**核间通讯控制系统**

**设计说明书**

目录

[1. 系统框架 3](#_Toc477808277)

[2. 实现流程 4](#_Toc477808278)

[3. VirtIO框架 6](#_Toc477808279)

[4. RPMSG设备驱动模块 10](#_Toc477808280)

# 系统框架

多核处理器由数个处理核构成，通常需要在这些核心上运行不同的软件环境，所有处理核心各自执行着自己的代码程序，所以要实现核心间协同工作必然需要涉及核间的数据共享和交换，这就需要有能够实现各个核心之间能够进行高效可靠的通信机制。

该框架设计实现基于软中断和共享内存的核间通讯机制，通过软中断和共享内存区域可有效实现不同处理器核之间数据高效稳定地共享和交换。图1简要介绍了本设计的主要框架结构。该设计中在CPU0上运行Linux操作系统，在CPU1上运行FreeRTOS操作系统。由于在Linux中用户空间程序无法直接访问硬件资源，故需要实现相应的驱动程序设计，该设计中涉及rpmsg用户设备驱动程序以及virtio设备驱动程序的设计；同时为了实现在用户空间通过文件操作的方式完成内存中共享数据的读写，还需要完成字符设备的注册和创建。在CPU1端完成FreeRTOS环境下Rpmsg和virtio设备的创建，并分配共享内存区域，然后告知CPU0设备创建的相关情况，从而与CPU0达成相关协议，建立起两处理器之间基于共享内存的通讯通道建立。

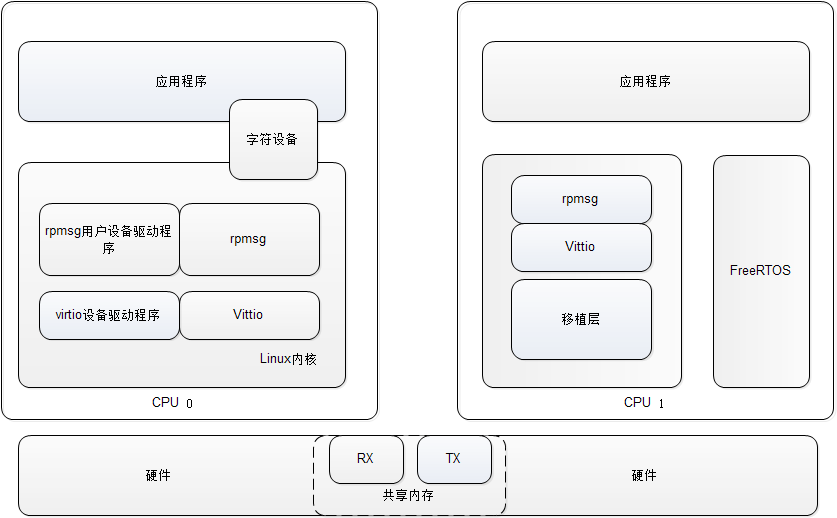


图1.系统框架

# 实现流程

如图2所示，在CPU0中，实现流程主要分为以下部分：

* 完成相关软件环境初始化，包括内存中断资源申请，内存映射等；
* 注册并创建用于核间通讯的VirtIO设备。设备注册完成后系统会自动扫描总线上的驱动，如若匹配到相应的驱动程序，将自动完成驱动的安装。
* 加载virtio设备驱动模块，这里采用动态加载的方式将驱动模块动态地加载到Linux内核中。驱动加载成功后系统将自动扫描总线上的设备节点，如若匹配到相应的设备节点，将自动运行驱动程序，完成设备和驱动程序绑定。
* 动态加载RPMsg设备驱动，为创建RPMsg通讯通道做准备。
* 等待CPU1发送RPMsg设备建立通知，即名称服务公告，rpmsg设备也即 通信通道。

在CPU1端搭载FreeRTOS操作系统，在其上运行的可执行程序建立过程如图2所示，其中的资源表信息主要包括使用的共享内存区域，virtio设备信息，包括TX virtqueue(数据发送虚拟队列)和RX virtqueue(数据接收虚拟队列)。一旦该可执行文件在FreeRTOS环境下运行后将完成以下任务：

* 完成相关软件环境初始化，包括内存中断资源申请，内存映射等；
* 根据资源表定义，完成中断注册和内存，并完成相关中断和内存初始化等；
* 根据资源表定义，开辟共享内存区域；
* 创建VirtIO设备和RPMSG设备；
* 向CPU0发送名称服务公告，告知已宣布的RPMSG器件，通知其协助完成RPMSG通信通道的建立。

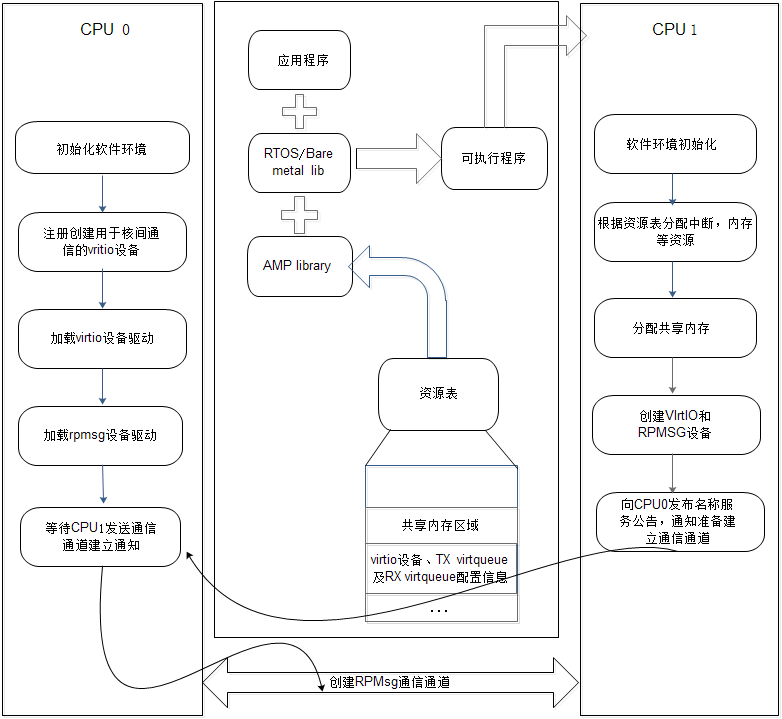


图2 实现框架流程

CPU1端向CPU0发布名称服务公告，CPU0根据接收到的名称服务公告中已宣布的rpmsg器件建立起对应的rpmsg通道。通道建立完成后，在两侧将调用由rpmsg通道创建的回调，通知两侧的运用通道已建立。此时，主机和远程环境可利用分别针对分块和不分块传输请求的 rpmsg\_sendxx API 和 rpmsg\_trysendxx API 相互传输数据。rpmsg 驱动程序使用 VirtIO 层提供的服务与对方实现共享存储器通信。rpmsg 驱动程序实例化一个 rpmsg VirtIO 器件，并使用 VirtQueue 接口推送和消耗其通信另一方的数据。

# VirtIO框架

VirtIO使用virtqueue来实现I/O机制，每一个virtqueue就是一个可承载大量数据的queue。每个设备可以拥有多个virtqueue用于大块数据的传输。Virtqueue是一个简单的队列，Vring是virtqueue的具体实现方式，针对Vring会有相应的描述符表格进行描述。框架如图3所示。

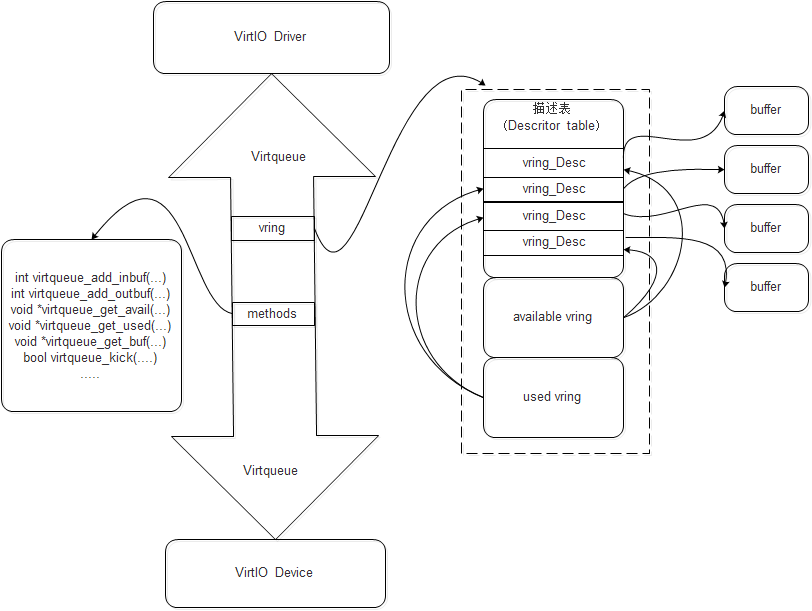


图3 VirtIO框架结构

VRing的具体实现依赖三部分：

1. 描述符表，在描述符表每一项都记录着buffer的地址(addr)、长度(len)、链接下一个描述符的next以及标识buffer只读/只写/可读可写的标志位(flags),flags的值得含义包括如下：

VRING\_DESC\_F\_NEXT:用于表明当前buffer的下一个域是否有效，也间接表明当前buffer是否是buffers list的最后一个。

VRING\_DESC\_F\_WRITE:表明当前buffer是read-only还是write-only还是read\_and\_write。

VRING\_DESC\_F\_INDIRECT:表明buffer的分配方式。

描述表结构如图4所示。

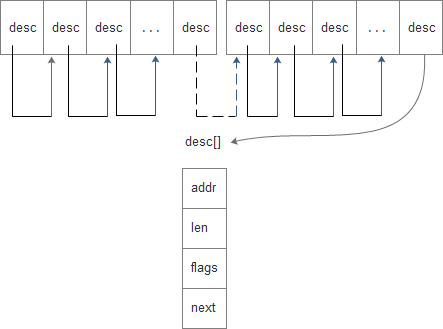


图4 描述表结构

1. available vring,该部分记录可写的buffer索引，在实际写buffer时是通过available vring中记录的索引值在描述符表(vring\_desc)中寻找空闲的buffer；available vring的结构如图5所示。

available vring提供的ring是一个环形缓冲区，每一项指向一个desc项，通过avail.ring[]便可以实现相应buffer的间接访问。

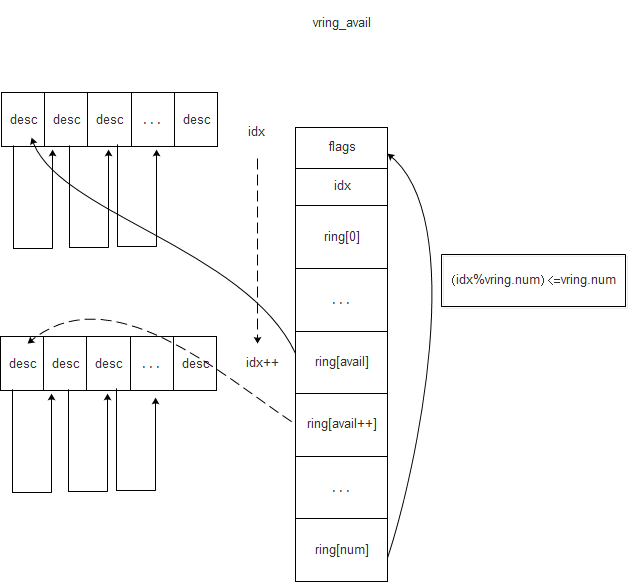


图5 available vring结构

1. used vring，该部分与available vring相似，但是used vring用于记录可读的buffer索引，在实际读buffer时通过used vring中记录的索引值在描述符表中寻找写有数据的buffer。Used vring的结果如图6所示。

Used vring提供的ring是一个环形缓冲区，每一项指向一个desc项，通过used.ring[]便可以实现相应buffer的间接访问。

每一个buffer都是物理上连续的一块存储区域。Virtqueue通过config\_ops配置项来实现buffer添加/获取buffer的virtqueue\_add\_xxbuf()和virtqueue\_get\_buf()以及在available vring有更新时通知另一侧进行数据读取的virtqueue\_kick()操作等。

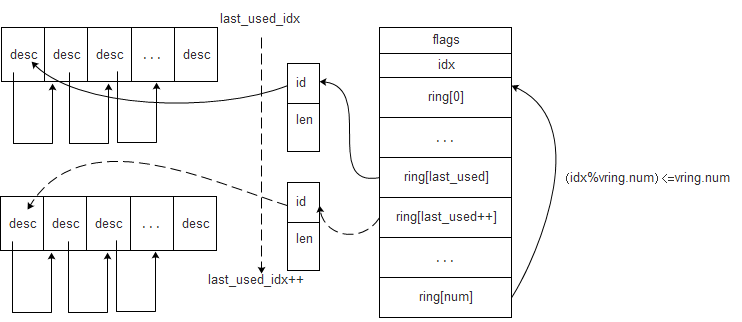


图6 used vring结构

# RPMSG设备驱动模块

RPMsg器件也可以理解为通信管道()。每一个RPMsg通信管道都有其各自的文本名称以及本地源()和目标地址()。RPMsg框架通过管道的文本名称对管道进行检索。同时，在这些通道上允许在其两头建立起多个RPMsg端点(),可以说RPMsg端点是一种逻辑抽象，用于实现处理核之间数据的定向传输。在RPMsg通道的任意一侧可以创建任意数量的端点，而且RPMsg端点的使用允许在同一个通道建立起多重回调。

所有RPMsg端点都有其各自唯一的地址以及相关联的回调函数()。在给RPMsg信道两端建立端点的时候，需要为其提供一个唯一的本地源地址()或者让RPMsg组件为其默认分配一个，作为在通信过程中锁定消息发送和接收端的索引。在所有目标地址()与本地源地址相等的端点将与创建端点时提供的回调函数相关联。为了使程序在没有创建新的端点的时候依旧能够进行通信，在创建RPMsg通道时将会为每一个通道建立一个默认的端点。

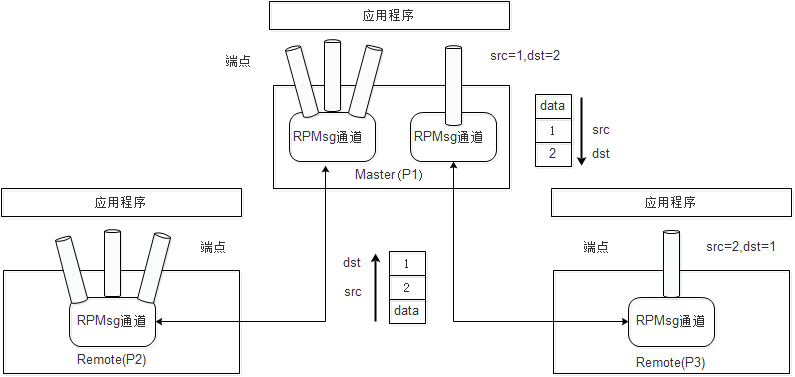


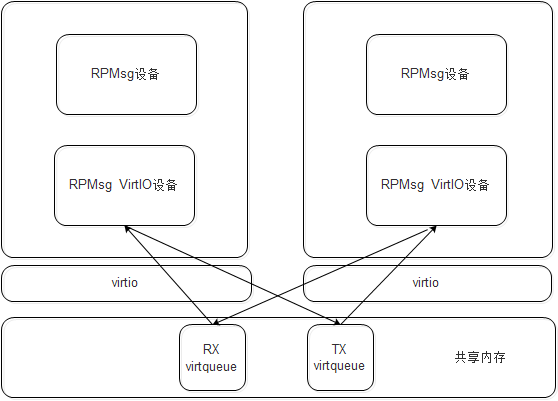
图7 RPMsg端点抽象

RPMsg通道和端点概念如图7描述。假设一个有P1,P2,P3三个核的多核系统， P1核RPMsg通道中的一个端点的src与dst值分别与P3核RPMsg通道中端点的dst和src值对应相等。因此这两个端点将建立起关联，主核根据端点的src和dst值便可以检索到相关联的端点，并通过RPMsg通信通道进行数据传输。

RPMsg驱动通过回调为输入的数据生成公告消息，RPMsg驱动里的公告处理机制会将消息数据直接发送给目的地址对应的程序。RPMsg驱动生成的公告事件主要有两种:数据接收完成(**RX Completion**)和通道创建/删除事件。该事件是当数据被相应的通信接收端接收后由RPMsg驱动程序产生的，而通道创建/删除事件在主机接收到远程环境发来的名称公告服务时被触发。

CPU0一旦接收到CPU1通过RPMsg组件发来的名称服务公告后，就会对相应的**RPMsg**器件进行注册，同时向远程端发送名称公告确认信息，并建立起通信通道。

RPMsg基于Virtio实现多核处理器IPC通信。实现过程简要描述如图8.

图8 rpmsg