文章编号: 1671－4598(2004)06－0578－03　　中图分类号: TP273　　文献标识码: B

嵌入式实时操作系统任务的同步与互斥机制

王晓春, 刘兴东
(深圳职业技术学院 计算机系, 广东 深圳　518055)

摘要: 文章对嵌入式实时操作系统任务管理的特点做了分析, 并侧重论述了嵌入式实时操作系统任务管理的特殊性, 尤其是对优先反转问题做了剖析, 并指出相应的处理机制和方法, 尤其是对应用较广的优先级继承算法的问题做了分析, 得出其最大的缺陷是隐含有死锁问题。经过对比, 得出优先级天花板算法更为可靠的结论。

关键词: 实时操作系统; 任务; 同步; 互斥

Characters of Synchronization and Mutual Exclusion in RTOS Tasks Management

Wang Xiaochun, Liu Xingdong
(Department of Computer Engineering, Shenzhen Polytechnic, Shenzhen　518055, China)

Abstract: The characters of the Embedded Real－Time operating system in task management is analyzed. The embedded teal－time operating system in task management, the corresponding of the managing mechanism and method are described. The drawback of the priority inheritance protocol is analyzed and reaches the conclusion that the priority ceiling protocol is more reliable.

Key words: real－time operating system; task; synchronization; mutual exclusion

0　引言

随着后 PC 时代的到来, 嵌入式实时操作系统(RTOS)发展迅速。嵌入式实时操作系统与桌面计算机常用的通用操作系统(GPOS)有所不同, GPOS 追求的是最大限度的利用硬件资源, 使系统的信息吞吐量达到最大。而 RTOS 注重的是系统实时性和并行性, 即系统能够在规定时间内对外部事件做出响应的同时又能同时完成多种任务的操作。实现系统的实时性和并行性, 多任务之间的调度是个关键问题。基于 RTOS 对实时性、多任务的要求, 对任务的管理也有其特有的方法, 在任务间的同步与互斥机制方面也有其不同的方式和特点。

1　嵌入式实时操作系统中任务调度的特殊性

1.1　实时确定性

对于一个实时嵌入式操作系统而言, 它与桌面操作系统的本质区别在于它的实时确定性。一般来讲, 桌面操作系统对实时性要求不高, 而嵌入式操作系统要求较高的实时性, 即系统的执行时间即使在最不利的情况下也应该可以预测。在设计一个实时嵌入式操作系统时, 所有任务需执行的时间都是可以事先预测的, 这是因为实时系统的正确性不仅依赖于系统计算的逻辑结果, 还依赖于产生这个结果的时间。为了保证这一点, 嵌入式操作系统的内核应该有相应的机制来实现实时性。在嵌入式操作系统中常采用的基于优先级的可抢占调度、不采用虚拟存储器以及任务对资源的有限等待等技术, 都是保证实时确定性的重要手段。

1.2　调度策略

在实时操作系统中, 一般采用基于优先级的抢占式调度法, 程序员为系统中的每一个任务定义一个优先级别。当调度程序选择下一个任务运行时, 它将选择处于任务就绪队列中优先级最高的任务, 一个时间片运行结束, 如果该任务比就绪队列上的其它任务的优先级高, 该任务将获得下一个时间片, 继续占有 CPU; 如果就绪队列上有比该任务优先级更高的任务出现, 优先级更高的任务将获得下一个时间片; 如果处于任务就绪队列中的任务具有与该任务相同的优先权, 则调度程序采用时间片轮转方法来调度任务的执行。优先级调度算法是嵌入式操作系统任务调度的的重要特征, 保证优先权最高的任务总能得到最先执行。实际上, 由于嵌入式系统应用环境的多样性及特殊性, 这样的调度策略才能保证程序员在任何情况下都始终知道应用程序中的哪个任务将被运行。

1.3　优先级反转问题

当然采用以优先级为主的调度策略也存在有隐患, 这就是优先级的反转问题[1]。出现优先级反转时, 系统至少有 3 个以上的任务处于就绪状态。假如系统中有三个任务 task1, task2, task3 就绪, 假设 task1 的优先权最高, task2 次之, task3 最低。当任务 task1 和 task2 因为请求资源不果被阻塞后, task3 得到运行, task3 在运

收稿日期: 2004－03－24

作者简介: 王晓春(1964－), 男, 内蒙呼和浩特市人, 硕士, 主要从事嵌入式系统的教学和科研工作。

行过程中占用了一个共享资源，正在执行时，task1 和 task2 都获得资源，请求调度运行，由于 task1 的优先权最高，task1 被调度执行，在执行时，task1 发现运行需要的另一个资源已被 task3 占用，task1 又一次被阻塞，从而 task2 获得调度执行。这时就出现了优先权反转的情况，最高优先权的任务 task1 得不到运行，而一个非最高优先权的任务 task2 却像一个最高优先权的任务在运行。

优先级反转会产生一种不确定的延迟，在这个例子中，最高优先权的 task1 必须等 task3 运行完成，释放资源后才能继续运行，但由于 task3 的级别最低，它又被 task2 所抢先，从而造成最低优先权的 task3 以一个不确定的时间阻塞最高优先权的 task1 的执行，task2 却从中获利，享受着最高优先权的待遇。

由于优先级反转会导致嵌入式系统对任务调度的不可预见，从而破坏系统的实时性和确定性。

1.4　死锁

死锁是任务之间因为申请共享资源而导致的任务无限期的等待现象，其最直接的现象就是死机。桌面操作系统中可以采用避免死锁的算法，如检测并拒绝可能引起死锁的资源请求，并通过死锁检测算法终止一个或多个任务的执行来解除死锁。在桌面操作系统中，即使死机，也允许人工干预使其复位或重新开机，令程序重新运行。但在嵌入式系统中，不仅死机是不允的，采用上述解决死锁的算法也是不可行的，因为这些算法不仅增加系统时间的额外支出，而且由于嵌入式应用的特殊性，也不允许随意终止任务来解除死锁，因为这样做可能导致不可预料的情况发生。所以在嵌入式系统中，不允许发生死锁。

2　嵌入式实时操作系统中任务的同步与互斥

在多任务系统中，多个任务之间相互等待共同推进，这是任务间的同步；如何协调任务间对共享资源的争夺，则是任务间的互斥。嵌入式系统中任务间的同步与互斥与桌面操作系统相比有不同的特点。

2.1　关中断

关中断是任务之间实现互斥的最强壮的方法，方法简单却高效。但由于关中断期间，CPU 不能再响应任何中断，因此这种方法要尽量少用。尤其是在嵌入式系统中，关中断可能会直接影响系统实时响应的能力，一般在对互斥资源访问时间极短的情况下，可以使用这种方法。

2.2　信号量

信号量是实现任务间同步与互斥的机制。信号量一般分为二种：二值信号量和计数信号量。二值信号量也称为二进制信号量，其值只能是 0 和 1，而计数信号量取值范围则很宽，根据系统的不同，可能在 0－255 或 0－65535 之间。

当程序在创建一个计数信号量时，它将赋予该计数信号量一个正的初始值，这个值表示了该信号量所指向资源的数量，意即该资源同时最多可以为多少个任务提供服务。每当一个任务请求该信号量时，系统首先检查该信号量的计数是否大于 0，如果大于 0，说明资源尚未用完，调用成功，同时计数值减 1；如果小于 0，则说明已没有资源，任务即被阻塞。当任务完成对一个信号量的占用后，该任务就发出一个释放调用，并使该信号量计数值增 1。

2.3　任务的同步

两个任务间的同步，常常采用两个信号量来完成。当两个任务在各自不同的节点上需要与对方同步时，双方通过相应的信号量来实现同步。例如当两个任务正在执行，在第一个任务到达某个节点时会通过一个信号量通知第二个任务已经到达同步点，并将自己阻塞，等待第二个任务返回一个信号，第二个任务到达相应的同步点时，也会通过一个信号量发出一个信号给第一个任务，告诉对方已经到达同步点，两个任务将在各自的同步点上协调同步执行。

2.4　任务的互斥

当任务为争夺共享资源时，出现任务间的互斥。在多任务系统中，这是很常见的现象。当任务间出现互斥时，就可能出现优先级反转问题，解决这个问题常用的手段是：优先级继承和优先级天花板算法。

2.4.1　优先级继承算法

优先级继承算法[3]的基本方法是，一个较高优先级的任务申请某信号量时，如果该信号量已被一较低优先级的任务占有，则系统将抬升较低优先级任务的优先级，使其等同与较高优先级任务的优先级别，从而保证其尽快执行完成，释放较高优先级任务所需的资源。例如，在上面的例子中，task1 被 task3 所阻塞，系统提升 task3 的优先级别，使其等同于 task1 的优先级别，从而 task3 可以优先 task2 获得执行，task3 执行完毕后，释放 task1 所需的资源，从而保证 task1 获得执行。一旦某任务的优先级被抬升，直至此任务释放完它所占有的所有互斥信号量，其优先级才被还原。

但优先级继承算法可能会导致系统的死锁。还是以上述的例子来分析，当 task1 和 task3 之间出现竞争两个二值信号量 S_1 和 S_2 时，就有可能会出现死锁。假设 task1 已经得到了信号量 S_1，task3 已经得到了信号量 S_2，当 task1 申请信号量 S_2 时，发现 S_2 已经被任务 task3 所占用，为了防止优先级反转，使 task2 得到优先运行，系统提升 task3 的优先权，使其与 task1 的优先权相同，以加快 task3 的完成，但 task3 继续运行时发现，其运行所需要的信号量 S_1 已经被 task1 所占用，task3 继续运行的条件得不到满足，从而使任务 task1 和 task3 陷入死锁。

2.4.2 优先级天花板算法

优先级天花板算法的基本方法是，首先给出可能申请某信号量的所有任务中最高优先级任务的优先级，将其作为天花板。此后，一旦有任务申请此信号量，如果它的优先级高于天花板，则出错。否则，将其优先级抬升到天花板。直至此任务释放完它所占有的所有互斥信号量，其优先级才被还原，这个算法同样可以解决优先级的反转问题。

优先级天花板算法可以避免像优先级继承算法中的死锁问题。这一点可以通过下面的论证得到：

假设有三个任务 task1、task2 和 task3，其中 task1 的优先权最高，task3 的优先权最低。任务 task3 目前正在占用信号量 S_2，当被高优先级的任务 task1 抢占时，则有：

任务 task1 只有满足以下条件时，才会抢占 task3 获得 CPU 以得到运行：

$$P(\text{task1}) > \text{ceil}(S_2) \tag{1}$$

其中 P（task1）来表示任务 task1 的优先级，ceil（S_2）为信号量 S_2 的优先级天花板。

任务 task2 只有当申请信号量 S_2 的某一个任务 task 受到阻塞时才会升级为天花板级的优先权，这时应该有

$$\text{ceil}(S_2) \geqslant P(\text{task}) \tag{2}$$

比较以上两个公式，可以得出：

$$P(\text{task1}) > P(T) \tag{3}$$

即表示对于任务 task3 来讲，在 task1 执行完成以前，task3 不可能继承获得与 task1 相同甚至更高的优先级。

在构成死锁的四个必要条件中，其中一个条件是循环等待，即任务 task1 正在等待任务 task2 占用的资源，task2 正在等待任务 task3 占用的资源……，任务 taskn 正在等待 task1 占用的资源。根据公式（3）可以得知，在优先级天花板算法中，不会出现三个以上的任务阻塞传递的情况。因为一旦出现任务 task3 阻塞任务 task2，进而阻塞 task1 时，task3 将继承 task1 并升级为该信号量的优先级天花板，这与前面得出的结论是相违背的，所以采用优先级天花板算法的任务不会出现信号量的循环等待，也就不会出现死锁。

比较以上这两种算法，各有优缺点，优先级继承法简单，但可能需多次改变占有某资源的任务的优先级，还可能造成死锁；优先级天花板虽只需改变一次，但要统计申请某个特定信号量的所有任务中，哪个优先级最高，也非易事。在实际的嵌入式系统中，优先级继承法用得更多一些。

使用优先级调度策略的二值信号量可以采用优先级继承和优先级天花板算法。

在嵌入式系统中，因为申请信号量失败而被阻塞的任务，也将进入等待队列，但系统会提供以下方法供任务选择：

（1）缺省情况下，任务进入等待队列等待；

（2）任务立即返回，并返回一个错误状态码，指示资源忙；

（3）指定任务等待时限，在指定时限内未获得资源，任务返回，并返回一个错误状态码，指示超时。

其中的处理方法体现了嵌入式系统中对时间的确定性要求，实际上也就是避免了死锁现象的发生。如果在一个时间范围内，任务得不到资源，系统则即时返回，并通知任务已经超时，申请该资源的任务则可以重新开始执行，从而保证程序员对整个系统状态的把握。

2.5 事件

事件提供一种任务间复杂的同步机制。可用于任务之间、任务与中断服务程序之间的同步。而且可实现一个任务与多个任务或中断服务程序的同步。事件用于协调任务间的处理步伐，但并不用于控制对共享资源的访问。一个事件标记被任务用来通知其它任务，系统出现一个预先定义的事件，从而实现任务间的同步。在具体的应用系统中，一个接口的任务可能就需要外部一个事件的触发，完成一次数据转换。应该注意的是：（1）事件只与任务相关；（2）任务可以同时等待多个任务；（3）事件间相互独立；（4）事件不提供数据传输功能；（5）事件无队列，即多次向任务发送同一事件，其效果等同于只发送一次。

3 结束语

在嵌入式系统应用中，任务的调度直接关系到嵌入式系统的性能表现，而任务间的同步与互斥问题也会影响系统的响应能力，所以在程序设计上要充分考虑到协调任务间的关系，使其能够在资源利用上做出取舍，以保证系统最佳性能的体现，在处理优先级反转问题时，虽然优先级继承算法简单，但隐含有死锁问题，值得引起重视，反而是优先级天花板方法，虽然稍微复杂，但避免了优先级继承的缺陷，是一个可靠的算法。

参考文献：

[1] Labrosse J J. 嵌入式系统构件 [M]. 袁勤勇，等译. 北京：机械工业出版社，2002.

[2] Berger A. 嵌入式系统设计 [M]. 吕俊，译. 北京：电子工业出版社，2002.

[3] Grehan R. 32 位嵌入系统编程 [M]. 许汝峰，译. 北京：中国电力出版社，2001.

[4] Krishna C M. Real—time Systems [M]. 北京：清华大学出版社，2001.