第 34 卷 第 1 期 计 算 机 工 程 2008 年 1 月
Vol.34 No.1 Computer Engineering January 2008
·软件技术与数据库· 文章编号：1000—3428(2008)01—0066—03 文献标识码：A 中图分类号：TP316.2

一种改进型优先级天花板协议设计与实现

刘 鹏，牛 强，陈 岱，张宝辉
(中国矿业大学计算机学院，徐州 221008)

摘 要：分析了嵌入式操作系统μC/OS-II 产生的优先级反转问题及缺陷，提出了一种改进型优先级天花板协议，给出了协议的原理，该协议可以解决优先级反转问题，具有预防系统死锁等特点，并介绍了“修改μC/OS-II 内核以实现改进型优先级天花板协议”的思想和方法。
关键词：优先级反转问题；优先级继承协议；改进型优先级天花板协议

Design and Implementation of Improved Priority Ceiling Protocol

LIU Peng, NIU Qiang, CHEN Dai, ZHANG Bao-hui
(School of Computer, China University of Mining and Technology, Xuzhou 221008)

【Abstract】 This paper analyzes why priority-reversal problem occurs in μC/OS-II, and the limitation of μC/OS-II. An improved priority ceiling protocol(IPCP) is introduced, which including its principle, basic content. The protocol solves priority reversal problem, has other good characteristics, such as preventing system dead lock, etc. The approach of implementation of IPCP by modification of kernel of μC/OS-II is proposed.
【Key words】 priority reversal problem; priority inheritance protocol; improved priority ceiling protocol

μC/OS-II 是一个源码公开的嵌入式实时操作系统，可以支持多达 64 个优先级任务，并支持信号量、消息队列、邮箱等多种常用的进程间通信机制。由于系统占用代码空间小、运行效率高、稳定可靠，因此在通信、电子、自动化等领域的嵌入式设备中获得了广泛的应用。但是，在嵌入式系统的应用中，实时性是一个很关键的指标，而μC/OS-II 固有的“优先级反转问题”却影响着系统的实时性保证，它会从整体结构上破坏系统的实时性。本文在分析μC/OS-II 现有的优先级反转问题解决方案“优先级继承协议”的基础上，提出了一种“改进型优先级天花板协议”。

1 μC/OS-II 优先级反转问题原因

在μC/OS-II 中，多个任务按照优先级高低由内核调度执行，而且任务调度所花的时间是常数，与应用程序中建立的任务数无关。对于占先式内核，任务的响应时间是确定的，它保证最高优先级的任务最先执行。但是，多任务实时操作系统常常会出现优先级反转问题，此时，已经不能保证高优先级任务的响应时间，系统实时性得到严重的破坏。

如图 1 所示，任务 1 优先级高于任务 2，任务 2 优先级高于任务 3。任务 1 和任务 2 处于挂起状态，等待某一事件的发生，任务 3 正在运行(图 1 中的(1))。此时，任务 3 要使用某临界资源。使用临界资源之前，首先必须得到该资源的信号量(semaphore)。任务 3 得到了该信号量，并开始使用该临界资源(图 1 中的(2))。由于任务 1 优先级高，因此它等待的事件到来之后剥夺了任务 3 的 CPU 使用权(图 1 中的(3))，任务 1 开始运行(图 1 中的(4))。运行过程中任务 1 也要使用那个任务 3 正在使用着的资源，由于该资源的信号量还被任务 3 占用着，因此任务 1 只能进入挂起状态，等待任务 3 释放该信号量(图 1 中的(5))。任务 3 得以继续运行(图 1 中的(6))。由于任务 2 的优先级高于任务 3，因此当任务 2 等待的事件发生后，任务 2 剥夺了任务 3 的 CPU 的使用权(图 1 中的(7))并开始运行，处理它该处理的事件(图 1 中的(8))，直到处理完之后将 CPU 控制权还给任务 3(图 1 中的(9))。任务 3 接着运行(图 1 中的(10))，直到释放那个临界资源的信号量(图 1 中的(11))。直到此时，由于实时内核知道有个高优先级的任务在等待这个信号量，因此内核进行任务切换，使任务 1 得到该信号量并接着运行(图 1 中的(12))。

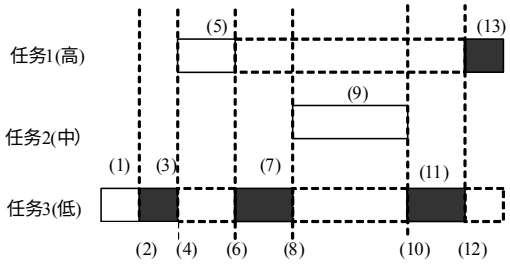

图 1 μC/OS-II 产生优先级反转问题

在这种情况下，任务 1 优先级实际上降到了任务 3 的优先级水平。任务 1 要等待，直至等到任务 3 释放占有的那个临界资源。任务 2 剥夺任务 3 的CPU使用权，使任务 1 的状况更加恶化，任务 2 使任务 1 增加了额外的延迟时间，即任务 1 和任务 2 的优先级发生了反转。在更恶劣的情况下，譬如任务 1 和任务 3 之间有多个这样的“任务 2”存在，这样的优先级反转问题可能会导致整个系统的崩溃[1]。

基金项目：中国矿业大学青年科技基金资助项目(OD4545)
作者简介：刘 鹏(1927－)，男，讲师、博士研究生，主研方向：嵌入式系统及其在下一代网络中的应用；牛 强，博士研究生；陈 岱，副教授；张宝辉，学士
收稿日期：2007-03-05 **E-mail：**liupeng@cumt.edu.cn

2 μC/OS-II 自身对优先级反转问题的解决方案

在 μC/OS-II 中，可以使用互斥型信号量实现对临界资源的独占访问，并可以由此来解决优先级反转问题。具体做法是：首先编程人员确定所有建立任务的最高优先级，再确定继承优先级(Inheritance Priority, IP)，即在应用程序中没有被占用的、略高于最高优先级任务的优先级，最后应用创建互斥型信号量函数 OSMutexCreate()建立互斥型信号量，此时 IP 作为函数的第 1 个参数，表示使用该信号量的极限优先级，这样当出现优先级反转时，将正在使用信号量的低优先级任务的优先级提升到 IP，从而保证其正常运行，并在运行完成后释放信号量，这样就可以保证系统整体的实时性不被破坏[2]。但是，这种“优先级继承协议”不能防止死锁以及阻塞链(传递阻塞)的发生[3]，由此可见，“使用优先级继承协议来解决μC/OS-II中的优先级反转问题”的方法存在不足之处。

3 在 μC/OS-II 中实现改进型优先级天花板协议

目前解决实时操作系统优先级反转问题比较普遍使用的有两种方法：优先级继承协议和优先级天花板协议。后者比前者有了一些改善，但仍然存在明显的不足。

3.1 优先级天花板协议分析及其改进

最初的优先级天花板协议(版本 1)规定：

(1)对于控制临界区资源的信号量，设置信号量的“优先级天花板”为将要申请该信号量的所有任务中具有最高优先级任务的优先级。

(2)如果某个任务成功获得信号量，则该任务的优先级将被提升为该信号量的优先级天花板；任务使用资源完毕释放信号量后，其优先级恢复到它原先的优先级。

(3)如果任务不能获得所申请的信号量，则任务被阻塞[4]。

但是，当一个任务占有信号量时，会继承此信号量的天花板优先级，可能出现两个任务具有相同优先级的情况，这是μC/OS-II所不允许的。对此，可以保留一个略高于所有使用此信号量的任务优先级的优先级作为此信号量的天花板优先级，把这个优先级称作PCP(Priority Ceiling Priority)[5]。这样，当占有信号量的任务继承此信号量的PCP时，就不会出现任务有相同优先级的情况，这样的修改既满足了μC/OS-II对任务优先级的限制，又不会改变优先级天花板的正确语义。

修改后的优先级天花板协议(版本 2)如下：

(1)对于控制临界区资源的信号量，设置信号量的优先级天花板 PCP。

(2)如果任务成功地获得了信号量，任务的优先级将被提升为它所占有的所有信号量的优先级天花板中最高的那个优先级。当任务使用完资源，释放信号量后，其优先级恢复到原先的优先级。

(3)如果任务不能获得所申请的信号量，任务被阻塞[6]。

优先级天花板协议(版本 2)解决了优先级反转问题，避免了优先级继承中的死锁与阻塞链问题，以及原有优先级天花板协议中 μC/OS-II 所不支持的同优先级下多个任务的问题。

但是，这个版本 2 改进方案仍然不很理想，存在着如下的问题(图 2)：任务 1 优先级高于任务 2，任务 2 优先级高于任务 3。任务 1、任务 2 处于挂起状态，等待某一事件的发生，任务 3 正在运行(图 2 中的(1))。此时任务 3 要使用某临界资源，得到信号量后，根据优先级天花板协议(版本 2)，系统提升任务 3 的优先级至信号量的 PCP(图 2 中的(2),(3))。任务 2 等待的事件发生，任务 2 进入就绪态，但此时任务 3 的优先级已经被提升至 PCP，任务 3 继续运行(图 2 中的(4),(5))。任务 3 使用完该临界资源并释放信号量后，系统恢复其至原来的优先级，此时任务 2 才得以运行(图 2 中的(6),(7))。任务 2 运行结束后，任务 3 接着运行(图 2 中的(8),(9))。任务 1 等待的事件发生，任务 1 抢占 CPU 并开始运行(图 2 中的(10),(11))。任务 1 请求临界资源信号量成功，提升任务 1 优先级至此信号量的 PCP，任务 1 运行直至结束(图 2 中的(12),(13))。此时，任务 3 得到 CPU 并继续运行。

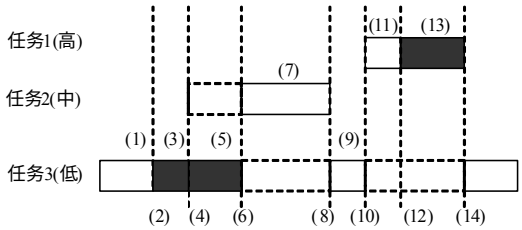

图 2 优先级天花板协议(版本 2)存在的问题

在这种情况下，虽然任务 3 占用临界资源时没有阻塞其他任务，但系统仍提升了它的优先级，这就导致了系统响应其他中间优先级任务(上例中的任务 2(图 2 中的(4)~(6))时间的增加，从而影响了系统整体的实时性。而且，“改变任务的优先级”操作本身是比较占用 CPU 时间的，这会进一步影响系统的实时性。

为防止发生优先级反转，内核最好只在“必需”时自动变换任务的优先级，只有这样，才能最大限度地保证系统的实时性。为了满足这个要求，经过对优先级天花板协议(版本 2)的仔细分析，可以采用如下解决方案：

(1)对于控制临界区资源的信号量，设置信号量的优先级天花板，信号量的优先级天花板等于 PCP。

(2)如果任务成功获得信号量，通常情况下，任务将在原有的优先级上运行。除非该任务在临界区的执行过程中阻塞了其他高优先级的任务，才提升占有资源信号量的任务的优先级至此信号量的 PCP。在任务释放信号量时，恢复其原有的优先级。

(3)如果任务不能获得所申请的信号量，则任务被阻塞。

上述经过修改过的优先级天花板协议(版本 3)，保留了版本 2 所具有的预防死锁及阻塞链的发生等优点，而且，只有当高优先级任务想要申请已被低优先级任务占有的资源这一事实发生时，才提升此低优先级任务的优先级为 PCP，因此，使用此协议对整个实时系统的多任务执行流程的影响已基本降至最低，即最大限度地保证了系统的实时性。这个优先级天花板协议(版本 3)称为“改进型优先级天花板协议”。

3.2 修改部分内核以实现改进型优先级天花板协议

一个事件控制块(ECB)代表一个被多个任务共享的资源，μC/OS-II 内核通过事件控制块来管理共享资源。基于改进型优先级天花板协议的事件控制块结构在系统启动时，初始化成单向线性空闲链表。示例代码如下：

```
typedef st ruct os_priceiling_event{
    INT8U OSEventPCP;      //资源的 PCP
    INT8U OSEventPri;      //保存任务的优先级
    struct os_priceiling_event *OSEventPtr;
//空闲时，指向链表中下一个事件控制块的指针，工作时用于保
//存任务先前的事件控制块
} OS_PRICEILING_EVENT;
```

如前所述，因为 μC/OS-II 内核通过事件控制块管理共享资源，所以需要在任务控制块(TCB) 中增加一个指向当前正

在被任务使用的事件控制块的指针 OSTCBCeilEventPtr，任务通过这个指针操纵当前的事件控制块。在多个资源被分配给一个任务后，事件控制块结构中的 OSEventPtr 指针和 TCB 中的 OSTCBCeilEventPtr 指针将形成一条事件控制块链，链头即为 OSTCBCeilEventPtr。示例代码如下：

```
/*如果不用优先级天花板，则定义
OS_PRICEILING_EN=0 以减少每个 TCB 所占内存*/
# if   OS_PRICEILING_EN
OS_PRICEILING_EVENT *OSTCBCeilEventPtr;
# endif
```

最后为 μC/OS-II 内核补充编写基于上述结构的相关操作函数(简述如下)：

(1)OSPriCeilingCreate ()：用于创建优先级天花板协议，它从空闲事件链表中摘下一个事件控制块结构，并将此结构中的 OSEventPCP 初始化成用户指定的 PCP，并返回事件控制块指针给任务。

(2)OSPriCeilingDel ()：用于删除优先级天花板协议，其主要操作就是删除一个优先级天花板协议的实例，并回收事件控制块到空闲链表中。

(3)OSPriCeilingPend()：用于申请资源。当任务需要使用一个临界资源时，通过 OSPriCeilingPend()函数申请该资源。

(4)OSPriCeilingPost()：用于释放已取得的临界资源。此函数基本上执行和 OSPriCeilingPend () 函数相反的操作。但是这个函数和 OSPriCeilingPend() 函数最大区别为：在函数结束前需要进行一次任务调度，以使高优先级的任务有机会得到处理器时间。

此外，还有一个需要修改的 μC/OS-II 已有内核函数 OSTaskChangePrio()，因为改进型优先级天花板协议假定任务的优先级是固定的，而 μC/OS-II 允许任务调用 OSTaskChangePrio() 来改变自己或其他任务的优先级，所以需要修改此函数，使其能够判断用户是否在内核配置文件 OS_CFG. H 声明了要使用改进型优先级天花板协议。如果已声明，就阻止其改变优先级并返回错误信息；否则，则继续其原来函数流程的执行。

4 结束语

针对优先级反转问题，本文给出了改进型优先级天花板协议，比 μC/OS-II 固有的优先级继承协议具有预防系统死锁等特点，同时具有更小、更确定的高优先级任务阻塞时间，从而最大程度地保证了系统的实时性。

参考文献

[1] Lbrosse J J. 嵌入式实时操作系统 μC/OS-II[M]. 邵贝贝, 译. 北京: 北京航空航天大学出版社, 2005: 44-46.

[2] 刘智臣, 孟益民. 嵌入式操作系统 μC/OS-II 中优先级反转问题及其解决方案[J]. 科学技术与工程, 2005, 5(1): 23-27, 32.

[3] Burns A, Wellings A. 实时系统与编程语言[M]. 王振宇, 陈 利, 译. 北京: 机械工业出版社, 2004-04: 382-386.

[4] 罗 蕾. 嵌入式实时操作系统与应用开发[M]. 北京: 北京航空航天大学出版社, 2005: 168-179.

[5] 彭守凡, 包 剑, 冀常鹏. 嵌入式 Linux 中就绪任务优先级反转问题研究[J]. 微计算机信息, 2004, 20(8): 51-52.

[6] 林 游, 韩志科. 在μC/OS-II 上实现优先级天花板[J]. 单片机与嵌入式系统应用, 2005, 5(4): 78-80.

（上接第 65 页）

4 HIUA 算法性能分析

更新关联规则，一种直接的方法是重新运行 Apriori 算法。现在将关联规则更新算法 HIUA 与重新运行一遍 Apriori 算法进行性能比较。在数据库开发工具 Delphi 7.0 中编程实现以上算法，交易数据库由数据项集{*A*, *B*,…,*Z*}中以随机方式产生任意长度(1-6)的数据项的 100 000 条交易记录组成。

在最小支持度发生变化的更新算法中，最小支持度分别取 2%,0.5%,0.25%,0.2%,0.15%,0.1%进行变化，2 种算法在各种最小支持度下对应生成的频繁项集是完全相同的。表 1 显示了测试交易数据库的实验结果。

表 1　2 种算法实验结果的比较

频繁项目集	最小支持度/(%)	Apriori 算法运行时间	HIUA 算法/(%)					
			2	0.5	0.25	0.2	0.15	0.1
26	2	3′47″	-	0	0	0	0	0
28	0.5	3′47″	2′25″	-	0	0	0	0
133	0.25	5′21″	1′15″	1′12″	-	0	0	0
191	0.2	7′12″	3′13″	3′06″	2′05″	-	0	0
307	0.15	10′02″	5′54″	5′53″	4′53″	3′21″	-	0
488	0.1	15′32″	10′47″	10′29″	9′40″	8′56″	6′12″	-

5 算法分析

改进后的算法 HIUA 克服了 IUA 算法存在的遗漏频繁项目集的问题，在运行效率上也较 IUA 有了提高。IUA 算法在生成 *Ck*3 时，要通过 *Lj*1 和 *Lk*-*j*2，中的项目集拼接而成，*j* 从 1 迭代到 *k*-1，这样做的结果是在拼接阶段产生一些无用候选频繁项目集，而此算法充分利用了任一频繁 *k* 项目集的 *k*-1 子项目集一定是频繁项目集的性质，避免了一些无用的候选项目集的产生。

从表 1 可以看出，HIUA 算法挖掘的频繁项集完整并且在更新关联规则的意义上性能要优于 Apriori, IUA 算法。

参考文献

[1] Jiawei H, Micheline K. 数据挖掘概念与技术[M]. 范 明, 孟小峰, 译. 北京: 机械工业出版社, 2001: 149-183.

[2] 数据挖掘讨论组数据挖掘资料汇编[Z]. (2006-06-02). http://www.dmgroup.org.cn/zongshu050322/zs20.htm.

[3] 史忠植. 知识发现[M]. 北京: 清华大学出版社, 2002: 2.

[4] Fayyad U, Smyth P, Uthurusamy R. Advances in Knowledge Discovery and Data Mining[M]. Canbridge, MA: MIT Press, 1996: 1-34.

[5] 冯玉才, 冯剑琳. 关联规则的增量式更新算法[J]. 软件学报, 1998, 9(4): 301-306.

[6] 黄明新, 刘椿年. 基于归纳逻辑程序设计的特异规则挖掘[J]. 北京工业大学学报, 2003, 29(4): 495-499.

[7] 毛国君, 刘椿年. 基于项目序列集操作的关联规则挖掘算法[J]. 计算机学报, 2002, 25(4): 17-22.