文章编号:1671-4598(2018)07-0203-04 DOI:10.16526/j.cnki.11—4762/tp.2018.07.045 中图分类号:TP302 文献标识码:A

支持双重触发的μC/OS—II内核的设计与实现

黄姝娟[1],朱怡安[2],刘白林[1],肖 锋[1]
(1.西安工业大学 计算机科学与工程学院,西安 710021;2.西北工业大学 计算机学院,西安 710072)

摘要:针对当前大多数嵌入式操作系统不能同时支持时间和事件双重触发的机制,对μC/OS—II内核进行了深入研究,对μC/OS—II的调度代码进行了分析,提出了将系统内核架构设计为上下两层,以时间触发部分为上层主要模块,事件触发部分为下层基础模块的层次性架构;从而使得系统核心的调度器既可以调度TT任务也可以调度ET任务;实验证明,该方法不仅能够支持时间和事件双重触发的任务调度,而且在不影响可靠性和确定性的情况下,提高了系统的灵活性和实时性。

关键词:嵌入式;操作系统;时间触发;事件触发;调度器

Design and Implementation of μC/OS—II Kernel for Double—Triggered Mechanism

Huang Shujuan[1], Zhu Yi'an[2], Liu Bailin[1], Xiao Feng[1]
(1. School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710021, China;
2. School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: At present, most of the embedded operating system cannot support both time—triggered and event—triggered mechanism. This paper analyzed the kernel of μC/OS—II in deeply and the scheduling code , designed the kernel architecture into two layers, the upper layer is the time—trigged module and the lower is the event—trigged module. The time—triggered partition is the main processing section and the event—trigged partition is the base module section. This make the scheduler of the system not only dispatching the time—triggered tasks but also scheduling the event—trigged tasks. Experiments show that this method can not only support the double—triggered mechanism forμC/OS—II , but also improve the real—time performance and flexibility without affecting the reliability and certainty of the system.

Keywords: embedded system; multi—core; scheduling algorithm; scheduling model; real—time tasks

0 引言

在实时操作系统中,事件触发指一个任务只有在与之相关的特定事件发生的条件下才能开始运行。早在最初的嵌入式领域中,事件触发机制已经得到了广泛应用[1],然而在航空、航天及核电等领域,由于系统的高可靠性与硬实时性要求,事件触发方式无论在设计,还是维护方面都存在较大困难[2]。许多任务可能因为等待资源而超时,这样就无法满足任务的实时性要求,更无法保证安全关键任务的完成。为此,很多基于事件触发的操作系统变得愈发复杂,不仅降低了研发效率,而且增加了维护成本[3]。

时间触发机制指在预先计划好的时间点执行系统行为,从而使系统具有良好的先验性和确定性[4]。时间触发设计思想中时间可控的特点从根本上提高了系统的实时性,避免了可能的直接冲突。因此,对于任务确定的嵌入式系统,时间触发设计思想具有明显优势[5-7]。然而单纯采用时间触发的系统在某种程度上会降低系统的灵活性和动态交互性,很难完全满足多样性的系统需求,尤其针对混合关键任务的执行[8-10]。因此,在实际应用中,往往需要将时间触发和事件触发机制相结合,设计出一种支持时间触发和事件触发的混合调度机制。

由于μC/OS—II是一款仅支持事件触发的开源、可移植、可裁剪的的实时操作系统,通过分析该系统的特点,将其改造为支持两种触发机制的操作系统。

1 μC/OS—II操作系统内核分析

μC/OS—II操作系统中的任务由三个部分组成:任务程序代码,任务堆栈和任务控制块。

任务程序代码通常被设计为一个无限循环结构。在这个循环中可以响应中断。任务堆栈是给每个任务分配的一

收稿日期:2017-11-15; **修回日期**:2017-12-15。
基金项目:国家自然科学基金面上项目(61572392),陕西省工业科技攻关项目(2015GY031),民用飞机专项科研项目(MJ—2015—D—066)。
作者简介:黄姝娟(1975-),女,博士,讲师,主要从事嵌入式与分布式计算方向的研究。
朱怡安(1975-),男,教授,博士研究生导师,主要从事高性能计算、云计算和普适计算方向的研究。
刘白林(1975-),男,副教授,硕士研究生导师,主要从事人工智能与故障诊断方向的研究。
肖 锋(1976-),男,教授,计算机学会高级会员,主要从事智能信息处理方向的研究。

块独立的连续内存空间，用来在任务切换和响应中断时保存 CPU 寄存器中的内容，或任务调用其他函数时使用。任务控制块（Task Control Block，TCB）的数据结构 OS _ TCB，用来记录任务的堆栈指针、任务的当前状态、任务的优先级等一系列与任务管理有关的属性。

任务调度主要包括两部分内容：一是判断哪些任务处于就绪状态；二是从处于就绪态的任务中确定应该马上执行的任务并为其分配 CPU。μC/OS－II 中完成第一部分工作的是任务就绪表，完成第二部分工作的是调度器。μC/OS－II 中设计了一个位图来登记系统中所有处于就绪状态的任务，这个位图就是任务就绪表。位图的每一位代表系统中的每一个任务，该位置的状态（1 或 0）就表示对应任务是否处于就绪状态。

2 内核设计

为了能够同时支持 TT（Time－Triggered）任务和 ET（Event－Triggered）任务，将系统分为两大部分：时间触发模块和事件触发模块，这两部分相对独立又紧密联系。以时间触发模块为上层主要模块，事件触发模块为下层基础模块的层次性架构如图 1 所示。

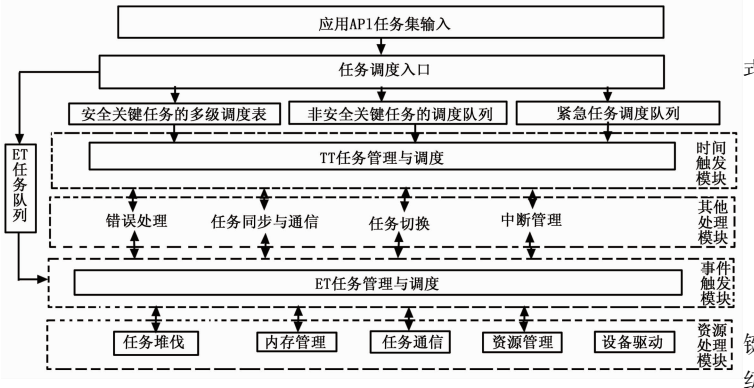

图 1 时间和事件双重触发的系统调度架构图

2.1 应用 API 任务集输入

应用 API 任务集输入位于整个内核架构的最外层，包括对应用提供的系统服务、系统配置和用户应用程序。系统配置模块可以配置与具体应用相关的参数和选项等，用户应用程序即包括用户创建的任务，其中有时间触发模块提供的 API，也有事件触发模块提供的 API。

2.2 任务调度入口

任务调度入口将应用 API 分为时间触发任务和事件触发任务。在调度器内核中，时间触发模块为 TT 任务的管理和调度服务，事件触发模块为 ET 任务进行管理和服务，该模块仍然采用原 μC/OS－II 中的管理方法。

2.3 TT 任务管理模块

在 TT 任务调度中，系统任务仅由预设的时间点来触发，调度器在运行期间每一时刻，都是通过提前计算好的调度表来进行调度决策的。

2.4 ET 任务管理模块

事件触发模块主要负责 ET 任务的管理、调度以及与安全性相关的访问控制决策和实施。此外，还包括中断管理和任务间通信等内核与硬件无关的部分。

2.5 资源处理模块

包括与具体硬件平台相关的任务堆栈管理、任务切换、时钟中断等模块，根据不同的应用环境进行个性化移植。

整个内核中，时间触发模块占主体地位，系统为其预先分配固定的时间槽用以执行时间触发任务，主要保证系统的可靠性和确定性。事件触发模块在时间触发模块的空闲时间内处理，而空闲时间也可以根据时间触发任务的实际运行情况动态调整，提高了系统的灵活性。

3 内核实现

3.1 任务管理

时间触发模块主要完成 TT 任务的管理与调度、任务间的通信以及中断管理等。为了更好的保留 μC/OS－II 的原有特性和代码结构，时间触发模块的实现是在由改造 μC/OS－II 实现的事件触发模块之上，另外新建的一个模块。其结构定义如下：

```
typedefstructos_tt_tcb
{OS_STK * OSTT_TCBStkPtr;  /* TT 任务的堆栈指针 */
    ttTaskTypeOSTT_TCBTaskID; /* 任务标识符 */
    ttAppModeTypeOSTT_TCBAppMode; /* 任务所属应用模式 */
    INT8U OSTT_TCBStartTime;  /* TT 任务的发布时间 */
    INT8U OSTT_TCBDeadline;  /* TT 任务的时限时间 */
    ttTaskStateTypeOSTT_TCBState;  /* TT 任务的状态 */
    INT8U  criti_level;  /* 混合关键任务的关键级别 */
    OS_TT_TASK_INFO  info[MAX_CRITICALITY];
    }OS_TT_TCB;
```

其中 OS _ TT _ TASK _ INFO 包含时间触发的关键任务处于各个关键级别时的信息，包括任务在各关键级别上的最坏执行时间和后续任务。而常量 MAX _ CRITICALITY 表示系统的最大的关键级别数，如果 MAX _ CRITICALITY 等于 1，则表示系统中只有一种关键级别的任务，则整个任务的调度就简化为普通的时间触发任务调度，只需要一张调度表即可。OS _ TT _ TASK _ INFO 的定义如下：

```
typedefstructos_tt_task_info
{  INT8U OSTT_TCBWcet;  /* TT 任务的最坏执行时间 */
   structos_tt_tcb * OSTT_TCBNext; /* 指向下一个 OS_TT_TCB 的指针 */
}OS_TT_TASK_INFO;
```

和 μC/OS－II 一样，事先定义了一批时间触发任务控制块，其数量由宏定义 OS _ TTTASK _ MAX 静态配置。系统将建立一个空闲 TT 任务链表和一个 TT 任务控制块链表来管理这些时间触发任务控制块。

3.2 调度表结构

时间触发的任务调度需要多张调度表，每张调度表就是一个任务控制块链表，对应每一个任务的优先级别。为此，设计了以下结构：

```
typedefstructos_tt_appmode
{
structos_tt_tcb *    first[MAX_CRITICALITY];    /* 指向任务的头结点 */
}OS_TT_APPMODE;
```

structos _ tt _ tcb * 类型的数组 first 包含了指向各张调度表中第一个 TT 任务控制块的指针，即各个 TT 任务控制块链表的头结点指针。

为了更方便快捷地管理被抢占的 TT 任务，设计了任务恢复链表，将被抢占的 TT 任务的任务控制块链接在一起，其中的任务按照最早时限排列，在恢复任务执行时，总是最先取出表头时限时间最找的任务。TT 任务恢复链表由任务恢复链表节点 OS _ TT _ PREEMPTED _ TCB 链接而成，其定义如下：

```
typedefstructos_preempted_tt_tcb
{
   OS_TT_TCB * OSTT_PTcb; /* 指向需要恢复执行的时间触发任务 */
   structos_preempted_tt_tcb * OSTT_PNext; /* 指向链表中下一个元素 */
}OS_TT_PREEMPTED_TCB;
```

和空闲 TT 任务链表类似，设计了一个 OS _ TT _ PREEMPTED _ TCB 类型的数组 OSTTPTCBTbl []，其元素个数由 OS _ TT _ PREEMPTED _ MAX 静态配置，表示系统最多可以维护被抢占的任务控制块的个数。把各个元素链接成一个链表，这就是空闲 TT 任务恢复链表，其头指针为 OSTTPTCBFreeList，用于任务恢复链表节点的分配；OSTTResumeFirst 和 OSTTResumeLast 分别为任务恢复链表头与表尾指针。

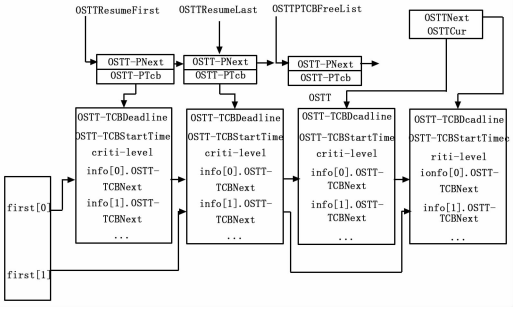

图 2　任务控制块链表和任务恢复链表

图 2 表示了 TT 任务控制块链表和空闲 TT 任务链表，任务恢复链表和空闲任务恢复链表及其相互间的关系。其中，OSTTCur 表示当前正在运行的 TT 任务的控制块，OSTTNext 指向下一个将要运行的 TT 任务的控制块。

3.3　任务调度的实现

时间触发模块中的任务调度和事件触发一样，也分为中断级调度和任务级调度。

3.3.1　中断级调度

时间触发模块中，中断级调度主要发生在时钟中断服务程序 OSTickISR（）中，调度通过 ttSchedTick（）函数实现。中断服务程序完成后退出中断时，调用 ttSchedTick（）函数，该函数会确认是否开始新一轮调度、是否有新的 TT 任务需要运行、是否 TT 任务都已执行完等各种情况。

1）如果开始新一轮调度，则时钟节拍重置为 0，将当前执行任务指针 OSTTCur 置为 NULL，将下一个将要执行的任务指针 OSTTNext 指向低级调度表中的第一个任务，开始重新执行；

如果不开始新一轮调度：

2）还没到下一个 TT 任务的触发时间，当前的 TT 任务还未执行完且执行时间未达到当前级别下的最坏执行时间时，则当前任务继续执行，不进行任务切换；

3）当前的 TT 任务是高级别安全关键任务，执行时间已达到当前级别下的最坏执行时间而还未执行完时，则系统发生状态切换，提高系统所处关键级别和调度表，根据调度表将 OSTTNext 指向高级调度表中下一任务，当前任务继续执行，不进行任务切换；

4）如果下一个 TT 任务触发，则抢占当前的 TT 任务并进行任务切换；

5）如果当前 TT 任务已执行完且下一个任务还未到达，则对时间触发模块来说，当前处于空闲时间，将系统切换到事件触发模块，进行 ET 任务的调度和切换。

中断级调度中的任务切换不需要保存任务上下文，因为任务被中断时已经进行了保存。中断级调度的主要流程图如图 3 所示。

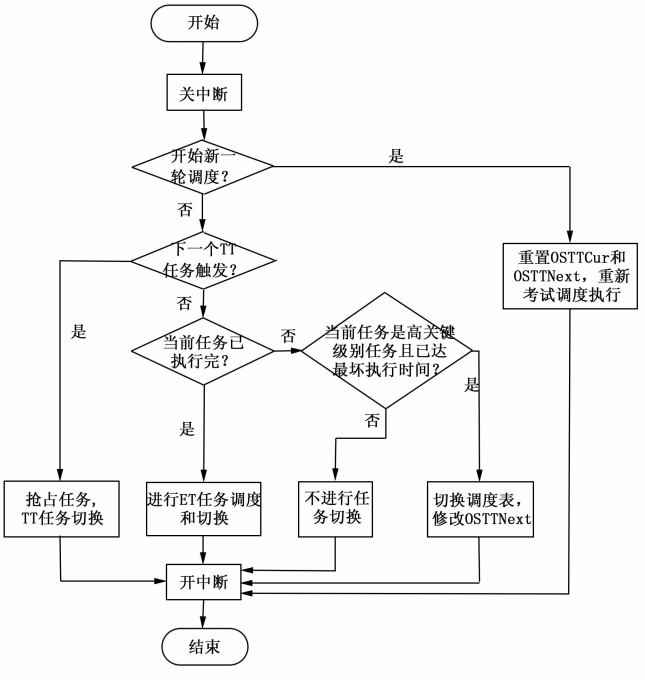

图 3　中断级调度流程图

3.3.2　任务级调度

时间触发模块中，任务级调度发生在任务执行完毕时，调用 ttTaskEnd（），可以让系统提前进行任务调度，而不必等时钟中断到来。

1）如果恢复链表为空，说明当前所有 TT 任务均已执行完毕，则将系统切换到事件触发模块，进行 ET 任务的调度和切换。

2）若恢复链表不为空，当前结束任务为恢复链表中最后一个任务，则将当前任务从恢复链表中删除，启动 ET 任务调度和切换。

3）若恢复链表不为空，当前结束任务不是恢复链表中最后一个任务，将当前任务从恢复链表中删除，然后恢复链表中下一个 TT 任务的执行，进行 TT 任务的任务切换。

4）若恢复链表不为空，且当前任务不在恢复链表中，则将恢复链表中的第一个任务恢复运行，进行 TT 任务切换。

任务级调度的主要流程图如图 4 所示。

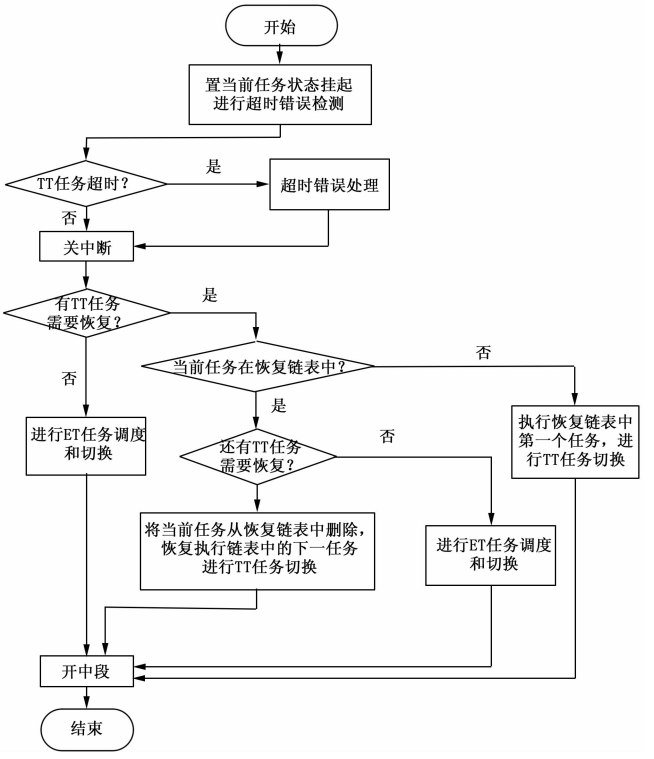

图 4 任务级调度流程图

4 实验验证

将设计的内核移植到 Xilinx Virtex－5 FXT FPGA ML507 的评估平台上进行测试。内核在开发板上的移植工作主要通过 Xilinx ISE Design Suite 12.3 软件完成。实验选取 4 个事件触发任务（ET 任务）和 3 个时间触发任务（TT 任务）；任务的各项参数如表 1 和 2 所示。

表 1 事件触发任务

任务名称	优先级	任务延时时间
etTask1	10	5 ms
etTask2	6	10 ms
etTask3	3	20 ms
etIdle	63	0 ms

表 2 时间触发任务

任务名称	开始时间（startTime）	最坏执行时间（wcet）	时限（deadline）	任务 id
ttTask1	10	10 ms	24	1
ttTask2	12	5 ms	20	2
ttTask3	30	5 ms	35	3

ET 任务的优先级数字越小表示其优先级越高，实验环境下系统时钟节拍为 1 ms，TT 任务执行周期为 50 ms。在完成多次实验后，选取任务开始执行的第一个周期内的执行结果，实验结果如图 5 所示。

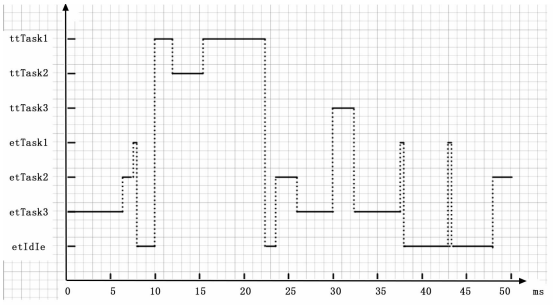

图 5 实验结果图

由实验结果可知，tick ＝ 0 时刻，ET 任务 etTask1，etTask2，etTask3，etIdle 在任务创建完成后就绪，由于优先级 etTask3 ＞ etTask2 ＞ etTask1 ＞etIdle，所以任务 etTask3 优先执行，此后 etTask2，etTask1 和 etIdle 依次执行。当 tick ＝ 10 时，ttTask1 开始执行，直到 tick ＝ 12 时，ttTask1 还未执行结束，这时 ttTask2 触发并抢占 ttTask1 任务执行。当 tick ＝ 15 时，ttTask2 执行结束，ttTask1 恢复执行；当 tick ＝ 22 时，ttTask1 执行结束。之后，ET 任务 etIdle，etTask2，etTask3 依次就绪并执行。当 tick ＝ 30 时，ttTask3 抢占 etTask3 开始执行；在 tick ＝ 32 时 ttTask3 执行结束，之后 etTask3 恢复执行并在 tick ＝ 37 时执行结束。此后，ET 任务 etIdle，etTask1，etTask2 依次就绪并执行。

5 结论

随着外部事件的不可预知性和实时任务复杂性的增加，单纯的事件触发机制或时间触发机制都难以保证系统的安全性和可靠性。只有将事件触发和时间触发结合起来，才能解决单一触发机制无法满足复杂嵌入式系统要求的问题，才能既保证系统的可靠性和确定性，又保证系统对外部事件拥有良好的响应能力。实验说明了这种方式可行，后期的工作将在时间触发模块内部，还要实现针对混合关键任务的调度算法和对应的安全级别调度表的设计，保证系统对安全关键任务的支持。在事件触发模块内部，还要实现包括决策和实施在内的访问控制机制，保证系统具备一定的安全性。

（下转第 211 页）