文章编号:1007-757X(2006)06-0060-02

TCP/IP 协议栈在Limux 与FreeBSD 中的实现分析

胡　影，蒋年德

摘　要:本文分析了Linux 和freeBSD 两个操作系统的部分网络实现代码,并通过研究一个完整的1/0 操作流程,遍历操作系统中TCP/IP 协议栈的内核实现,最后对相关实现作出了相应的评价。

关键词:Linux;FreeBSD;TCP/IP 协议栈

中图分类号:TP316　　文献标识码:A

1　引言

在众多的开源操作系统种,LINUX 以开放性和众多的驱动支持著称,而FreeBSD 有着优良的UNIX 传统,是公认的最稳定的操作系统。对网管来说如何进行选择,本文拟从协议栈的实现中寻找答案。

2　Linux、FreeBSD 和TCP 协议组简介

2.1　Linux 与Freebsd 简介

Linux 作为一种开放源代码的操作系统,自1991 年问世以来,不管是从功能上,还是从流传的广度上,都得到了稳步的增长。Linux 系统包含了建立Internet 网络环境所有服务的软件包,如 Apache Web 服务器、MAIL 服务器、firesall、Database 服务器等。因此研究Linux 操作系统下的TCP/IP 协议栈的实现具有重要意义。

LinuxBSD 是一个稳定的、开放源码的、完全32 位的操作系统,它是基于X86 平台上的一种UNX,FreeBSD 以其良好的高负荷稳定性,被众多ISP(Internet 服务提供商)、ICP 选作各种Server 的os,它可以直接用于教学和商务服务,能方便的进行二次开发。目前,FreeBSD 已被研究中心或大学用来研发出了一些高水平的应用程序,其应用范围已逐步推广。

2.2　TCP/IP 协议组简介

TCP/IP(传输控制协议/网间协议)是一种网络通信协议,它规范了网络上的所有通信设备,尤其是一个主机与另一个主机之间的数据往来格式以及传送方式。TCP/IP 是INTERNET 的基础协议,也是一种电脑数据打包和寻址的标准方法。大数据传送中,可以形象地理解为有两个信封,TCP 和IP 就像是信封,要传递的信息被划分成若干段,每一段塞入一个TCP 信封,并在该信封而上记录有分段号的信息,再将TCP 信封塞入IP 大信封,发送上网。在接受端,一个TCP 软件包收集信封,抽出数据,按发送前的顺序还原,并加以校验,若发现差错,TCP 将会要求重发。因此,TCP/IP 在INTERNET 中几乎可以无差错地传送数据。

3　TCP/IP 协议的实现

从进程的角度上讲,可以调出send,sendto 来发送一段数据,也可以使用文件系统中的write 和writev. 同理,接收数据可以使用相应的recv,recvfrom 这样的接口,或者使用文件系统提供的read,readv. 对于接收来说,是异步进行的,也就是说,是中断驱动的,在以后的分析中,我们要注意这点。为简单起见,同时又不失一般性,我们将分析TCP 协议的输入输出全过程,并对Linux 及FreeBSD 的实作一对比。

在实现上,FreeBSD 与最初的实现一脉相承,而Linux 的实现自成体系,仅与传统实现保持接口上的兼容,我们将针对源码级的实现,来分析一下两者的异同。

3.1　TCP 协议栈在FreeBSD 上的实现

首先介绍FreeBSD 上的协议实现,这也是最正统的实现。下图是完整的输入输出路径。

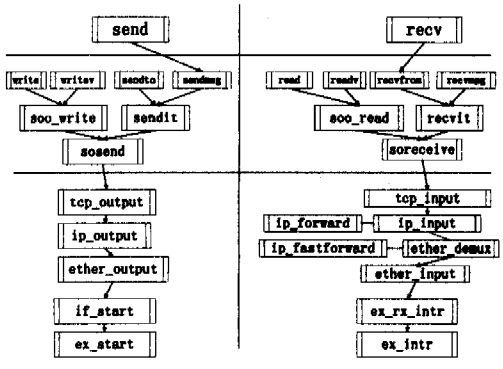

图1　TCP 协议栈在FreeBSD 上的实现

在图1 的左边是TCP/IP 的输出,不管应用程序调用哪个输出函数,最终都要调用sosend 来完成输出。Sosend 将从用户空间把数据复制进内核管理的m-buf 数据结构中(m-buf

作者简介:胡　影,江西抚州东华理工学院计算机与通信系,抚州　344000
蒋年德,江西抚州东华理工学院计算机与通信系,抚州　344000

是FreeBSD的TCP实现使用的数据缓冲结构)。在sosend完成数据复制后,将调用TCP的输出函数tcp－output,它要做的事情是分配一个新的m－buf确良不保存tcp头,并计算相应的数据校验码,在下一步的ip－output中,同样也要进行数据校验工作,并进行数据路由选择。最后ether－output并通过if－start来调用具体的硬件驱动程序来完成数据发送。在某 个网卡的驱动中,ex－start负责将数据从内核的m－buf缓冲复制进硬件自己的缓冲区,以完成数据发送工作。在这整个过程中,数据被复制两次,同时也被编历两次(计算校验码),这是主要的影响效率的地方。

图1右边是TCP/IP的输人,当网卡收到数据时,中断处理程序ex－intr将数据从硬 件缓冲复制进m－buf数据结构中,并调用ether－input来进一步处理。ether－input通过ether－demux进行分用。如果是一个IP包,将通过软中断调用ip－fastforward进行数据校验,并判断是否要转发,如果失败,用ip－input进行完整的处理。在tcp－input中,进行数据校验和验证后,有一个叫做首部预测算法的优化,可以加快数据处理速度。所有的操作完成后,如果是用户数据,将唤醒用户进程进行处理。同理,用户可以使用多个函数进行数据接收,soreceive负责将数据从m－buf转移至用户进程缓冲。

由以上的分析可以得出,在FerrBSD中,发送和接收数据,所进行的操作差不多,都要进行两次数据复制和两次数据遍历。

3.2 TCP协议栈在Linux上的实现

Linux开发之初并没有网络功能,因此在内核在内核对于网络的支持上与FreeBSD相比仍然有一定的差距。图2是TCP协议栈在Linux上的实现。

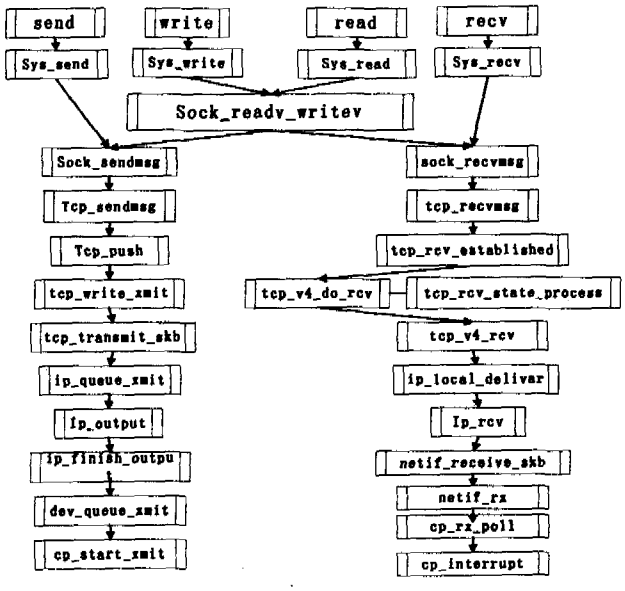

图2　TCP协议栈在Linux上的实现

TCP协议在LINUX上的实现稍显复杂。首先从发送数据开始分析。在LINUX上,socket被实现为一个文件系统,这样可以通过vfs的write来调用,也可以直接使用send来调用,它们最终都是调用sock－snedmsg. sock－sendmsg通过它的内核版本—sock－sendmsg直接调用tcp－sendmsg来发送数据。在tcp－sendmsg中,同时完成数据复制和数据校验,节省了一次遍历操作,这是和FreeBSD不同的地方。Linux使用skb结构来管理数据缓冲,这与FreeBSD的m－buf大同小异。当复制完数据后,使用tcp－push来进行发送。Tcp－push通过—tcp—push－pending－frames调用tcp－write－xmit将数据填人tcp的发送缓冲区。这里的填充仅是指针引用而已。下一步,tcp－transmit－skb将数据放人ip的发送队列。Ip－queue－xmit函数完成IP包头的设置以及数据效验,并调用ip－output进人下一步发送。如果不用分片,将使用ip－finish－output继续发送。在这里,将检查硬件头部描述符(这里就是以太网包头)并填充人数据缓冲区,而后调用dev－queue－xmit函数来进一步处理。Dev－queue－xmit函数将数据排队放人硬件缓冲区以等待随后的发送。而使用具体的网卡驱动动程序cp－start－xmit来完成数据的最终发送。最后的cp－start －xmit做的事情和freebsd的相应函数差不多,检查数据,并将其复制进硬件 级冲。

当接收到一个数据包的时候,网卡会产生中断,这样网卡驱动的cp－interrupt会被调用。Cp－interrupt做的事情很省,只进行必要的检查后就返回了,更多的事情通过cp－rx－poll来完成,这是因为linux驱动分为上半部和下半部,下半部的cp－rx－poll函数在软中断中被调用,这样做是为了提高驱动的处理效率。Cp－rx－poll做的事情主要就是把申请并将数据复制进一个skb缓冲中。Netif－rx函数将数据从这个队列中转移至网络核心层队列中,netif－receive－skb从这里接收数据,并调用ip－rcv来处理。Ip－rcv和ip－vcv－finish —起检查数据包,得到包的路由,并调用相应的input函数来完成路由,在这里就是ip－rcv－finish —起检查数据包,得到包的路由,并调用相应的input函数来完成路由,在这里就是ip－local－deliver,ip－local－deliver完成IP包的重组,而后将使用ip－local－deliver－sinish来进人TCP的处理流程,tcp－v4－do－rcv先判断是否正常的用户数据,如果是则用tcp－rcv－cstablishde处理,否则用tcp－rcv－state－process来更新本条TCP连接的状态机。在tcp－rcv－cstablished中同样实现有首部预测。如果一切顺利,将唤醒等待在tcp－recvmsg中的用户进程。后者将数据从skb缓冲中复制进用户进程缓冲。

4　结论

通过以上分析得出:

(1)Linux的代码比较混乱,可读性没有FreeBSD的好,例如,Linux省略了以太网层,而且在接收数据中有多次异步操作,也许这将会影响内核的稳定性。FreeBSD的代码就比较清晰,程序处理一目了然,可读性也高,最稳定的操作系统名不虚传。这也可以从两个操作系统的起源得到解释。Linux起源于互联网时代,由众多爱好者一起完成,并没有一个完整的

(下转第64页)

tomers— Third
UNION ALL
SELECT * FROM;
Server — First. CompanyDatabase. TableOwner. Customers— First
UNION ALL
SELECT * FROM;
Server — Second. CompanyDatabase. TableOwner. Customers—Second

3 如何才能对分布式分区视图进行更新操作

若一个分布式分区视图进行更新操作需要注意下列问题:

3.1 使用UNION ALL 来合并各个查询结果。

每一个SELECT 命令语句仅引用一个SQL Sever 数据表。该数据表可以是一个本地数据表或通过OPENROWSET()函数或OPENDATASOURCE()函数取得的远程数据表。

每一个成员数据表在视图的SELECT 命令语句的FROM参数被引用的次数不能超过一次以上;成员数据表不能有任何索引是根据运算字段所创建的;每一个成员数据表都必须在相同的字段应用PRIMARY KEY 约束。

3.2 视图的SELECT 命令语句的选择串行中的每一个字段都必须遵循下列规定:

每一个成员数据表的所有字段都必须包含于选择串行中;相同的字段不能在选择串行中使用一次以上;每一个字段的顺序都必须相同,而且相对应的数据类型也必须相同。

3.3 分区字段(即CHECK 约束所应用的字段)必须遵循下列规定:

每一具成员数据表的分区字段所定义的CHECK 条件约束的数据范围不能重叠;而CHECK 条件约束只能使用下列运算符:"BETWEEN"、"AND"、"OR"、"<"、"<="、">"、">="、"="。

3.4 分区字段不允许NULL 值:

分区字段必须是数据表的主索引键的一部分;只能应用一个CHECK 条件约束到分区字段。

3.5 使用INSERT 命令通过分区视图将数据添加到各个成员数据表时,应遵循以下的规则:

所有的字段都必须包含在INSERT 命令中,即使某一个字段在来源数据表中允许接受NULL 值或是已在来源数据表中已定义了一个默认值;不能在INSERT 命令的VALUES 参数中使用DEFAULT 关键字;INSERT 命令所添加的数据至少必须符合其中一个成员数据表所定义的CHECK 条件约束;如果成员数据表中包含自动编号字段或timestamp 字段,将不允许运行添加操作;若使用了自连接,也将不允许运行添加操作。

3.6 使用UPDATE 命令通过分区视图对各个成员数据表进行更新时,应遵循以下的规则:

不能在UPDATE 命令的SET 参数中使用DEFAULT 关键字来更新字段的数据;不能编辑自动编号的字段;如果成员数据表中包含timestamp 字段,将不允许运行编辑操作;若使用自连接,也将不允许运行编辑操作。

3.7 使用DELETE 命令通过分区视图删除各成员表中的数据时,需注意:若使用自连接,将不允许运行删除操作。

以上简述的一些技术要点,对合理创建和使用SQL Server 的分布式分区视图会有所帮助。

参考文献:

[1]Microsoft SQL Server 2000 数据库编程[M]. [美]微软公司.
[2]方盈编著,SQL Server 2000 中文版彻底研究[M]. 中国铁道出版社
[3] http://www. microsoft. com/china/sql/evaluation/features/distpart. asp
[4]"深入介绍sql server 2000 的新特性[EB/OL]. ——www. ithome—cn. net/technology/data/data050. htm
[5]在 Microsoft SQL Server 2000 数据仓库中使用分区[EB/OL]—www. csdn. com. cn/database/1231. htm

(收稿日期:2005—4—22)

(上接第61页)

规划,代码也多次经过变动,而作者水平也参次不齐,造成现在的样子。而FreeBSD 系出名门,一直由一个独立的小组进行维护,多年来更新不大,只有少许优化,所以代码的可读性非常高。但从另一方面讲,不断更新的Linuxd 在代码方面比较激进,比如Linux 使用skb 缓冲效率要较FreeBSD 使用的m—buf 为高,而且linux 发送数据时,复制数据的同时完成TCP 的效验,节省了一次数据的遍历操作,也提高了效率。

(2)从效率上讲,Linux 略占优势,而如果从稳定性上考虑,FreeBSD 应该更胜一酬。不过,这两个操作系统都是非常优秀并久经考验,它们之间的差别也许仅存于纸面分析上。比如说,它们都采用数据缓冲区管理,以避免额外的数据复制操作,也都采用针对体系结构的数据校验算法,以提高数据数据校验的效率。而这,也正是协议栈实现中最大的瓶颈所在。所以,无论选择哪种系统做为服务器,这给养都不会让人失望的,而要是希望学习协议栈的实现,则可以考虑从FreeBSD 来入手。

参考文献:

[1]W. Richard Stevens. TCP/IP Illustrated, Volume 1: The protocols. [M]机械工业出版社. 2004
[2]Gary F. Wright &W. Richard Steven. TCP/IP Illustrated, Volume 2: The Implentation. [M]机械工业出版社. 2002
[3] Alessandro Rubini, Jonathan Corbet. Linux Device Drivers. [M]中国电力出版社. 2004
[4]RFC791(rfc791)—INTERNET PROTOCOL[Z]
[5]RFC1180(rfc1180)—TCP/IP tutorial[z]
[6]RFC793(rfc793)—Transmission Control protocol[z]

(收稿日期:2006—2—13)