第 37 卷 第 17 期
Vol.37 No.17

计 算 机 工 程
Computer Engineering

2011 年 9 月
September 2011

• 工程应用技术与实现 • 文章编号：1000—3428(2011)17—0242—03 文献标识码：A 中图分类号：TP316

μC/OS-Ⅱ中断底半部机制的设计与实现

褚东升，孟德国，黎 明

(中国海洋大学工程学院，山东 青岛 266100)

摘 要：设计并实现一种应用于 μC/OS-Ⅱ系统的中断底半部机制。该机制与内核本身提供的信号量、消息邮箱等机制位于同层，同时支持有优先级的中断底半部静态触发和无优先级的动态注册。在大型海洋资料浮标项目中的应用结果表明，该机制可有效提高系统的响应速度和稳定性。

关键词：μC/OS-Ⅱ实时操作系统；中断底半部；推后执行；进程调度；嵌入式系统

Design and Implementation of Interrupt Bottom Half Mechanism for μC/OS-Ⅱ

CHU Dong-sheng, MENG De-guo, LI Ming

(College of Engineering, Ocean University of China, Qingdao 266100, China)

【Abstract】For μC/OS- doesn't support bottom half interrupt, this paper presents an implementation scheme of bottom half mechanism for μC/OS-Ⅱ. According to the priorities of interrupts, it can be triggered statically by some special interrupts with priorities, and it also supports dynamic registration for interrupts without priorities. This mechanism improves the system performance. This mechanism has already been achieved, and used in ocean data buoy system successfully. Result shows that it can improve the system response speed and stability.

【Key words】μC/OS-Ⅱ real-time Operating System(OS); interrupt bottom half; deferrable function; process schedule; embedded system

DOI: 10.3969/j.issn.1000-3428.2011.17.081

1 概述

μC/OS-Ⅱ是一个小型的嵌入式实时操作系统，支持基于优先级的抢先调度，具有结构精巧、可靠稳定、易学易用且源代码开放的特点，在许多场合得到了广泛应用，已被先后移植到 40 多种不同架构的 CPU 上。

不管中断服务程序执行期间系统中断是否被屏蔽，系统中断服务程序都不可过长。这就要求中断服务程序尽可能短，即只在其中做必要的、最紧急的工作。为此 Linux 等操作系统提供了中断底半部机制[1-2]。中断底半部是指中断处理密切相关而中断本身又不做的那部分工作，它是一种重要的推后执行方式。中断底半部的必要性体现在 2 个方面：(1)中断服务程序中的大部分工作虽然紧急，但并不是必须在中断服务程序中处理它们，这部分任务可以被稍微推后执行(Linux 中被推后执行的任务被中断调度，并且此后很快就被执行[3])。(2)中断并没有进程上下文，有些工作不便于在其中实现。

一般中断服务程序中只要做必要的、最紧急的工作，中断底半部程序所完成的恰恰就是中断中该做而未做的那部分紧急度次之的、可以被推后执行的工作。但 μC/OS-Ⅱ并没有像 Linux 那样提供中断底半部机制，大量的任务都要留给中断来做，将这种设计方案应用于大型的、存在大量异步事件的嵌入式系统是有风险的。为此，本文设计 μC/OS-Ⅱ的中断底半部机制，实现了一个与内核本身提供的信号量，消息邮箱等同等层次的内核机制，开发了专用的中断底半部进程和对应的新的中断服务程序的实现方法。

2 底半部机制设计

由上述分析可知，被推后执行的任务应该在一个优先级相对较低的、安全的环境中运行，这里的安全是指中断是打开的，即中断底半部机制本质上要求被推后执行的底半部任务在一个用户态进程的级别上运行，使它可以被中断，并且有这样的运行机制：在没有等待处理的底半部任务时，将自身挂起并触发进程调度，有了等待处理的底半部任务时该进程被唤醒。现有的内核中没有直接满足这些要求的结构，为此本文实现了融入 μC/OS-Ⅱ内核的中断底半部机制(BH 机制)，称该部分其为 BH_Core，该部分由 2 个函数(bh_pend、bh_post)以及 2 种专用结构(BH_struct 和 tasklet)组成其命名遵循了 μC/OS-Ⅱ的习惯。图 1 为包含中断底半部机制的 μC/OS-Ⅱ系统结构。

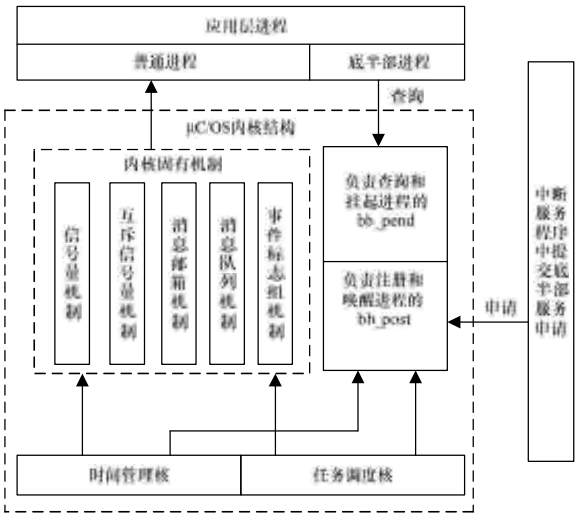

图 1 包含中断底半部机制的μC/OS-Ⅱ系统结构

作者简介：褚东升(1956－)，男，教授，主研方向：智能信息处理，智能控制；孟德国，硕士研究生；黎 明，副教授

收稿日期：2011-03-21 **E-mail：**mengdeguo@163.com

由图 1 可以看出，实现底半部机制的各个功能函数，与内核中的其他机制的实现结构并列。中断底半部机制同样以μC/OS-Ⅱ的时间调度核和任务管理核为基础，接收中断服务程序的服务申请，并完成对专用的底半部服务进程的调度。

bh_pend 函数流程如图 2 所示。该函数可能引起进程的切换，而在中断服务程序中或者μC/OS-Ⅱ的调度结构被锁的情况下，都不能作进程切换，因此，函数的一开始就是关于这个问题的 2 个判断。接下来判断如果有已经注册了的等待处理的底半部任务，则该函数正常返回，那么调用该函数的进程便可依据底半部的申请而处理各底半部任务。如果没有要处理的任务，则将当前进程挂起，此时一定要将当前进程设为暂停状态，并将任务控制块中的字段 OSTCBDly 的值设为 0，这是由μC/OS-Ⅱ的时间管理核[4-5]的实现决定的。执行底半部服务的进程只有在有要处理的底半部任务时才运行，否则一直挂起，这里不存在因挂起超时而运行的情况。触发进程调度即调用内核函数 OS_Sched()，它会保存当前进程的上下文，并将处于就绪态的优先级最高的进程投入运行。当 OS_Sched()函数返回时，只有一种可能那就是有要处理的底半部任务(前面已经说过不可能是挂起超时)，因此，该函数也是正常返回。

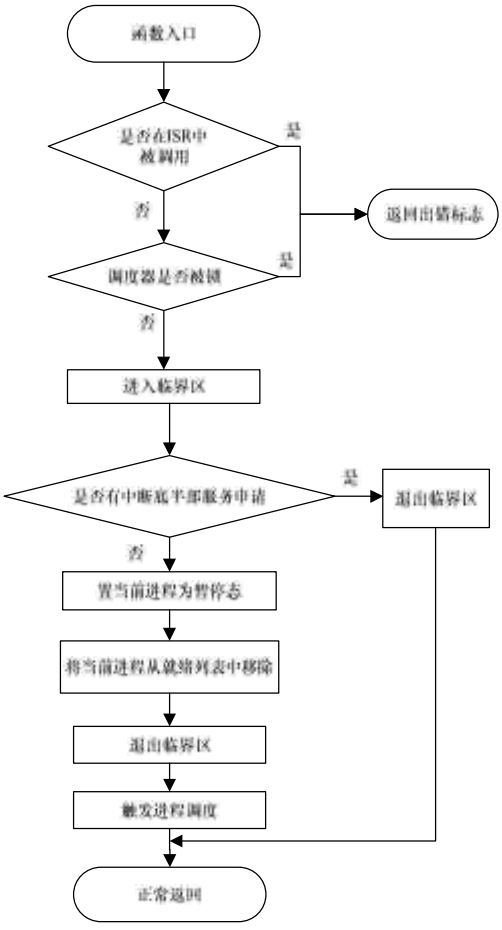

图 2　bh_pend 函数流程

bh_post 函数流程如图 3 所示。在说明其流程之前，先说明为实现底半部机制而引入的特殊数据结构：BH_struct，它的 2 个关键字段是：mask 和 taskletCnt。设计中考虑到多个底半部申请同时产生时，必须要有优先级的区分，这个优先级是静态的，目前的实现只是给定时器、网络发送和网络接收这几个对时间要求严格的底半部任务分配了优先级，其优先级由 mask 字段中的相应位标识，当这些底半部服务申请产生时，mask 的相应位被置位，这些任务便被触发并依优先级被顺序处理。另外，mask 中还有一个 TASKLET_BH 标志，这个标志标识有其他类型的底半部服务申请，由于这些申请种类多样，而且也不像前几种服务请求那样急迫，这里用统一的无优先级机制来处理它们(确切说是先来先服务机制)。为此引入 tasklet_t 数据结构，它有 2 个字段：func 和 data，func 是一个函数指针，用于保存完成底半部任务的回调函数。而 data 是该 func 指向的函数的参数，为支持任意类型参数，data 是 void *类型。考虑到一般嵌入式系统，在一个较小的时间段内，产生的底半部申请不会超过 32 个，定义数组 tasklet_t g_taskletArray[32]来存放所有申请，这样每次有这种申请时，一方面 mask 的 TASKLET_BH 位被置位。另一方面，等待执行的底半部函数及其参数被动态注册到 g_taskletArray[taskletCnt]中，即用 taskletCnt 字段来记录这种底半部申请的数目。

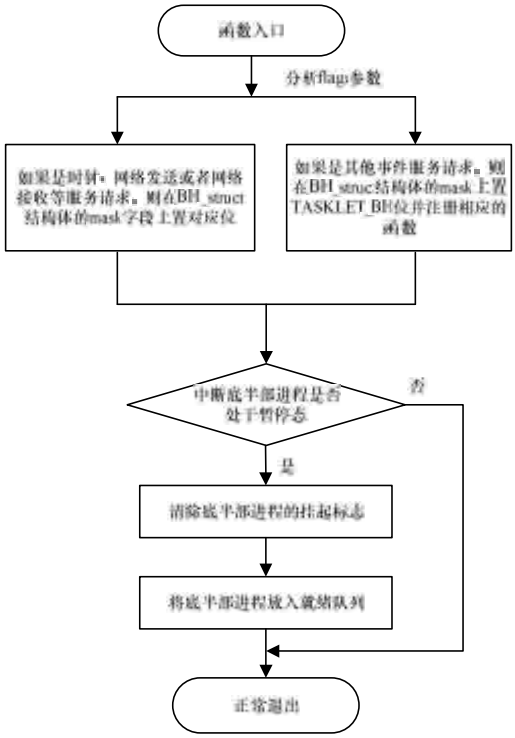

图 3　bh_post 函数流程

图 3 中函数分析 ISR 类型目的在区分上面提到的两大类底半部服务申请，依据类型的不同做不同的处理。接下来检查底半部服务进程是否被设为暂停态，如果没设该标志则说明当前进程正在运行或被高优先级的进程抢占了 CPU，此时底半部服务进程仍在就续表，所以不用再将其重新设为就绪态。这步判断也使得在确定底半部服务进程的用户态优先级时可以有更大的灵活性，该进程不必总是优先级最高的进程(多数情况下它应该是优先级最高的任务，这样可以保证底半部任务在中断发生后迅速被处理)。在有大量异步事件的系统中，底半部进程可能长时间连续运行，如果它被设为优先级最高的进程，可能会影响其他进程的响应速度，这时可以将其优先级适当的设低些，这样可以保证底半部服务在系统空闲时运行。如果底半部服务进程被设置为暂停态，则接下来要将它重新放回就绪列表，然后退出。由于该函数一般在中断服务程序中被调用，因此它不去触发进程调度，等到嵌套的最外层的中断服务函数返回时，进程调度自然会被触发。

有以上新的内核机制的支持之后，即可改写中断服务流

程并且设计底半部服务进程。新的中断服务进程流程如图 4 所示。

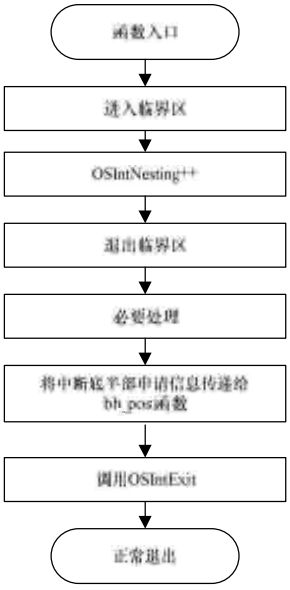

图 4 ISR 流程

该过程与原μC/OS-Ⅱ中断服务的经典流程的区别在于：做完必要的处理之后就调用了 bh_post 函数来注册中断底半部服务函数，同时也是在调度底半部服务进程，将这部分工作交给底半部服务进程来处理。而不是自己处理。

底半部服务进程的流程如图 5 所示。

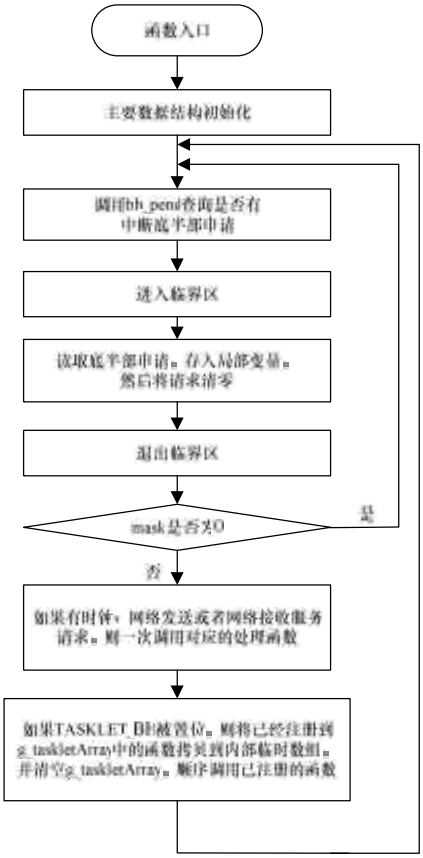

图 5 底半部进程流程

在初始化必要的数据结构之后，进入任务循环，在循环开始调用 bh_pend 有多方面的作用，一方面该进程初次执行时，可能还没有待处理的底半部服务请求，这时直接将进程挂起，另外，在该进程执行期间，可能有新的底半部服务请求被注册，这样一趟循环结束，接着判断是不是有新的请求，如果有则接着处理，这样保证了任务的快速响应。在该进程执行过程中，新的请求可能产生，这时全局的 mask 和全局的 g_taskletArray 是典型的临界资源，所以在进程中将这些结构拷贝到内部的局部变量，并对全局变量清零，这样的操作才是安全的。

3 应用分析

目前该方案已经成功运用到中国海洋大学与国家海洋局东海分局联合研发的大型海洋资料浮标采集系统中，该系统同时挂接 13 个传感器，其中多数为串口传感器，协议相对复杂，加之风速、风向、方位等量要不停地采集，这一切的实现都有赖于系统的响应速度，关中断的时间尽可能的短。

表 1 说明了采用中断底半部机制前后部分各中断服务程序代码行数的变化。

表 1 中断代码统计

中断名称	代码行数	
	采用底半部机制前	采用底半部机制后
系统时钟中断	85	20
CAN 收发中断	67	27
北斗串口收发中断	77	31
风速 timer 计数中断	56	18
多路 AD 的 DMA 控制中断	108	32
方位串口收发中断	62	25
波浪串口收发中断	75	28

中断底半部机制的引入，明显减少了系统中多数中断服务函数的代码，中断的关闭时间也相应缩短，直接提高了系统的响应速度。缩短了关闭中断的时间也就提高了多数传感器的采集精度，降低了为提高精度而带来的算法实现上的复杂性，使系统更加稳定。

4 结束语

本文提出了一种针对 μC/OS-Ⅱ的中断底半部机制的实现方案，解决了 μC/OS-Ⅱ没有中断底半部服务的问题。该解决方案具有同时支持有优先级的中断底半部静态触发和无优先级的动态注册的特点，能有效扩展μC/OS-Ⅱ的内核。

参考文献

[1] Love R. Linux 内核设计与实现[M]. 陈莉君, 康 华, 张 波, 等, 译. 2 版. 北京: 机械工业出版社, 2006.

[2] Bovet D P, Cesati M. 深入理解 Linux 内核[M]. 陈莉君, 张琼声, 张宏伟, 译. 北京: 中国电力出版社, 2007.

[3] Corbet J, Rubini A, Kroah-Hartman G. Linux 设备驱动程序[M]. 魏永明, 耿 岳, 钟书毅, 译. 3 版. 北京: 中国电力出版社, 2006.

[4] 韩明峰, 李小滨, 郑永志. μC/OS-Ⅱ内核超时等待机制分析与改进[J]. 计算机工程, 2009, 35(7): 259-260.

[5] Labrosse J. 嵌入式实时操作系统μC/OS-Ⅱ[M]. 邵贝贝, 译. 2 版. 北京: 北京航空航天大学出版社, 2003.

编辑 金胡考