**Protothreads的简化**

-- 微控制器中基于协程的实时协作多任务方法 (1)

　　以小型微控制器，例如MCS51、MSP430、MICROCHIP等单片机、以至于ARM Cortex-M0甚至M3，为核心的嵌入式系统中，由于资源的限制，实时操作系统是无法施展其优势的。虽然资源丰富的芯片不断出现，但是，在单机控制领域，以及M2M或者说物联网的出现，小型微控制器还是具有相当的生命力。所以我们需要一个与之相适应的实时多任务编程方法。

　　特别是，需要高速轮转的多任务，比如任务的轮转周期为1毫秒，以至于亚毫秒（三、五百微秒），通常的实时操作系统是无能为力的。因为这点时间还不够实时操作系统倒腾。

　　针对这种情况，目前看来，如果用C来编程，Protothreads就是最好的选择了。自2005-2006年Adam Dunkels等推出Protothreads后[1,2]，2007-2008年以来，国内已有杂志上的文章对它做了介绍和推荐[3,4,5]。此外，Lych(边缘独行者) 的两篇博文，文字不多，却很深刻[6,7]。

　　大家都强调使用它极少内存和无堆栈。其实，我觉得更应该注重它在任务切换时的高效。这是通常的实时操作系统无法与之相比的。当需要高速切换任务时，切换效率就显得特别重要。

　　尽管Protothreads已被认为是精简到了极致，但是我还是想对它做一点简化，以推进Protothreads的使用。

　　强调一下，我不想从理论方面，譬如“实时操作系统”等方面讨论Protothreads。而是以工程应用的观点，从解决问题的方法方面进行讨论，并且主要针对小型微控制器的应用。这里我所提出的一些措施，对Protothreads进行简化，主要是为了更加提高它的切换效率。

　　好在语句不多，我们把主要的简化前后都列出来，以便对比。

**Protothread 1.4**

lc-switch.h ：

typedef unsigned short lc\_t;

#define LC\_INIT(s) s = 0;

#define LC\_RESUME(s) switch(s) { case 0:

**#define LC\_SET(s) s = \_\_LINE\_\_; case \_\_LINE\_\_:**

#define LC\_END(s) }

pt.h ：

#define PT\_INIT(pt) LC\_INIT((pt)->lc)

#define PT\_BEGIN(pt) { **char PT\_YIELD\_FLAG = 1;**LC\_RESUME((pt)->lc)

#define PT\_END(pt) LC\_END((pt)->lc); **PT\_YIELD\_FLAG = 0;** \

    PT\_INIT(pt); return PT\_ENDED; }

#define PT\_WAIT\_UNTIL(pt, condition) \

do { \

**LC\_SET((pt)->lc); \**

**if(!(condition)) { \**

**return PT\_WAITING; \**

} \

} while(0)

#define PT\_YIELD(pt) \

**do { \**

**PT\_YIELD\_FLAG = 0; \**

**LC\_SET((pt)->lc); \**

**if(PT\_YIELD\_FLAG == 0) { \**

**return PT\_YIELDED; \**

**} \**

**} while(0)**

**本文（by fy\_zhu）**

lc-sw.h ：

typedef unsigned short lc\_t;

#define LC\_INIT(s) s = 0;

#define LC\_RESUME(s) switch(s) { case 0:

**#define LC\_YIELD(s,ret\_code) s = \_\_LINE\_\_; return ret\_code ; case \_\_LINE\_\_:**

#define LC\_END(s) }

pt.h ：

#define PT\_INIT(pt) LC\_INIT((pt)->lc)

#define PT\_BEGIN(pt) { LC\_RESUME((pt)->lc)

#define PT\_END(pt) LC\_END((pt)->lc); \

    PT\_INIT(pt); return PT\_ENDED; }

#define PT\_WAIT\_UNTIL(pt, condition) \

do { \

**while(!(condition)) { \**

**LC\_YIELD((pt)->lc,PT\_WAITING); \**

} \

} while(0)

#define PT\_YIELD(pt) **LC\_YIELD((pt)->lc,PT\_YIELDED);**

　　文中的粗体红字，是主要的差别。简化的要点就是把“**case \_\_LINE\_\_:**”放在**return**之后，因而省去了与标志“**char PT\_YIELD\_FLAG**”有关的一些语句，进而使宏PT\_WAIT\_UNTIL和宏PT\_YIELD得到简化。这两个宏的指令多少，影响到任务切换效率。

　　事实上，“**case \_\_LINE\_\_:**”就是**return**之后要重入的后续点LC(Local Continuation)。为了LC\_SET的书写方便写在**return**之前，实在不合逻辑。而重入后，再借**PT\_YIELD\_FLAG**绕过**return**，就显得不简练了。

　　Protothreads的switch方法在各种C编译器都能通行，简化后的Protothreads的switch方法，用KEIL、IAR、CooCox+GCC编译器，在ARM Cortex-M0的实验板上调试通过。另外用KEIL、IAR、TASKING、RAISONANCE、SDCC的编译器，在C8051F的实验板上调试通过。

Protothreads的addrlabels方法也同样简化，我们只把有差别的部分写出：

**lc-addrlabels.h :**

#define LC\_SET(s) \

do { \

    LC\_CONCAT(LC\_LABEL, \_\_LINE\_\_): \

    (s) = &&LC\_CONCAT(LC\_LABEL, \_\_LINE\_\_); \

} while(0)

**本文（by fy\_zhu） :**

#define LC\_YIELD(s,ret\_code) \

do { \

    (s) = &&LC\_CONCAT(LC\_LABEL, \_\_LINE\_\_); \

    return ret\_code ; \

    LC\_CONCAT(LC\_LABEL, \_\_LINE\_\_): \

} while(0)

　　遗憾的是，小型微控制器常用的主流的C编译器KEIL、IAR等都不支持“**Labels as Values**”的扩展，因而简化的Protothreads的addrlabels方法也无法用这些编译器实施。在支持“**Labels as Values**”扩展的CooCox+GCC编译器，简化方法在ARM Cortex-M0的实验板上调试通过。

　　GCC不支持8051，因此简化的Protothreads的addrlabels方法也无法在8051上实施。

　　正如大家所知，C语言的switch语句的目标代码的执行效率比较低。尽管各家C编译器以各种方法改善switch语句的目标代码的执行效率，未尽人意。真正能使Protothreads发挥效率的，还是其addrlabels方法。

　　addrlabels方法的高执行效率和C编译器的不支持，是一大矛盾。如何解决这个矛盾，后面我们将逐步展开。

　　顺便说一句，在FreeRTOS的coroutine中，也是把“**case \_\_LINE\_\_:**”放在**return**之后。FreeRTOS的coroutine只使用switch-case方法，没有提供对addrlabels方法的支持。

　　所以，这里“Protothreads的简化”只不过一个引子，在了解这种编程思想的真谛后，我们就不难理解如何用各种C编译器实现比Protothreads的addrlabels方法目标代码的执行效率更高的实时协作多任务编程方法。

[1] Protothreads - Lightweight, Stackless Threads in C , SICS Technical Report T2005:05

[2] Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems

　　Proc. ACM SenSys, Nov 2006.

[3,4] 罗光平等

　　使用Protothread简化嵌入式系统中的顺序流控制，单片机与嵌入式系统应用， 2007年11期

　　利用Protothread实现实时多任务系统 单片机与嵌入式系统应用  2008年5期

[5] 闫石等 时间触发模式下的Protothreads设计应用 单片机与嵌入式系统应用  2009年1期

[6,7] 最轻量级的C协程库：Protothreads ，2008-05-05

　　传说中不可能的任务：在C语言中实现协程 ，2009-01-15

===

fy\_zhu

2013-02-12 SV\_CA