPC_server则负责解释IPC消息中的数据,并将解释后的数据传送给相应的SOCKET内部处理函数.

2.1.2 多线程的实现

线程作为现代操作系统中一种运行单位,能够很好地实现程序运行的并行性.不同于BSD的核内单线程垂直处理的体系结构[4],基于微内核的CONET V2.0在服务层实现了基于消息的多线程处理(如图4所示).在网络系统初启时,SOCKET中的IPC_server利用内核的线程管理工具生成多个相同的线程,等待接收DML的IPC消息,从而可以同时处理多个用户的服务请求.执行完毕后,线程仍回到初启状态继续等待.这种基于消息的并行机制相对于某些网络系统所实现的协议模块之间的并行性,具有以下优点:(1) 并行程度高,不受系统内部分层数的限制;(2) 各协议层间通信少;(3) 易于实现并行模块之间的同步.

2.2 面向对象机制对SOCKET层实现的影响

由第1.2节已知,在CONET V2.0中,各协议模块皆以协议对象的形式存在,这些协议对象是在网络系统初启时创建的,同时初始化协议对象以及各协议对象之间的接口.由于各协议对象都可看做是同一个类的实例,它们之间具有相同的接口函数,因而大大简化了各层协议之间界面的处理.SOCKET作为CONET V2.0的一个协议模块,与其他协议之间的接口也应遵守协议对象之间的统一接口.但考虑到程序兼容性问题,这种接口并不能涉及到应用程序,即这种面向对象机制以及由此而获得的各对象之间的统一接口只能作为网络系统服务器内部实现方法而存在,而不能对应用程序产生影响,应用程序仍以系统调用的方式来使用网络服务.这样就对作为通信协议应用程序接口的SOCKET提出了特殊的要求:对上层应用程序而言(在微内核环境下,实为对DML而言),SOCKET应是一个普通系统模块,提供一组标准socket函数;对下层协议而言,SOCKET则应是一个协议对象 $P(socket)$,使用协议对象间的统一接口进行交互.

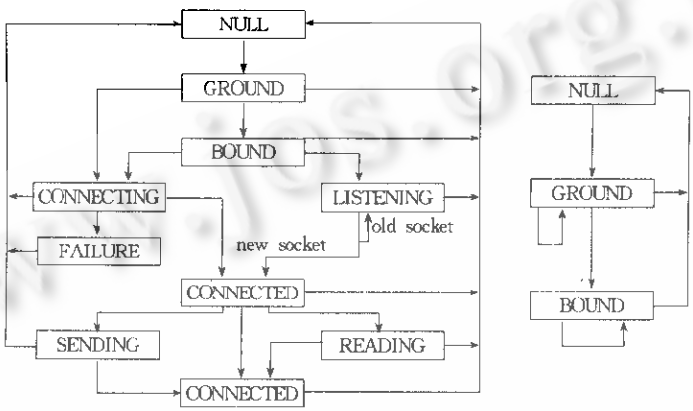
在实现时,我们综合了上述两种要求.在整体上用普通的模块化思想来实现SOCKET,即将SOCKET的各种功能用一组函数实现,并且这些函数皆可为外部模块所调用,这样就能较好地实现SOCKET与DML间IPC的交互,从而实现了网络系统与应用程序的接口.与此同时,在SOCKET内部,按协议对象的形式实现一个初启模块(如图4所示).在网络系统初启时,由系统利用这个初启模块创建 $P(socket)$,并初始化它与下层协议对象间的接口.该初启模块中的数据都可被SOCKET层的其他函数访问,从而可方便地将下层协议对象上传的数据交给应用程序.利用这种实现方法,我们较好地实现了从应用程序的系统调用到网络系统内部各对象间统一接口的转变.

2.3 套接字状态转换图及其在CONET V2.0中的实现

在应用程序中,套接字(socket)由套接字描述符(socket descriptor)来表示,在功能上,它类似于文件系统中的文件描述符.每个套接字描述符在系统内部与一个socket address结构的数据相关联,由该数据保存套接字的

状态、类型等信息.另外,应用程序依自身应用领域不同而对网络系统所提供的服务有不同的要求,这种要求在套接字系统中由套接字类型来表示,共有 5 种套接字类型(SOCK_STREAM,SOCK_SEQPACKET,SOCK_DGRAM,SOCK_RDM,SOCK_RAW),各种类型是否实现主要依该类型所应用的通信域(communication domain)不同而不同.例如,在 Internet domain 中主要涉及 3 种类型:SOCK_STREAM(提供面向连接的、可靠、双向有序数据流通信)、SOCK_DGRAM(提供非连接、不可靠的数据通信)和 SOCK_RAW(参见文献[5]).

套接字作为传输端点,在数据传输过程中要经历各种不同阶段(如连接准备,建立、接收连接,数据传送与接收等),不同阶段的套接字具有不同的特性与能力,我们将称这些套接字所处的不同阶段为套接字状态.为了更好地刻画状态之间的关系,根据两种主要的套接字类型(SOCK_STREAM 与 SOCK_DGRAM),我们给出套接字的状态转换图,如图 5 所示.



(a) SOCK_STREAM型　　(b) SOCK_DGRAM型

图5　套接字状态转换图

从图中可以清楚地看出一个套接字要进行数据通信所应进行的准备工作.CONET V2.0 中主要使用 *client_t* 与 *socket_t* 两个数据结构来实现 SOCKET 的各种状态转换及相应的各种功能.在网络系统中,通信双方之间进行数据交互要分别进行以下几步:

(1) 套接字描述符的获取.这一步在图 5 中就是由 NULL 向 GROUND 状态的转变,套接字描述符对于应用进程而言只是一个整数 *s*,在 SOCKET 层中它对应于一个 *socket_t* 型数据.在实现 *socket*()时,为每一个应用进程分配一个 *client_t* 型数据,并根据用户参数初始化 *sds*[*s*]所指向的 *socket_t*,由它来完成应用进程在套接字描述符上的各种动作.

(2) 连接的建立.由 CONET V2.0 面向对象的设计思想可知,一次连接的建立在网络系统内部实际上就是一次协议-会话对象链的建立.对于连接请求方而言,SOCKET 层只需发出连接请求即可.而对于接收连接方而言,它要通过上层对下层的授权,告诉下层协议本套接字接收数据的端口号等信息,以便在该套接字为多条连接服务时数据能正确地向上传递,这些工作都是由 SOCKET 中的 *bind*()与 *listen*()两函数来完成.这也是为什么接收连接方在完成连接之前要经过 BOUND 与 LISTENING 两状态,而连接请求方只要获得了套接字描述符即可发起连接的原因.当接收连接的套接字到达 LISTENING 状态后,它实际上成为一个分配器(dispatcher),即每成功接收一次连接(*accept*()),就产生一个新的套接字描述符和相应的 *socket_t* 数据,并使这个新的套接字进入CONNECTED状态(可以与对等进程进行数据通信),而原来的套接字则仍处于 LISTENING 的状态,继续监听连接请求(如图 5 所示).这样,就使一个服务进程可以同时接收多个连接请求,这也是为什么我们往往称接收连接方为服务方的原因.

(3) 数据的传递.在连接建立完成之后,数据的通信相对就简单一些.发送时,由 *P*(*socket*)将数据以消息对象形式向下传送;而对于接收,则需通过自身 *socket_t* 结构中的 *so_rcv* 来实现,*so_rcv* 指向一个 FIFO 的消息对象队列,每次 *P*(*tcp*)或 *P*(*udp*)都将自身处理完的数据以消息对象的形式插入该队列,而 *P*(*socket*)则从该队列

中取出消息对象，抽取出数据交于应用进程.

(4) 套接字描述符的关闭.根据不同的应用要求，我们实现了套接字的两种关闭函数：(I) *close*()，它释放掉与该进程相联系的 *client_t* 及 *socket_t* 数据，从而使应用进程不再与网络系统有关；(II) *shutdown*()，有时用户只希望将套接字双向传输中的一个方向关闭，此时并不回收 *client_t* 与 *socket_t* 数据，因而不能使用 *close*()，*shutdown*()通过设置 *socket_t* 中参数 *so_state* 的两个状态位，同时配合数据发送与接收的实现来实现这种功能.

2.4 SOCKET 的选项控制功能

SOCKET 还向应用程序提供了用于设置、获取网络系统各协议模块参数的辅助服务功能，应用程序可通过 *getsockopt*()，*setsockopt*()，*ioctl*()，*fcntl*()，*getpeername*()等函数调用来请求服务.我们在实现时根据各种参数实现的不同位置与方法，将这些服务分为 3 类：

(1) 由 SOCKET 层内部数据维护的参数：SOCKET 层的 *socket_t* 数据结构中的 *so_state*，*so_type*，*so_error* 等记录了的 SOCKET 主要运行属性，可直接回答某些应用程序的服务请求.如应用进程使用 *ioctl*()或 *fcntl*()来获取套接字所属进程或进程组标志符时，即可直接由 *socket_t* 的参数 *so_pgrp* 及 *so_cid* 获取.

(2) SOCKET 层内部实现的功能：主要指应用进程设置了套接字的某种属性，但这种属性需要在运行过程中体现.我们利用 *socket_t* 结构中的参数，同时配合相关 *socket* 函数的实现来达到这一要求.如套接字阻塞状态的设置，若套接字被设置为阻塞状态，当其接收数据并发现相应的接收数据队列为空时，则使应用进程在调用接受数据的函数时进入睡眠等待状态，直到下层将接收到的数据插入其接收队列并被唤醒为止；否则，应用进程并不等待，而是直接将出错标志作为系统调用的返回值.

(3) 由其他网络协议实现的服务功能：对这些功能(如接收、发送数据缓冲区大小的设置等)，SOCKET 层仅是一层界面，向下传递服务请求，向上传递服务结果，不作任何附加处理.

3 结束语

CONET V2.0 中 SOCKET 的设计方法较好地实现了整个网络系统与微内核的操作系统结构相协调，解决了由微内核的系统调用到网络系统中各对象统一接口的过渡，同时利用多线程实现了网络系统内部的并行.今后的工作是进一步完善系统.

参考文献

1 Cheung W H, Anthony H S Loong. Exploring issues operating systems structuring: from microkernel to extensible systems. ACM Operating Systems Review, 1995,29(4):4～16

2 Liedtke Jochen. On μ-kernel construction. ACM Operating Systems Review, 1995,29(5):237～250

3 Hutchinson N C, Perterson L L. The *x*-kernel: an architecture for implementing network protocols. IEEE Transactions on Software Engineering, 1991,17(1):64～75

4 Schmidt D C, Suda Tatsuya. Transport system architecture services for high-performance communications systems. IEEE Journal on Selected Areas in Communications, 1993,11(4):489～505

5 Quarterman J S, Silberschatz Abraham, Peterson J L. 4.2BSD and 4.3BSD as examples of the UNIX system. ACM Computing Surveys, 1985,17(4):379～418

Implementation of Interface for Microkernel Based Network System

HU Ning-ning JIN Zhi-quan XIE Li

(*Department of Computer Science and Technology Nanjing University Nanjing* 210093)
(*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)

Abstract CONET V2.0 is a network system under microkernel environment. Compared with the traditional monolithic kernel, microkernel provides a different approach to construct operating system, which leads to different implementation of those modules running under such environment. In this paper, the architecture and design of CONET V2.0 are outlined, and then the design and the implementation of SOCKET, which is one of the application interfaces for the network system, are presented in detail.

Key words Network system, application interface, socket, microkernel.