文章编号:1001 - 9081(2010)04 - 0895 - 03

基于 VxWorks 的 KAME 协议栈 Socket 描述符的研究与扩展

张焱焱[1],冉祥金[2]
(1. 解放军理工大学 指挥自动化学院,南京 210007; 2. 中兴通讯股份有限公司 南京研究所,南京 210012)
(zhangyanyanyy@163.com)

摘 要:通过对 KAME 协议栈的 Socket 机制进行研究,提出了一种扩展 KAME 协议栈的方法,修改了 KAME 协议栈中 Socket 描述符的实现机制。这种方法有效地按照要求扩充了 IPv6 Socket 的个数,在高性能的嵌入式系统上提高了 IPv6 的运行效率,在 VxWorks 操作系统上运行可靠,并可作为 KAME 协议栈的扩展应用于 IPv6 网络。

关键词:IPv6;KAME 协议栈;Socket;VxWorks 操作系统;嵌入式系统
中图分类号: TP316.2 **文献标志码**:A

Research and extension of Socket in KAME stack based on VxWorks

ZHANG Yan-yan[1], RAN Xiang-jin[2]
(1. *Institute of Command Automation, PLA University of Science and Technology, Nanjing Jiangsu* 210007 *China*;
2. *Institute of Nanjing, Zhongxing Telecommunication Equipment, Nanjing Jiangsu* 210012, *China*)

Abstract: This paper researched the Socket mechanism of KAME protocol stack, and proposed an approach to extend the KAME stack, which modified the implementation mechanism of Socket. It increases the number of IPv6 Socket effectively, and improves the efficiency of IPv6 in high performance embedded system, and it is reliable to run in VxWorks operating system, and it can also be used in IPv6 network as extension of KAME stack.

Key words: Internet Protocol version 6 (IPv6); KAME protocol stack; Socket; VxWorks operating system; embedded system

0 引言

随着网络的飞速发展,目前的 IPv4 协议在许多方面已经显得不太适应,如 IPv4 地址严重不足,缺乏服务质量(QoS)控制,安全性差等。因此,为了满足人们对地址空间、性能以及安全性等方面的新需求,Internet 工程任务工作小组(IETF)的 IPng 工作组确定了 IPng 的协议规范,并称之为“IPv6”[1]。IPv6 是为适应未来对于网络基础设施的数量和质量的需求而设计的下一代互联网协议,解决了地址不足的问题,更好地实现了端到端的通信,能够更容易地实现 QoS 控制[2]和网络安全[3]。

KAME 协议栈是日本一个因特网社团——WIDE 项目组研究并实现的一个 IPv6 协议栈,它是一个开源的项目[4],最初实现在 BSD 系统上,目前已经在越来越多的系统上(包括各种 BSD 系统的变种:BSD/OS、FreeBSD、NetBSD 和 OpenBSD 等以及 VxWorks[5]等实时嵌入式操作系统)得到了应用。

与 IPv4 协议栈类似,KAME 协议栈同样实现了 Socket 套接字,应用层通过 Socket 描述符来与协议栈进行交互[6-8]。在一些路由器或网关设备上,随着 IPv6 的应用越来越多,IPv6 的 Socket 个数明显不足,因此研究一种新的 IPv6 Socket 实现方案,具有较高的实用价值。

1 KAME 协议栈的 Socket 实现

1.1 实现原理

KAME 协议栈以文件描述符的方式实现了 Socket 描述符。具体的实现原理如图 1 所示。从图 1 可以看出 KAME 协议栈与操作系统内核之间的紧密关系,其中虚线所表示的是 KAME 协议栈与操作系统内核之间 Socket 描述符和 Socket 控制块的查找和保存操作。

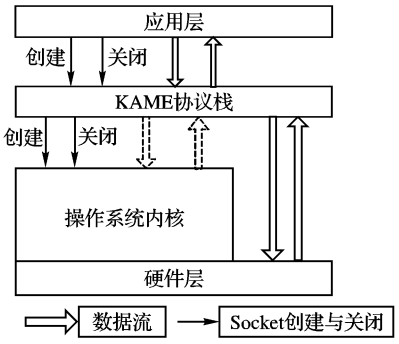

图 1 KAME 协议栈中的 Socket 实现

在图 1 中,KAME 协议栈实现了如下的操作。

1)创建 Socket 描述符(Socket 系统调用)。从操作系统内核中获取一个 Socket 套接字,然后动态申请一个 struct atd_socket 结构体变量,用于保存与网络连接相关的成员。如果内存申请成功,就把变量指针保存在 Socket 套接字所指向的内核资源中。

2)关闭 Socket 描述符(close 系统调用)。将 Socket 描述符传入内核,由内核执行 KAME 协议栈的关闭函数——atd_soo_close 函数,对 Socket 所指向的连接进行关闭操作,然后调用 VxWorks 中的文件描述符释放函数,释放该 Socket 描述符资源。

3)使用 Write 操作通过 Socket 描述符发送数据(Write 系统调用)。将 Socket 描述符传入内核,由内核执行 KAME 协议栈的发送数据函数——atd_soo_write 函数,进行数据的发送。

4)使用 Read 操作通过 Socket 描述符接收数据(Read 系

收稿日期:2009 - 10 - 09;**修回日期**:2009 - 12 - 30。

作者简介:张焱焱(1982 -),女,吉林四平人,助教,硕士,主要研究方向:网络与控制、网络应用; 冉祥金(1982 -),男,山东泰安人,高级工程师,硕士,主要研究方向:TCP/IP 协议栈、嵌入式系统。

统调用)。与 Write 操作相似,将 Socket 描述符传入内核,由内核执行 KAME 协议栈的接收函数 atd_soo_read,接收数据。

在上面所描述的操作中,atd_soo_close、atd_soo_read、atd_soo_wirte 等这些 KAME 协议栈提供的函数是在系统初始化时,通过调用 iosDrvInstall 函数注册在内核中的。

1.2 存在的缺陷

这种 Socket 的实现机制受文件描述符个数的限制,适用于 Socket 比较少的系统,不适用于需要较多连接、内存有限的(如嵌入式服务器)系统。

嵌入式系统所提供的文件描述符比较少(一般提供 512 个左右),虽然可以通过修改配置文件来扩大描述符的个数,但是由于获取描述符是通过循环来查找空闲资源,其效率在资源数比较多时下降很快。另外,对于嵌入式系统来说,将文件描述符扩展为几万个,会给系统带来安全隐患。

2 IPv6 Socket 描述符的扩展

本方案摒弃了 KAME 协议栈的 Socket 实现机制,创建了新的 IPv6 Socket 资源池,并以链表的方式来管理资源:空闲链表和忙链表。具体的资源管理方式如图 2 所示。

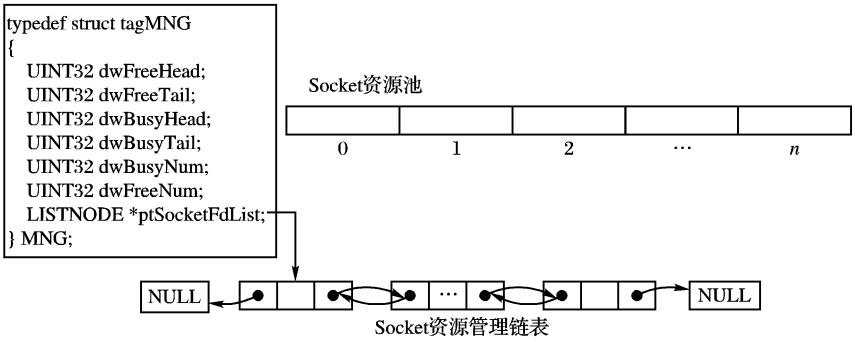

图 2 扩展后的 IPv6 Socket 资源的管理方式

2.1 Socket 描述符资源池的初始化

Socket 描述符对于使用者来说是一个整型数字,但是对于协议栈来说,它是协议栈中资源的索引,协议栈通过这个描述符来找到相应的资源。为了使 Socket 套接字与 struct atd_socket 该结构体关联起来, Socket 描述符资源定义如下:

```
typedef struct tagIPV6_SOCKET_FD
{
 int so_fd;
 struct atd_socket * ptso;
} IPV6_SOCKET_FD;
```

so_fd 是一个 Socket 描述符,在申请一个套接字资源时,把该值返回给调用者。

Socket 描述符资源池定义为:

```
IPV6_SOCKET_FD * g_ptFdPool;
```

g_ptFdPool 作为 IPV6_SOCKET_FD 类型的全局变量指针,在系统初始化时申请指定容量的内存,形成一个套接字资源池,并且定义链表来管理 Socket 描述符的申请和释放。具体定义的链表结构体如下:

```
typedef struct tagLISTNODE
{
 UINT32 dwPrev;
 UINT32 dwNext;
} LISTNODE;
```

该链表定义为一个双向链表,在资源的申请和释放过程中,形成忙链表和闲链表,通过忙链表可以对当前 Socket 资源的使用情况进行查看。另外定义一个全局变量用于保存链表的头和尾,其结构如下:

```
typedef struct tagMNG
{
 UINT32 dwFreeHead;
 UINT32 dwFreeTail;
 UINT32 dwBusyHead;
 UINT32 dwBusyTail;
 UINT32 dwBusyNum;
 UINT32 dwFreeNum;
 LISTNODE * ptSocketFdList;
} MNG;
```

使用该 MNG 结构体定义一个全局变量 g_tIpv6ListMng,在系统初始化时,对该变量的成员进行初始化,并申请 LISTNODE 类型的内存,保存在 ptSocketFdList 成员中,然后将链表初始化形成一个空闲链表,如图 2 所示。

2.2 Socket 描述符的申请

在申请 Socket 描述符时,通过 MNG 结构体中的 dwFreeHead 成员从 ptSocketFdList 链表池中得到一个空闲资源,将其从空闲链表中摘除,然后添加到忙链表中,并修改 dwBusyTail 的值。在申请到描述符并得到 atd_socket 结构变量后,把 atd_socket 变量保存在 Socket 描述符所对应的 g_ptFdPool 资源池中,以备后续操作使用。具体实现流程如图 3 所示。

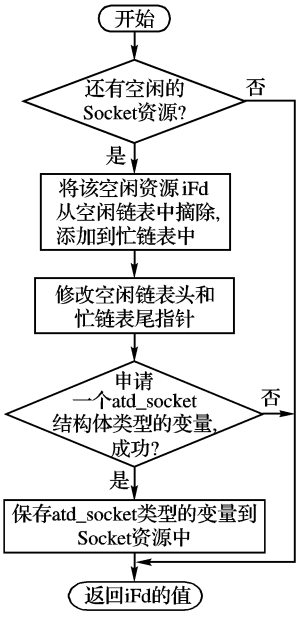

图 3 申请 Socket 描述符流程

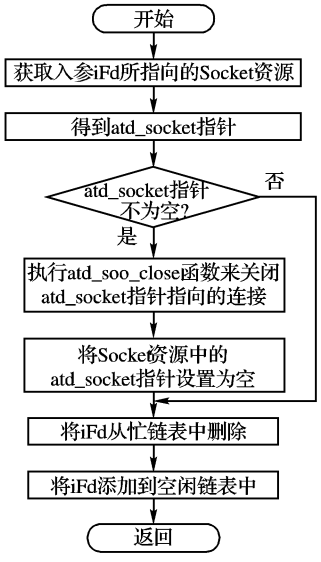

图 4 关闭 Socket 流程

2.3 关闭 Socket 描述符

关闭操作把该描述符对应的连接关闭,并释放连接相关的信息资源。根据 Socket 描述符调用 IPv6_Fd_GetSock 函数

来获取到 atd_socket 结构体,接着调用关闭连接函数 atd_soo_close 来释放那些与连接信息相关的资源,然后再将 Socket 描述符从忙链表中删除,添加到空闲链表中。具体实现流程如图 4 所示。

2.4　读取数据

在有数据到达时,IPv6 协议栈已经成功地接收了数据,并保存在 atd_socket 结构体中。在上层应用调用 IPv6_Read 函数读取数据时,通过参数 iFd 给出了 Socket 描述符的值,找到协议栈中保存的与连接信息相关的资源,然后调用读取数据的函数 atd_soo_read,把目的缓冲区指针和缓冲区长度作为入参传入,由 KAME 协议栈把数据内容取出,保存在缓冲区中。具体实现流程如图 5 所示。

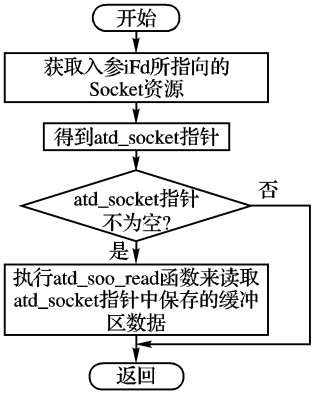

图 5　读取数据流程

2.5　发送数据

发送数据时,把源缓冲区中的内容和长度作为入参传入,通过 Socket 描述符的值调用 IPv6_Fd_GetSock 函数得到 atd_socket 结构体,然后调用 atd_soo_write 函数将数据写入底层驱动,发送出去。具体实现流程与图 5 相似。

3　实验与结果分析

3.1　实验环境及方法

实验采用的配置为 MIPS CPU 400 MHz,512 MB 内存,VxWorks 操作系统,采用 C 语言实现。为了测试当前支持的 Socket 个数,在系统中编写了一个测试函数,该函数通过循环来申请最大个数的 Socket,直到申请不到为止。申请完成后,使用每个申请到的 IPv6 Socket 发送一个 UDP 报文给服务器端,服务器端在收到 UDP 报文后,回复一个响应;客户端收到响应后,认为该 Socket 有效,如果长时间收不到响应(此处设置为 2 min),则认为该 Socket 无效,然后再测试下一个 Socket,直到测试完所有的 Socket。测试结束后,给出测试结果。在测试代码中,分别就申请 Socket、关闭 Socket、发送数据、接收数据的耗时进行了测量。

实验中服务器端采用 Windows XP 操作系统,安装 IPv6 协议栈后,使用 VC 6.0 编写一个 UDP 监听程序,在收到 UDP 报文后,给发送者回复一个响应。

3.2　实验结果

实验针对 KAME 协议栈原有的 Socket 个数及扩展后的 Socket 个数进行测试,给出两种方法在 Socket 个数、资源使用效率方面的实验结果。实验结果如图 6、7 所示。从图 6、7 可以看出,扩展后的 Socket 实现机制大大增加了 IPv6 Socket 的个数,并且在数据收发效率方面好于原有的 Socket 机制;原有的 Socket 机制随着 Socket 个数增加,对 Socket 进行操作的耗时也随之增加。但是修改后的 Socket 机制不仅扩大了 Socket 的个数,而且在进行 Socket 操作时,不受 Socket 个数的限制,耗时比较均匀,适合于嵌入式系统的应用。

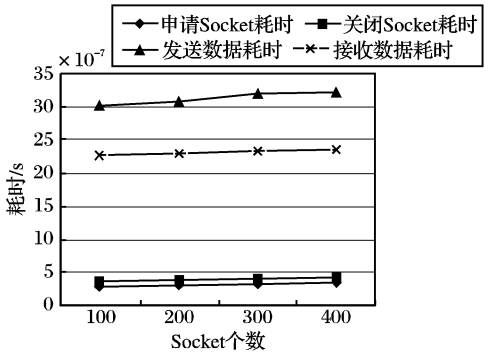

图 6　KAME 协议栈原有的 Socket 机制测试结果

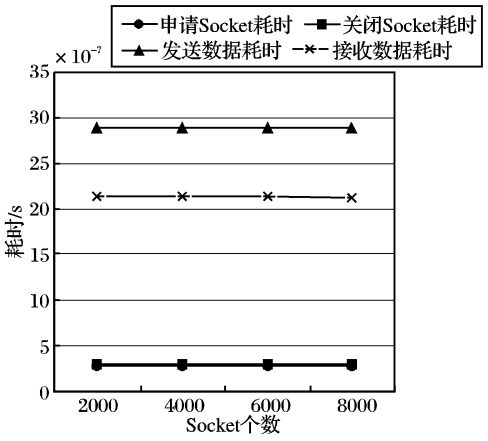

图 7　扩展后的 Socket 机制测试结果

4　结语

改造后的 IPv6 Socket 描述符的实现可以按照所需要的容量进行配置,不再受文件描述符个数的限制,而且申请和释放 Socket 套接字时,避免了循环查找的操作,满足嵌入式系统高效率的要求,在宽带网关设备上运行稳定。

参考文献:

[1] 张云勇, 刘韵洁, 张智江. 基于 IPv6 的下一代互联网[M]. 北京: 电子工业出版社, 2004.

[2] 张云勇, 张智江, 刘韵洁, 等. IPv6 业务技术研究[J]. 计算机科学, 2005, 32(1): 8 - 12.

[3] 李信满, 赵宏. IPv6 的安全体系结构[J]. 中兴通讯技术, 2002, 8(3): 4 - 7.

[4] LI QING, JINMEI T, SHIMA K. IPv6 core protocols implementation[M]. San Fransisco: Morgan Kaufmann Publishers, 2006: 10 - 12.

[5] 孔祥营, 柏桂枝. 嵌入式实时操作系统 VxWorks 及其开发环境 Tornado[M]. 北京: 中国电力出版社, 2002.

[6] 刘利强, 吴永英, 王勇智. IPv6 下 Socket 网络编程的研究与实现[J]. 计算机技术与发展, 2006, 16(6): 201 - 203.

[7] 夏涛, 余胜生, 周敬利, 等. 开发支持 IPv6 的应用程序[J]. 计算机工程, 2007, 27(10): 51 - 53.

[8] 袁德明. 用 IPv6 编程接口实现有连接通信的方法[J]. 计算机时代, 2007(9): 23.