第 35 卷 第 7 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.7 Computer Engineering April 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)07—0259—02 文献标识码：A 中图分类号：TP316.2

uC/OS-II 内核超时等待机制的分析与改进

韩明峰，李小滨，郑永志
(烟台东方电子信息产业股份有限公司，烟台 264000)

摘 要：分析 uC/OS-II 内核的超时等待机制，证实在一定情况下超时时间间隔不准确，在时间间隔到期的情况下，内核仍有可能返回成功，不符合一般的操作系统原理，且超时等待机制在有附加的挂起操作时，处理效率低下。指出 uC/OS-II 内核的超时等待机制不完善的原因，对不足之处提出改进方法。

关键词：超时等待；资源；挂起；内核

Analysis and Improvement of Waiting-timeout Mechanism in uC/OS-II Kernel

HAN Ming-feng, LI Xiao-bin, ZHENG Yong-zhi
(Yantai Dongfang Electronic Information Industry Co. Ltd., Yantai 264000)

【**Abstract**】This paper analyzes the waiting-timeout mechanism of uC/OS-II kernel from source code. It indicates waiting-timeout of uC/OS-II is not correst in some cases. The kernel can return success while it is time out. System efficiency is low if kernel waiting-timeout mechanism has additional suspension operation. The reason why waiting-timeout mechanism of uC/OS-II kernel is not perfect is analyzed. A method is advanced to resolve these problems.

【**Key words**】waiting-timeout; resource; suspend; kernel

uC/OS-II 是专为嵌入式应用设计的源码公开的嵌入式实时内核[1]，可用于各类 8 位、16 位、32 位单片机或 DSP，已被移植到各种类型的芯片上。目前关于 uC/OS-II 内核机制的论述主要集中于优先级反转，在基于优先级的基础上引入时间片调度策略、扩充任务数[2-3]等技术。作为嵌入式实时系统的基础性软件，嵌入式实时内核的安全性、可靠性至关重要，否则可能会给应用系统带来巨大损失[4]。本文剖析 uC/OS-II 内核超时等待机制，结合实例验证，指出 uC/OS-II 内核的超时等待机制在某些情况下存在不正确性、低效性，并提出相应的解决方法。

1 uC/OS-II内核超时等待机制的分析

1.1 内核超时等待机制的基本原理

内核的定时机制主要包括以下几个方面：

(1)任务时间片，主要用于时间片轮转调度算法。

(2)应用定时器，主要用于应用程序设置的各种定时器。

(3)任务延时，指任务主动睡眠一段时间。

(4)超时等待资源，在任务获取资源不能满足时，可以选择超时等待。

uC/OS-II 内核 V2.52 的定时机制只涉及(3)和(4)，其中，(4)是本文论述的重点。在实际使用中不管基于何种操作系统平台，应用程序经常会等待一些系统资源，如信号量、事件标志、消息、邮箱等。等待类型共有 3 种：

(1)不管是否能获取资源，马上返回，不会等待。以消息队列为例，uC/OS-II 内核单独定义了一个系统调用 *OSQAccept*()用于实现此功能。

(2)如果不能马上获取资源，将进行有限时间的等待，即超时等待。以消息队列为例，uC/OS-II 内核定义了一个系统调用 *OSQPend*()，当入口参数 *timeout*>0 时用于实现此功能。

(3)如果不能马上获取，一直悬挂等待。调用 *OSQPend*()，当入口参数 *timeout*=0 时用于实现此功能。

应用程序通过操作系统提供的系统调用接口按超时等待获取资源时，在系统调用的入口参数里可以指定超时等待的最大时间，通常以 ms 为单位，内核会将其转化为系统的时钟滴嗒数(tick)，一般内核都会执行以下流程：

(1)如果资源能马上获取，系统调用将成功返回。

(2)如果资源不能马上获取，内核将设置一定时器进行计时，把当前任务悬挂在该资源的等待队列上，将该任务从就绪表中删除，并进行调度，让出 CPU 的使用权。

(3)如果在指定的时间内资源变得可以获取了，定时器应马上停止计时，该任务从等待队列里摘下并且重新回到就绪表中等候调度。

(4)如果定时器到时，任务应该从资源等待队列里摘下并且重新回到就绪表中，系统调用返回超时信息。

内核在每一个 tick 都会做一系列工作，包括任务的延迟以及超时等待资源的定时器等相关的检查操作。严格来说，在指定的时间间隔以外到达的资源和信号被认为是无效的，这也是指定超时时间间隔的意义所在。某些对时间要求苛刻的场合就有这种需求，内核必须精确地处理好这方面的问题。

1.2 uC/OS-II内核超时等待不准确性的分析

假设消息队列 *Q* 里没有消息，某任务 *T* 通过系统调用 *OSQPend*()超时等待消息资源 *Q*。根据 *OSQPend*()的实现流

作者简介：韩明峰(1973－)，男，高级工程师、硕士，主研方向：实时嵌入式操作系统，变电站自动化系统；李小滨，博士；郑永志，高级工程师

收稿日期：2008-09-17 **E-mail：**hanmingfeng@dongfang-china.com

程，将执行以下操作：

(1)内核将在任务 *T* 的任务控制块中设置任务等待消息标志 *OS_STAT_Q* 和超时等待时间 *OSTCBDly*；

(2)调用 *OS_EventTaskWait*()，将消息队列控制块指针存放在任务 *T* 的任务控制块中，将任务 *T* 移出系统就绪表、移入到消息队列 *Q* 的任务等待表中；

(3)调用 *OS_Sched*()进行任务切换。

将任务 *T* 唤醒有 2 种途径：超时后由系统时钟节拍函数 *OSTimeTick*()来唤醒；通过消息发送函数，如 *OSQPost*()。本文考虑超时状态下的情况。

时钟中断服务程序在每一个时钟中断处理中调用，其主要功能是遍历系统任务链表，检查任务控制块的延时项 *OSTCBDly*，如果 *OSTCBDly* 不为 0，则进行减 1 操作。如果任务 *T* 的定时时间间隔到期(延迟项被减为 0)，并且任务 *T* 没有附加的挂起操作，任务 *T* 就会进入就绪表，然而该函数却没有进一步将任务 *T* 移出资源 *Q* 的等待队列，也就是说此时任务 *T* 跨了 2 个状态，同时处于就绪表和相应资源的等待表中。这 2 个状态从本质上讲是矛盾的。当 *OSTimeTick*()函数退出后，系统将在函数 *OSIntExit*()中进行一次任务调度。虽然任务 *T* 此时处于就绪表中，但未必马上就能获得执行权，这取决于任务 *T* 是否是就绪表中的最高优先级。

OSQPost()在消息队列 *Q* 的等待表中有等待任务的情况下，调用 *OS_EventTaskRdy*()，将等待表中最高优先级的任务从等待表中删除，将消息指针直接复制到等待表中最高优先级任务的控制块数据项 *OSTCBMsg* 中，删除等待状态 *OS_STAT_Q*，如果没有附加的挂起状态，任务 *T* 进入就绪表。如果等待表中的最高优先级任务就是前面讲的等待超时的任务 *T*，这相当于任务 *T* 又一次进入就绪表，不过只有一次从等待表中删除。任务 *T* 获得执行权以后从调度程序 *OS_Sched*()返回将检查任务控制块的 *OSTCBMsg* 数据项，此数据项不为空，任务 *T* 按获取到资源对待，只不过是在超时时间以外获取到的，其时间关系如图 1 所示。

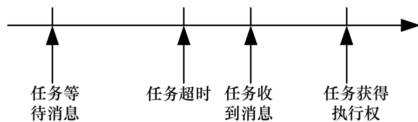

图 1　超时等待时间轴

如果任务 *T* 由于超时进入就绪态，到 *T* 获得执行权之前，仍没有获取到资源 *Q*，*OSQPend*()将调用函数 *OSEventTo*()，此时任务 *T* 才被从等待表中删除，任务状态被置为就绪态 *OS_STAT_RDY*，最后返回超时状态。

1.3　uC/OS-II内核超时等待在有挂起操作时的特点分析

有挂起操作的超时等待时间轴如图 2 所示。假设消息队列 *Q* 里没有消息，某任务 *T* 通过系统调用 *OSQPend*()超时等待消息资源 *Q*，在超时等待期间任务 *T* 有一个挂起操作 *OSTaskSuspend*()，超时后任务 *T* 也没有获取到消息，但超时后任务 *T* 又有一个恢复操作 *OSTaskResume*()。

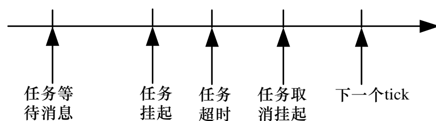

图 2　有挂起操作的超时等待时间轴

时钟节拍函数 *OSTimeTick*()有 1 个特点：在任务控制块的延时项 *OSTCBDly* 递减为 0 并且有挂起状态 *OS_STAT_SUSPEND* 时，要把 *OSTCBDly* 赋值为 1。在任务 *T* 被执行恢复操作 *OSTaskResume*()之前，每一个时钟节拍中都要重复地做判断、递减、赋值，浪费了 CPU 资源。在任务 *T* 被执行恢复操作 *OSTaskResume*()时，按 *OSTaskResume*()的处理流程，当满足下列 3 个条件时任务 *T* 将被移入到就绪表中：(1)内核检查任务控制块数据项 *OSTCBStat*，如果状态中含有挂起状态 *OS_STAT_SUSPEND*；(2)没有其他的等待资源状态，如 *OS_STAT_Q* 等；(3)任务控制块数据项 *OSTCBDly* 为 0。在该流程中，目前任务 *T* 不满足条件(2)和条件(3)，因此，任务 *T* 尽管从实质上讲超时已到期，挂起已取消，但还不能进入就绪表。

当下一个时钟节拍到来时，*OSTimeTick*()将任务 *T* 控制块的延时项 *OSTCBDly* 递减为 0，此时挂起状态 *OS_STAT_SUSPEND* 已被清除，任务 *T* 终于被移入就绪表。其内核代码注释中也已指出：延时项 *OSTCBDly* 赋值为 1 是为了在取消挂起时防止任务不能被唤醒。的确如此，如果延时项 *OSTCBDly* 递减为 0 时保持为 0，因为任务等待资源状态 *OS_STAT_Q* 没被清除，所以 *OSTaskResume*()不会唤醒任务 *T*；又因为 *OSTimeTick*()对延时项 *OSTCBDly* 为 0 的任务不予处理，所以任务 *T* 也不会被 *OSTimeTick*()唤醒。

内核的这种处理方式尽管保证了结果的正确性，但却是近似的，因为延迟了一个 tick，并且效率低下。

2　uC/OS-II内核超时等待机制的改进

上文论述了 uC/OS-II 内核超时等待机制在某些情况下的不正确性，其根本性的原因是超时到期将任务移入到就绪表时没有同时将该任务从它所在的资源等待表中移走，从而导致超时以后再度获取资源的可能。

uC/OS-II 内核需要在时钟节拍函数 *OSTimeTick*()里增加代码解决这一问题，将延时期满的任务从相应的资源等待列表中删除。内核任务控制块有指向该任务所等待的消息、邮箱等事件控制块的指针 *OSTCBEventPtr*，事件控制块里有相应的等待表。对于 uC/OS-II 新引进的事件标志组，任务控制块有指向相应的等待节点的指针，等待节点有指向相应的事件标志组控制块的指针，删除一个等待节点也能实现。代码改进描述如下：

```
void    OSTimeTick (void)
{
#if OS_CRITICAL_METHOD == 3
OS_CPU_SR    cpu_sr;
#endif
OS_TCB        *ptcb;

OSTimeTickHook();
#if    OS_TIME_GET_SET_EN > 0
      OS_ENTER_CRITICAL();
      OSTime++;
      OS_EXIT_CRITICAL();
#endif
      if (OSRunning == TRUE) {
            ptcb = OSTCBList;
            while(ptcb->OSTCBPrio!=OS_IDLE_PRIO) {
              OS_ENTER_CRITICAL();
                if (ptcb->OSTCBDly != 0) {
```

（下转第 266 页）