第 36 卷 第 8 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.8 Computer Engineering April 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)08—0264—03 文献标识码：A 中图分类号：N945

RTEMS 下 Firewire 协议栈的设计与实现

张纪胜，陈香兰，周学海
(中国科学技术大学计算机科学与技术学院，合肥 230027)

摘 要：目前 RTEMS 操作系统缺乏对火线的支持。针对 Linux 和 RTEMS 在中断机制、定时器机制、延迟过程调用机制等方面的较大差异，提出一种高效的解决方案，实现 Linux 新火线协议栈到 RTEMS 的移植。在管理等时传输缓冲区时，结合等时传输的特点，修改环形队列的入队算法，解决可能的缓冲区满的问题。结果表明，实现的火线协议栈已达到对火线协议基本功能支持的目标。

关键词：实时多处理器系统；火线；设备驱动；协议栈

Design and Implementation of Firewire Protocol Stack on RTEMS

ZHANG Ji-sheng, CHEN Xiang-lan, ZHOU Xue-hai
(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027)

【Abstract】 Currently, the Real Time Executive for Multiprocessor Systems(RTEMS) operating system is lack of support for Firewire. This paper implements an efficient mechanism to eliminate the difference of interrupt, timer, deferred procedure call, etc. between Linux and RTEMS. The new Linux Firewire stack is ported to RTEMS successfully. During the porting program, on the basis of characteristic of isochronous transmission, it also modifies the enqueue algorithm of circular queue to solve the overflow problem of driver buffer. Result shows that the goal of basic support for Firewire is achieved.

【Key words】 Real Time Executive for Multiprocessor Systems(RTEMS); Firewire; device driver; protocol stack

1 概述

RTEMS(Real Time Executive for Multiprocessor Systems)最早用于美国国防系统，早期的名称为实时导弹系统(Real Time Executive for Missile Systems)，后来改名为实时多处理器系统(Real Time Executive for Military Systems, RTEMS)，现在由 OAR 公司负责版本的升级与维护。RTEMS 是免费的、源码公开的嵌入式实时操作系统，其部分性能指标甚至超过了著名的商用实时操作系统 Vxworks[1]。

RTEMS 具有如下特点：支持多种处理器体系结构，支持同构或异构的多处理器系统，裁剪性和移植性非常好，支持多任务和任务间的同步与通信，支持事件驱动、基于优先级的实时调度，有相当优秀的实时性能，支持优先级天花板协议(Priority Ceiling Protocols, PCP)，支持优先级继承，优良的中断管理，提供精简过的可重入的 libc 库。

Firewire(火线)即 IEEE 1394 总线，是苹果公司率先开发的一种高品质、高传输速率的串行总线技术，1995 年被 IEEE 命名为 IEEE 1394-1995 标准。其主要技术特征为：支持点对点传输，支持多种总线速度，支持热插拔即插即用，总线拓扑改变时自动配置，支持等时和异步 2 种传输方式，64 位的地址空间，可寻址 1 024 条总线的 63 个节点，每个节点有 256 TB 的地址空间等。该总线所具有的上述特征，使得它在视频传输、网络互联及计算机外设等领域得到广泛应用[2]。

1394 协议栈自顶向下分为 4 层[3]，如图 1 所示：总线管理层，事务层，链路层和物理层。总线管理层负责管理总线拓扑信息、节点速度信息、信道信息和带宽信息等；事务层为 read, write, lock 等 3 种异步传输操作提供支持；链路层负责等时或异步数据包的生成、分解、寻址、校验和信道解码等；物理层为数据位发送和接收、仲裁提供电气和机械接口。

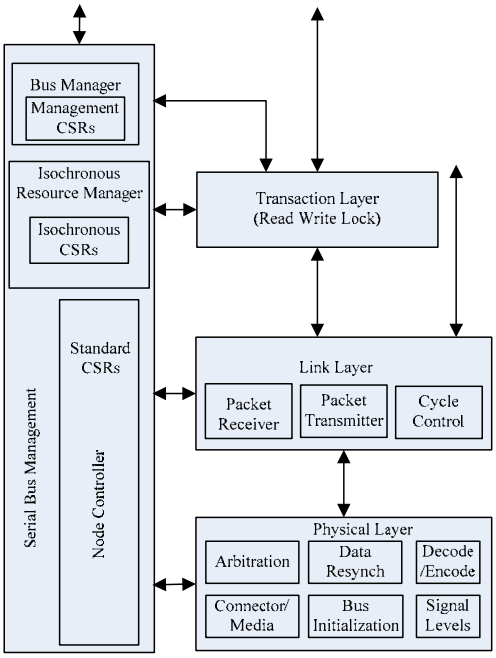

图 1 1394 协议栈示意图

2 协议栈设计

2.1 移植 Linux 新协议栈的理由

在实现 RTEMS Firewire 协议栈时有 2 种选择，一是从头

基金项目：安徽省自然科学基金资助项目(070412030)
作者简介：张纪胜(1983－)，男，硕士研究生，主研方向：嵌入式操作系统；陈香兰，讲师、博士；周学海，教授、博士生导师
收稿日期：2009-11-18 **E-mail：**jszhang3@gmail.com

写起(write from scratch)，二是移植目前已有的开源的 Firewire 协议栈实现，比如 FreeBSD 和 Linux 的实现。

其中，Linux 内核目前有 2 套 Firewire 协议栈，其一是从 2.4 时代就已有并不断完善的，另一个是 Linux kernel 2.6.22 引入的新协议栈。经过考察比较最后决定不重新发明“轮子”而是移植 Linux kernel 2.6.22 中的新协议栈，其理由如下：

(1)该协议栈代码架构清晰、适合移植和维护；

(2)代码量少，减小了编译后的目标文件大小；

(3)经少量工作就能移植好应用层库(libraw1394)获得库层次的兼容性；

(4)使用 libraw1394 的应用程序丰富；

(5)整个系统中每个 1394 设备都对应一个设备文件，可以获得更好的访问控制。

当然该协议栈也有不足之处，例如 eth1394 和 sbp 等功能还没有实现，并且硬件上只实现了 ohci 芯片的支持[4]。但对于无需 eth1394 和 sbp 功能的需求来说还是比较适合的。

2.2 RTEMS 与 Linux 驱动设计的不同

从驱动设计角度来看，RTEMS 和 Linux 内核相比除实现同样功能的 API 接口不同之外，最为关键的是它们在机制上有较大区别。

(1)Linux 系统分应用层和内核层，Linux 的设备驱动位于内核层，但 RTEMS 中没有这种区分，设备驱动和应用程序位于同一个层，这有利于应用程序和设备驱动共享数据，但同时也带来了一定的越界访问的危险。

(2)Linux 内核中提供中断底半部机制，如 softirq, tasklet, work queue，把一些非重要的工作从中断处理程序中抽出来，使得中断处理能够尽快完成，而 RTEMS 中则没有。

(3)Linux 内核有较为丰富的同步和互斥机制，比如原子操作、内存屏障、自旋锁、信号量和本地关中断等，而 RTEMS 4.8 版本之前只提供了信号量机制。

(4)Linux 内核中定时器触发于时钟中断底半部上下文中，而 RTEMS 中定时器触发于时钟中断处理上下文中，所以在 RTEMS 中应避免时钟中断处理时间过长，为此需要对执行时间较长的定时器谨慎设计。

(5)Linux 内核设备的主设备号可以动态分配而 RTEMS 中在 bsp 初始化时主设备号已经确定等。

2.3 解决方案

整个 Linux 1394 协议栈代码框架如图 2 所示，其中：(1)libraw1394 是向应用程序提供的一个库，它对底层 raw1394 设备驱动的读、写和控制接口做了相当完善的封装，更易使用。(2)raw1394 是一个抽象出的设备，该设备封装各种服务接口。(3)fw-iso 向 raw1394 设备提供具体的等时事务处理功能。(4)fw-transaction 向 raw1394 设备提供具体异步事务处理功能，它对应于 1394 协议栈中的事务层。(5)fw-card 主要处理本地 1394 芯片的管理。(6)fw-device 则负责整个系统里所有的 1394 节点的管理，比如节点加入、删除。(7)fw-topology 则是负责管理系统中的拓扑信息，比如拓扑树的生成、更新等，它和 fw-card, fw-device 对应于 1394 协议栈中的管理层。(8)fw-chip 是底层链路层芯片的驱动，对应于 1394 协议栈中的链路层。

移植的主要难点除前面所述中断底半部和定时器等的处理之外，还需要根据具体芯片手册重写 fw-chip 这一层。同时由于嵌入式系统资源有限，因此等时传输缓冲区管理还须紧密结合嵌入式系统和等时传输的特点来实现。

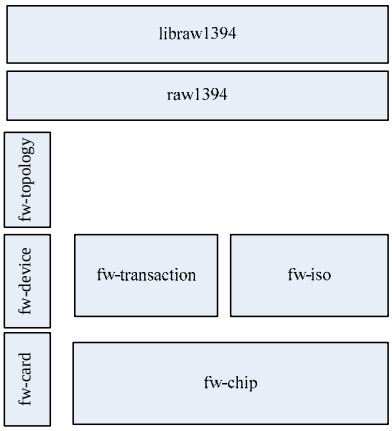

图 2　新 1394 协议栈代码框架

2.3.1　中断底半部的处理

对于 tasklet 中断底半部问题的处理办法如下：把协议栈原来运行于 tasklet 中的代码并入中断服务程序中。这是由于 1394 芯片的中断使能位在 tasklet 中代码运行结束后才被置位，上述处理办法相对于 Linux 中的情形并不会额外增加进程被中断的时间，也不会额外增加 1394 芯片的关中断的时间。

对于原来使用 work queue 机制的代码，为能与原来代码兼容从而减少代码的改动，定义了 work_struct 结构体：

```
struct work_struct {
  work_func_t func;
};
```

相应地，在协议栈初始化时创建一个服务于中断的任务 fw_irqbth_timer_task、一个消息队列 FWWQ 和一定数目的定时器。fw_irqbth_timer_task 的关键流程如下：该任务平时因接收消息而阻塞在消息队列上，接收到消息后从消息的数据域中取出函数指针和参数执行。

启动与 Linux 中 work queue 相似机制的方法如下：

(1)对于原来需要延时执行的(比如以非 0 的延时参数使用 schedule_delayed_work 启动)work_queue 操作，启动一个定时器，在延时到达后触发定时器，由定时器函数 fw_timer 构造好消息并发送到消息队列 FWWQ 上。

(2)对于原来不需要延时的(比如使用 0 作为延时参数启动 schedule_delayed_work)work_queue 操作则直接构造消息并发送到消息队列 FWWQ 上。

2.3.2　原定时器的处理

为减少对原 Linux 代码的改动，同样定义了 timer_list 结构体：

```
struct timer_list {
  void (*function)(unsigned long);
  unsigned long data;
};
```

为了利用定时功能需要在 RTEMS 中启动定时器，但如前所述，RTEMS 中的定时器函数运行在时钟中断处理上下文中。为避免定时器中断处理时间过长，需要把原 Linux 中定时器函数放在另一处执行，但又要让它尽快运行，而在 RTEMS 中启动的定时器，只是在时间到时通知定时器函数运行。前面为处理中断底半部而引入的任务和定时器函数恰好可以满足此条件，但要把中断底半部处理和定时器处理两者揉合起来，需要对该任务的核心流程和定时器函数 fw_timer 做出少量修改：任务接收到消息后需要先判断一下消息类型，

然后从消息的数据域中取出函数指针和参数；定时器函数发送消息前应根据不同的定时器构造不同的消息。其伪码如下：

```
void fw_timer(rtems_id id, void *data){
if(定时器是用于处理原 Linux 定时器的)
  msg_type = FW_WQ_MSG_TIMER_EVT;
else
  msg_type = FW_WQ_MSG_WORK_EVT;
发送消息;
}
static rtems_task
fw_irqbth_timer_task(rtems_task_argument arg){
for(;;){
从消息队列接收消息 msg;
switch(msg.type){case FW_WQ_MSG_WORK_EVT:
//处理原 work queue 中断底半部
{
从 msg.data 中取出 work 结构体
从 work 结构体中取出函数指针给 f
f(work);
break;
}
case FW_WQ_MSG_TIMER_EVT://处理原定时器
{
从 msg.data 中取出 timer 结构体;
从 timer 结构体中取出定时器函数指针给 f;
f(timer->data);
break;
}
default:
出错处理;
break;
}
}
}
```

2.3.3 fw-chip 层

fw-chip 层核心在于从中断寄存器中读出中断事件并根据不同的中断事件进行相应处理，中断服务程序(ISR)的伪码如下：

```
irq_handler
{
从中断寄存器中读出中断事件 event;
清除中断寄存器;
if 异步发送出错
    清空异步发送 FIFO 寄存器(ATF);
if 等时发送出错
  清空等时发送 FIFO 寄存器(ITF);
if event & TXREADY {
  if 异步发送队列非空 {
    数据包 A 从队列摘下;
    if 异步发送队列非空
      发送队列头所在数据包;
    对数据包 a 的后续处理;
  }
  if 等时发送已经开始 && 等时发送队列非空
    调用等时发送函数;
}
if event & RXDATA {
  读取 tcode;
  if 异步数据包
    调用异步接收函数;
  if 等时数据包 {
    if 等时接收已经开始
      读取数据并处理等时数据;
    else
      读出数据并丢弃;
  }
}
```

2.3.4 等时传输缓冲区

发起等时传输时分配 M 个大小为 N 的数据块作为等时传输的缓冲区(M 和 N 由应用程序确定)，每个数据块都有一个管理信息块，这 M 个管理信息块采用环形队列[5]来管理，另外附加 rear 和 front 指针分别指示生产者和消费者位置。生产者把数据放入队列尾部，消费者从队列头部移走数据。在等时发送时，发起等时传输的应用程序是生产者，中断处理程序是消费者；在等时接收时则恰好相反。

嵌入式系统上处理器能力有限，等时接收时有可能来不及处理数据而造成缓冲区满，同时等时传输是一种无应答的传输方式，它需要保证充分的实时性但不保证所传输数据的完整性，利用等时传输的这个特点修改了正常的环形队列的入队算法[5]，并不判断队列满的情形，显然这种情形下队列满后会自动丢弃队列中的数据。这样做的另外一个好处是提高了环形队列入队的速度。其算法伪码如下：

```
Status EnQueue(Queue &Q, ElemType e) {
  Q.base(Q.rear) = e;
  Q.rear = (Q.rear + 1) % MAXQSIZE;
  return OK;
}
```

3 结束语

本文基于 Linux 新火线驱动协议栈，为 RTEMS 实时操作系统开发了功能比较全面的火线驱动协议栈。在协议栈移植过程中分析了 Linux 和 RTEMS 这 2 个操作系统驱动设计的不同，并为这些不同找到了解决方案，尤其是提出了一种高效的仿 Linux 内核中断底半部的机制，在不大量改动代码和不影响性能的前提下解决了协议栈的中断底半部和定时器的移植问题。在管理等时传输缓冲区时结合等时传输的特点修改环形队列的入队算法，解决了可能的缓冲区满的问题。至此，1394 驱动协议栈的基本功能已经在 RTEMS 上实现。但是此协议栈还不支持 eth1394 和 sbp，下一步的工作是在此协议栈基础上加入对这两者的支持并进一步完善它。

参考文献

[1] Straumann T. Open Source Real Time Operating Systems Overview[C]//Proc. of the 8th International Conference on Accelerator & Large Experimental Physics Control System. San Jose, USA: [s. n.], 2001.

[2] 张大朴, 王 晓, 张大为, 等. IEEE 1394 协议及接口设计[M]. 西安: 西安电子科技大学出版社, 2004.

[3] Microprocessor and Microcomputer Standards Committee of the IEEE Computer Society. IEEE 1394-1995 IEEE Standard for a High Performance Serial Bus[S]. 1995.

[4] Høgsberg K. New Firewire Stack[EB/OL]. (2008-11-12). http://lkml.org/lkml/2007/5/1/460.

[5] 严蔚敏, 吴伟民. 数据结构(C 语言版)[M]. 北京: 清华大学出版社, 2002.

编辑 任吉慧