文 章 编 号: 1000-1220(2002) 01-0018-07

高性能路由器操作系统 HEROS的设计与实现

徐 恪 喻中超 吴建平 徐明伟

(清华大学 计算机科学与技术系 北京 100084)

摘 要: 实时分布式操作系统是高性能分布式路由器的控制核心,为了保证路由器系统的整体性能和安全性,本文设计并实现了实时分布式操作系统 HEROS(Highly Efficient Router Operating System). HEROS基于微内核体系结构,其多任务内核实现了基于优先级的抢先式调度,高效率的任务间同步和通信原语,实时的中断处理和高效的内存管理机制.为了更好地服务于分布式路由器体系结构,HEROS基于 CompactPCI总线实现了一种分布式通信机制和面向网络协议的高性能的缓冲管理机制.目前,基于 HEROS的高性能安全路由器原型系统已经设计完成.

关 键 词: 实时; 分布式操作系统; 路由器

中图分类号: TP393 文献标识码: A

Internet的飞速发展,对 Internet互连的核心设备-路由器的性能提出了越来越高的要求.目前,路由器的体系结构已经从单处理机结构转向了分布式多处理的结构[1,2],而实时分布式操作系统是路由器的控制核心,它的稳定性和可靠性直接关系到路由器系统的整体性能和安全性.

为了能够完全掌握路由器核心技术,并真正保证系统的安全性,清华大学在承担"863"重点攻关项目-"高性能安全路由器"时,实现了实时分布式操作系统 HEROS(Highly Efficient Router Operating System). HEROS基于微内核体系结构,其多任务内核实现了基于优先级的抢先式调度,高效率的任务间同步和通信原语,实时的中断处理和高效的内存管理机制.为了更好地服务于分布式路由器体系结构,HEROS基于 CompactPCI总线实现了一种分布式通信机制和面向网络协议的高性能的缓冲管理机制.

1 HEROS的总体结构

HEROS的总体结构如图 1所示. HEROS是一种松散耦合的多处理机体系结构,整个系统由多个具有自治能力的节点组成,每个节点都有自己的 CPU,内存和输入输出设备(包括网络接口设备),每个节点上都运行单处理机的多任务 HEROS内核,该内核实现了常用实时操作系统的主要功能.在各个 HEROS内核之上是一种全局的分布式消息通信机制,它可以透明地在各个节点之间传递消息.用户的程序,也就是高性能安全路由器中的 IP协议和上层软件就基于该分布式消息机制构造.下面几节将详细介绍 HEROS的各个组成部分.

2 HEROS中的多任务内核

本节将讨论 HEROS多任务内核中的多任务机制,任务间通信和中断处理,它们是操作系统的核心功能.

2.1 HEROS中的任务调度

多任务操作系统都需要适当的任务调度算法为准备好的任务分配 CPU时间, HEROS支持两种任务调度算法,基于优先级的抢先式调度算法和轮转调度算法.

使用基于优先级的抢先式调度时,每个任务都有一个优先级,内核把 CPU分配给处于就绪状态的任务中的优先级最高的任务.当一个任务运行时,如果有另一个优先级更高的任务处于就绪状态,那么内核将立即保存当前任务的上下文并切换到优先级更高的任务.任务在创建时分配优先级,在执行过程中,任务也可以通过系统调用修改自己的优先级,这种动态优先级特性使系统具有很好的灵活性和适应性.

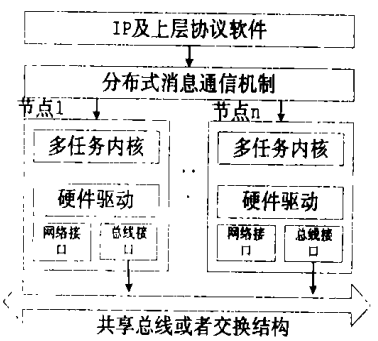

图 1 HEROS的总体结构

上述调度算法只解决了不同优先级之间的任务的调度问题.当系统中同时存在多个相同的最高优先级的任务时,任务调度有两种选择.在缺省状态下,当其中某个任务运行时,如

收稿日期: 2000-10-17 收修改稿日期: 2000-12-22 基金项目: 国家自然科学基金(No. 90104002)资助;国家"八六三"重大项目"高性能安全路由器"(项目编号: 863-306-ZD-07-01)的资助 作者简介: 徐 恪,博士,讲师.研究方向为计算机网络体系结构,实时分布式操作系统.喻中超,硕士研究生,研究方向为实时分布式操作系统.吴建平,博士,教授,博士生导师,研究方向为计算机网络体系结构,计算机网络协议测试,形式化技术.徐明伟,博士,副教授,研究方向为计算机网络体系结构,计算机网络性能测试. E-mail: xuke@csnetl.cs.tsinghua.edu.cn

果它不主动让出 CPU,那么它就会一直运行下去,这会导致其它的任务没有机会执行.另一种选择是在这些相同优先级的任务之间进行轮转调度.轮转调度利用时间片来保证相同优先级的任务得到相同的处理时间.

2.2 HEROS中的任务间通信

HEROS中的任务间通信机制主要包括共享内存,信号量和消息队列.

共享内存　在实时操作系统中最常见的任务间通信方式就是共享内存.由于实时操作系统的各个任务都在同一个地址空间里运行,因此相互之间共享数据结构很容易实现.

信号量　当多个任务共享同一个数据结构时,需要某种同步和互斥机制来保证数据的一致性.信号量就提供了这种机制. HEROS中提供了两种信号量: 二值信号量和计数信号量.二值信号量顾名思义,就是只有两个值,它实际上相当于一个标志位,表示共享资源是否可用.二值信号量实现简单,运行效率高.计数信号量则是在创建时指定初始值,当有任务取信号量时,值减 1,当有任务还回信号量时,值加 1.当信号量减为 0后,试图取该信号量的任务将被阻塞.

在多任务实时操作系统中,由于采用了抢先式的调度算法,常常会造成优先级逆转 (priority inverse)[10],这时优先级高的任务必须等待优先级低的任务完成之后才能执行.为了解决这一问题, HEROS内核中采用了优先级继承算法 (priority inheritance).该算法的基本思想是,当高优先级的任务由于申请信号量而阻塞时,当前正在使用该信号量的低优先级的任务要继承高优先级任务的优先级直到它释放该信号量对应的共享资源为止.

为了避免出现优先级逆转,我们在高优先级的任务因为申请信号量而阻塞时,把正在使用该信号量的低优先级的任务优先级提高到高优先级一样.这样虽然任务执行的顺序是一样的,高优先级的任务仍然需要等待低优先级的任务完成,但是从任务优先级来看没有出现逆转现象.

消息队列　虽然信号量提供了快速的任务间同步和互斥机制,但是在实际的系统中,任务之间除了需要同步和互斥之外,还需要交换信息.消息队列就是 HEROS中主要的任务间的信息交互方式.单节点的 HEROS内核中实现了单节点的消息队列机制.

消息队列是独立于任务的对象,它可以缓存一定量的变长的消息.任何一个任务或者中断服务程序都可以向消息队列发送消息,任何一个任务 (为了保证中断服务程序的效率,我们规定中断服务程序不能接收消息)也都可以从消息队列接收消息.多个任务可以同时使用同一个消息队列.消息队列是一种单向的通信机制,因此,如果两个任务之间需要双向的数据通信,则需要建立两条消息队列.

创建消息队列时,需要指定队列中能缓存的最大消息数和每条消息的最大长度.这样内核可以为该消息队列预先分配足够的空间.

3 HEROS中的分布式消息通信机制

随着对路由器性能要求的提高,路由器的体系结构也从单处理器体系结构发展成多处理器分布式体系结构[1].为了适应分布式处理的需要,我们对 HEROS的单处理器内核进行了扩展,实现了分布式的消息通信机制,目前实现的版本面向使用 CompactPCI总线的 MCP750开发平台[10].

操作系统对分布式处理的支持有两种方式,一种是实现真正的分布式操作系统[3-5],在这种分布式操作系统中,整个系统只有一个单一的映象,用户看起来就好象面对的是一个单处理器的操作系统,这种系统实现复杂,需要实现任务的动态分配和迁移机制,需要实现分布式的文件系统等等,而且进程间通信的性能也可能会受到影响.另一种方式就是在现有的单处理器操作系统内核的基础上进行扩展,使其能部分支持分布式处理[6].这样的优点是开发周期短,系统易于维护,且性能较好.

考虑到在路由器的软件处理过程中[7]并不需要真正的单一系统映象的分布式操作系统的支持,而是只需要多节点之间的分布式通信机制,因此我们选择采用第二种方式实现分布式消息通信.

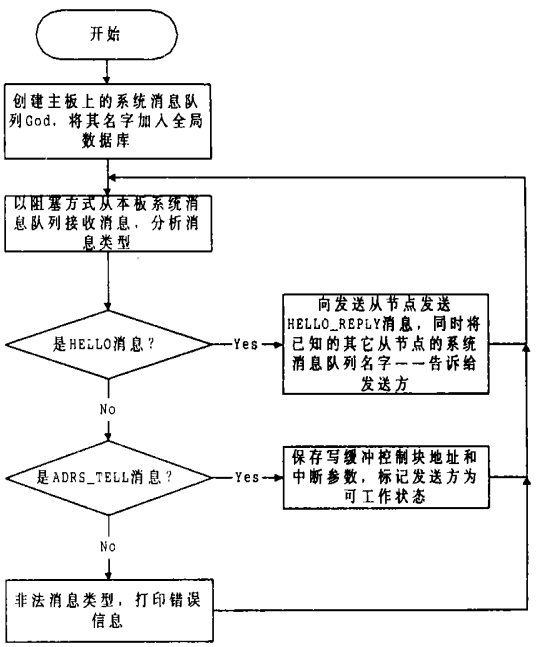

图 2　skServer任务流程图

我们实现的分布式消息通信机制 (以下简称为 SK消息机制)有以下一些特点:

(1) 充分利用了 CompactPCI总线的能力,基于 CompactPCI总线的 MCP750开发系统采用非透明的 PCI到 PCI的桥芯片 21554[11]实现了板间的内存共享,提供了共享内存的读写原语.我们就基于这些原语实现了分布式消息通信机制.

(2) 该消息机制是面向 IP应用的,并不是通用的任务间消息机制,这是为了最大限度地提高 IP转发的性能.

(3) 该消息机制和我们下面将要讨论的缓冲管理机制有机地结合起来,组成了一个整体,共同完成高性能的分组转

发.

3.1 SK消息通信机制的组成

我们设计的 SK消息机制是面向高性能路由器的,高性能路由器采用了分布式的体系结构[2],整个系统由一个主节点和多个从节点组成.主节点负责处理路由协议包;维护全局路由表;保持局部路由表与全局路由表的同步;完成对路由器的操作配置与网络管理功能和维护全局安全数据库.从节点则主要负责根据局部路由表完成分组转发,因此我们分别在主节点和从节点上运行 skServer和 skClient任务用于实现各个节点之间的消息通信.

3.2 SK消息通信机制的初始化

系统启动后,首先要进行 SK消息通信机制的初始化,初始化的主要工作包括根据操作管理子系统的配置信息建立系统消息队列;设置中断处理函数;命名分布式消息队列等等.然后根据节点号不同分别创建 skServer和 skClient任务.

3.3 skServer任务和 skClient任务的主要工作

skServer任务的流程如图 2所示,skClient任务的流程如图 3所示.

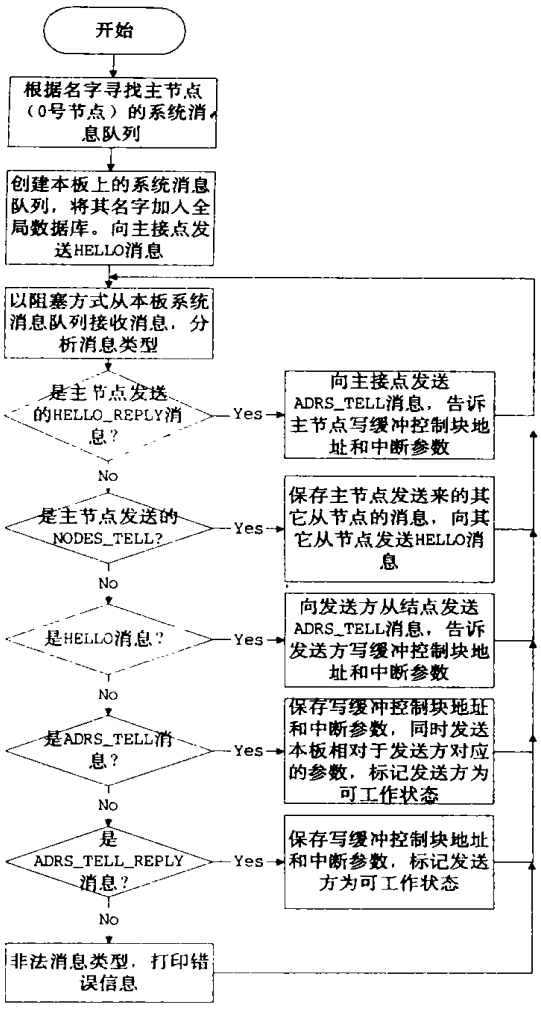

图 3 skClient任务流程图

下面说明一下 skClient和 skServer所完成的工作.

SK消息通信信息交换协议比较简单.每个从节点 i启动之后需要等待主节点创建系统分布式消息队列,然后通过该消息队列向主节点"报到",同时向主节点报告自己的与主节点有关的 SK队列的地址信息.主节点在收到从节点的报到之后,将此前已经向主节点报到的其它从结点的系统队列名字告诉给从节点 i.此后,从节点 i通过系统消息队列与其它从节点 j进行地址信息的交换.

主节点和从节点进行信息交换的协议如图 4所示.各消息包含的信息如下:

HELLO: 包含从节点系统队列名字;

HELLO_ REPLY: 不含额外信息,仅起确认收到作用;

ADRS_ TELL: 包含从节点 SK消息队列相关的地址信息;

NODES_ TELL: 已经向主节点"报到"的从节点的系统消息队列信息.

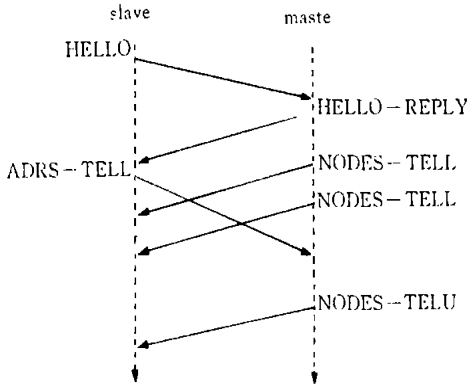

图 4 slave-master SK信息交换协议

此后节点 i与每一个从 NODES_ TELL中得知的其它存在的结点 j进行从节点和从节点间的信息交换. Slave-slave的信息交换的协议如图 5所示.

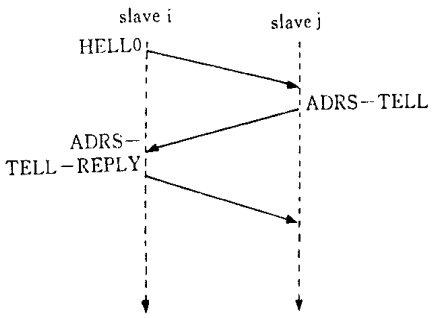

图 5 slave-master SK信息交换协议

在图 5中,各消息包含的内容如下:

HELLO: 同上;

ADRS_ TELL: 同上;只是地址是 j节点上与 i节点相关的地址信息;

ADRS_ TELL_ REPLY: 包含信息同 ADRS_ TELL.

只有收到 ADRS_ TELL或者 ADRS_ TELL_ REPLY, 才认为发送节点的 SK消息模块处于可工作状态 SK_ STATE_ READY.

3.4 sk消息机制的详细设计

sk消息通信机制向上层提供了下列系统调用: STATUS skMsgQLibInit(): SK消息通信模块初始化;

SK_ MSG_ Q_ ID skIpQueueCreate (int32_ t nodeifId, int32_ t options): 创建 SK消息队列;

STATUS skIpQueueDestroy(SK_ MSG_ Q_ ID qId): 销毁 SK消息队列;

SK_ BUFF* skIpReceive(SK_ MSG_ Q_ ID qId, int32_ t flags): 从 SK消息队列中接受缓冲块;

STATUS skIpSend(SK_ MSG_ Q_ ID qId, SK_ BUFF* pSkb, int32_ t flags): 通过 SK消息队列发送缓冲块.

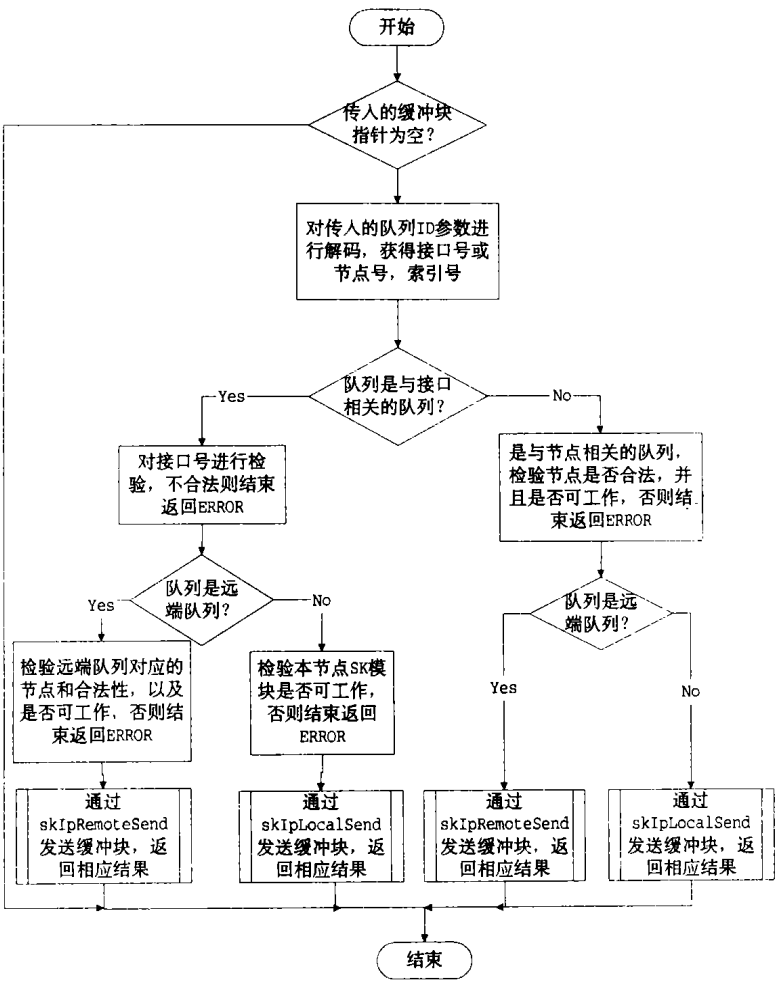

图 6　skIpSend的流程图

限于篇幅,下面只列出 skIpSend的流程图,如图 6所示.

4 面向网络协议的高性能的缓冲管理机制

作为专用的路由器操作系统,HEROS必须对面向协议的缓冲进行有效的管理,影响缓冲管理性能的关键是缓冲内存管理.实践证明,直接通过核心内存分配器 KMA(Kernel Memory Allocator)的 kmalloc接口获得缓冲内存的方式对路由器应用来说是很慢的.因此,必须采取专门的手段管理缓冲内存.

HEROS中的缓冲管理主要实现了下面三个子模块: 缓冲内存管理子模块,缓冲队列操作子模块,缓冲块操作子模块.这些子模块都围绕主要的数据结构 SKBUFF进行操作.

4.1 SKBUFF数据结构

skbuff队列头:

```
typedef struct skBuffHead
{
  struct skBuff  *  pNext;
  struct skBuff  *  pPrev;
  uint32_t   qLen;       /*  length of queue*  /
} SK_BUFF_HEAD;
```

其中 pNext和 pPrev为双向链表指针,qLen为队列长度.

SKBUFF数据结构定义:

```
typedef struct skBuff
{
  struct skBuff  *  pNext;      /*  Next buffer in list*  /
  /*  Previous buffer in list, or message queue Id in skMsgQ Lib
  *  /
  struct skBuff  *  pPrev;
  void           *  pDev;      /*  Pointer to device*  /
  uint16_t       ifNum;        /*  used by message queue*  /
  uint16_t       devNum;       /*  used by message queue*  /
  uint32_t       truesize;     /*  Buffer size*  /
  int32_t        flags;        /*  used for memory managment
                                   or debug*  /
  int32_t        debug;        /*  used for debug*  /
  struct ippkt   *  pIppkt;    /*  Network layer header*  /
  /*  Link layer header*  /
  union
  {
    struct ether_header  *  pEthernet;  /*  h/netinet/if_
                                           ether.h*  /
  char    *  pRaw;
  } mac;
  int32_t     niNum;
  uint32_t    len;            /*  Length of actual data*  /
  char    *    pData;         /*  Data head pointer*  /
  char    *    pTail;         /*  Tail pointer*  /
  char     pHead[1];          /*  variable length*  /
} SK_BUFF;
```

skbuff中一些关键的数据结构之间的关系如图 7所示.关于 skbuff数据结构有如下几点需要说明.

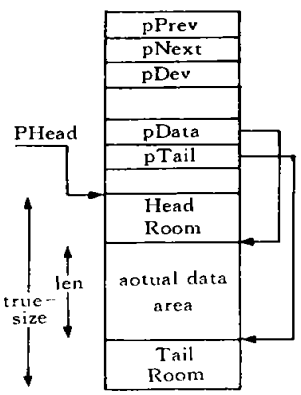

图 7 skbuff结构图

(1) skbuff的数据结构主要借鉴了 Linux操作系统的 skbuff数据结构,但是去掉了许多不必要的控制域;

(2) flags目前可以取的值为 0, SF_ ALIEN 和 SF_ WRITE,用来标识该 skbuff是通过何种分配方式获得的.由于中断中不能进行 malloc和 free操作,因此类型为 SF_ ALIEN的 skbuff暂时没有实现,在下一个版本中可以考虑加上这种类型,因为一般来讲在中断中进行 malloc和 free操作也是不多见的;

(3) 考虑到 skbuff的数量非常多,为了节省 skbuff所占的内存空间,有的控制域是复用的,如 pPrev域.已分配的 skbuff通常是通过双向链表的形式组织的,在这种情况下 pPrev是一个前向指针.但是在不同板间通过消息队列(在后文中,为区别传统的消息队列机制,我们把这种专用于协议数据传送的消息队列称为 SK消息队列,或者 SK队列)传送 skbuff时,写缓冲是通过单向链表组织的,在这种情况下,pPrev用来表示当前的 skbuff对应的 SK队列的 ID,以方便中断处理程序或者轮询任务(SK消息队列同时实现了中断和轮询两种方式,可以通过改变编译选项进行选择)将 skbuff挂入到相应的 SK队列;

(4) 由于历史原因,ifNum和 devNum两个域不再表示接口号和设备号,相应的信息分别由 niNum和 pDev代替.此时,devNum和 ifNum用于表示从写缓冲中分配(称为隐式分配,与之对应的显示分配是指通过 skbAlloc调用分配)的 skbuff对应的节点号和相对于节点号的接口号(称为局部接口号).注意此接口号不是全局接口号,全局接口号是通过对全部接口进行统一编号得到的唯一标识.在进行编号的时候,必须保证同一节点的所有接口的全局接口号是连续的.对特定的节点而言,其第一个接口对应的写缓冲的 skbuff的 ifNum是 1,第二个接口的 skbuff的 ifNum是 2,依此类推.这样只要我们记下特定节点的第一个接口的全局接口号,就可以通过 ifNum实现全局接口号和局部接口号之间的转换.注意局部接口号为 0的不对应任何接口;

(5) skbuff控制域的顺序安排是有一定考虑的.skbuff队列的一些操作要求 pNext和 pPrev两个域一定是放在最前面.另外,出于板间传输效率的考虑,从 pIppkt到 pTail的控制域一定要连续放置,这样可以通过 bcopyLongs进行连续数据的拷贝,测试表明,这种方式比一一跨总线的赋值方式的效率至少提高了 15%.

4.2 缓冲内存管理子模块的设计与实现

该模块主要负责管理缓冲块的分配和释放.需要说明的是系统中的缓冲块并不都是通过该模块分配得到的.还有一种缓冲块是在系统初始化部分为每个接口预分配的写缓冲队列中的.我们称通过缓冲内存管理子模块分配的缓冲块为显式分配的,通过写缓冲得到的缓冲块为隐式分配的.在缓冲块的释放操作中会对缓冲块做如上区分,并相应地做不同的处理.该模块由 4个函数组成:

skbMemInit负责完成缓冲内存管理的初始化工作;

skbCleanup释放缓冲块池,完成与前面函数相反的工作;

skbAlloc显式分配缓冲块;

skbFree释放缓冲块(无论是显式的还是隐式的);

四个函数都是独立的,第一个函数必须在后面三个函数调用之前进行.

图 8列出了 skbAlloc函数的流程图.

4.3 缓冲队列操作子模块的设计与实现

该模块主要负责与缓冲队列相关的一些操作.由于缓冲管理库函数在同一时间可能有多个调用者,因此必须提供互斥保护.本模块对每个函数提供了具有互斥保护和不无互斥保护的两种接口.无互斥保护的接口主要是调用方在保证不会出现两个调用者同时对一个队列进行操作的情况下进行调用的,其开销较小.

4.4 缓冲块操作模块的设计与实现

该模块提供对缓冲块 (skbuff)进行操作的库函数.对缓冲块的基本操作包括 push, pull, put, trim和 reserve等基本操作.这部分比较简单,不再详细介绍.

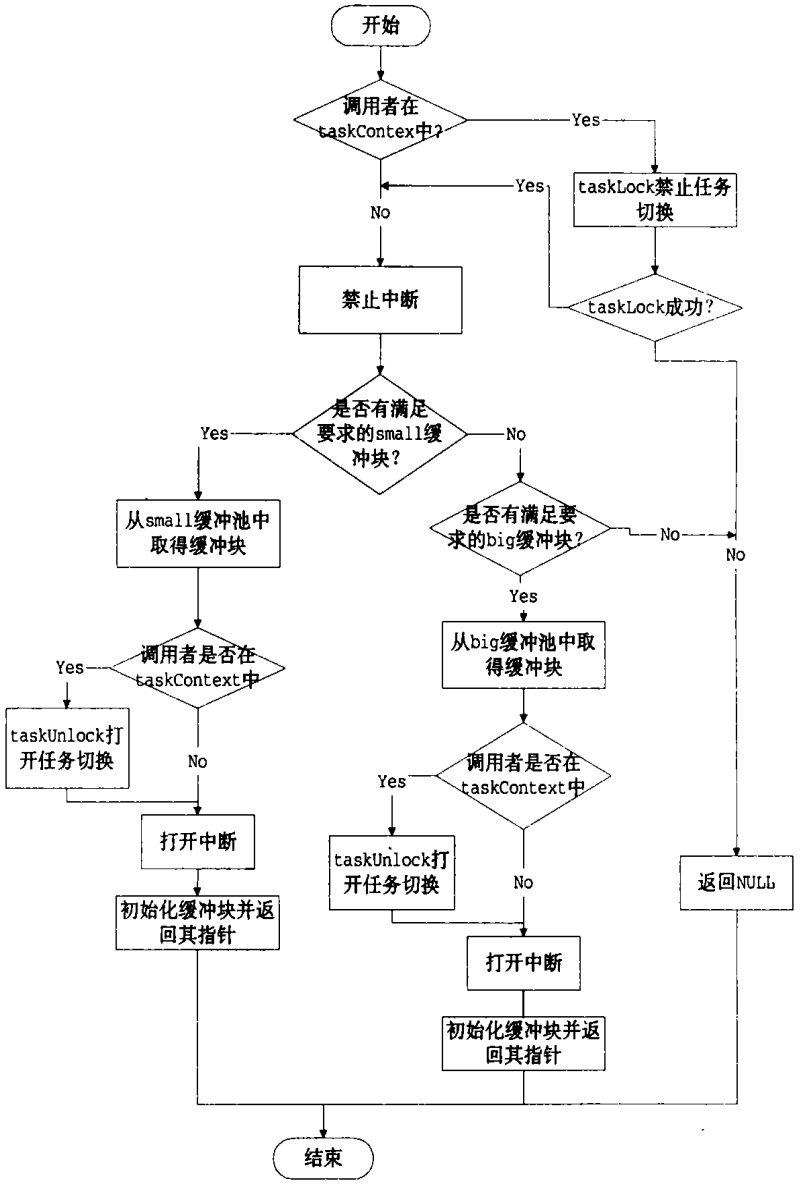

图 8　skbAlloc的流程图

5 结束语

本文详细介绍了面向高性能安全路由器的实时分布式操作系统 HEROS, HEROS基于微内核体系结构,其多任务内核实现了基于优先级的抢先式调度,高效率的任务间同步和通信原语,实时的中断处理和高效的内存管理机制.为了更好地服务于分布式路由器体系结构, HEROS实现了一种基于CompactPCI总线的分布式通信机制和面向网络协议的高性能的缓冲管理机制.目前, HEROS的底层硬件平台是基于CompactPCI总线的 Motorola CPX2208体系结构,处理节点均采用 MCP750单板计算机.基于 HEROS的千兆以太驱动和 SDH 155M驱动已经设计完成,均达到了较高的性能 (千兆以太驱动速度达到了 900M,基本上实现了线速).基本的节点间转发测试证明,基于我们实现的缓冲管理和分布式消息机制实现的 IP转发引擎每秒可以转发 240K个分组 (64字节长),整个 CompactPCI总线的速率已经超过了 1Gbps.这一数据已经超过了 cisco高档路由器 7000系列的转发能力,这也证明我们设计和实现的 HEROS操作系统具有良好的性能.

参　考　文　献

1 Xu Ke, Xiong Yong-qiang, Wu Jian-ping. Analysis of broadband IP router architecture[J]. Journal of Software, 2000, 11(2): 179~186. (in Chinese)
(徐恪, 熊勇强, 吴建平. 宽带 IP路由器体系结构分析[J]. 软件学报, 2000, 11(2): 179~186.)

2 Fan Xiao-bo, Lin Chuang, Wu Jian-ping, et al. Performance model and analysis of distributed router[J]. Chinese Journal of Computer, 1999, 22(11): 1223~1227. (in Chinese)
(范晓勃, 林闯, 吴建平, 等. 分布式路由器的性能模型与分析[J]. 计算机学报, 1999, 22(11): 1223~1227.)

3 Hideyuki Tokuda, Clifford W. Mercer. ARTS: a distributed real-time kernel[J]. Operating Systems Review, 1989, 23(3): 29~53.

4 Paulo Verissimo, Hermann Kopetz. Design of distributed real-time systems[M]. Sape Mullender, Distributed Systems, USA: ACM Press and Addison-Wesley, 1993. 511~530.

5 Ahmed Gheith, Karsten Schwan. Chaosarc-kernel support for multi-weight objects, invocations, and atomicity in real-time applications[J]. ACM Transactions on Computer Systems, 1993, 11(1): 33~72.

6 Ernesto F V Martins, Antonio Nunes da Cruz. An operating system extension for a multiprocessor[J]. Journal of Systems Architecture, 1998, 45(5): 341~361.

7 Rayfield J T, Silverman H F. System and application software for the Armstrong multiprocessor[J]. Computer, 1988, 21(6): 38~52.

8 Tanenbaum A S, Van Renesse Robbert, Van Staveren Hans et al. Experiences with the Amoeba distributed operating system[J]. Communications of the ACM, 1990, 33(12): 46~63.

9 Davari S, Sha L. Source of unbounded priority inversions in real-time systems and a comparative study of possible solutions[J]. Operating Systems Review, 1992, 26(2): 110~120.

10 MCP750 Series Single Board Computer Programmer's Reference Guide[Z]. Motorola Inc., Computer Group, 1999.

11 21554 PCI-to-PCI Bridge for Embedded Applications Hardware Reference Manual[Z]. Intel Inc., 1998.

Design and Implementation of a Highly Efficient Router Operating System

XU Ke YU Zhong-chao WU Jian-ping XU Ming-wei
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract Real-time distributed operating system is the control kernel of high performance distributed router. In this paper, we design and implement a real-time distributed operating system: HEROS (Highly Efficient Router Operating System) which aim is to ensure the total performance and security of router. HEROS based on micro-kernel architecture, its multitasking kernel implement priority-based preemptive scheduling, high efficiency intertask synchronization and communications routines, real-time interrupt services and high efficiency memory management. In order to support distributed router architecture, HEROS implement distributed communication mechanism based on CompactPCI bus and network-oriented high performance buffer management. At present, the high performance security router prototype based on HEROS has been finished.

Key words real-time; distributed operating system; router