线程让出

RealTouch 评估板 RT-Thread 入门文档

版本号：1.0.0

日期：2012/8/12

修订记录

日期	作者	修订历史
2012/8/12	bloom5	创建文档

实验目的

- 快速熟悉线程相关接口

硬件说明

本实验使用 RT-Thread 官方的 Realtouch 开发板作为实验平台。涉及到的硬件主要为

- 串口 3，作为 rt_kprintf 输出，需要连接 JTAG 扩展板

具体请参见《Realtouch 开发板使用手册》

实验原理及程序结构

实验设计

本实验的主要设计目的是帮助读者快速了解线程相关 API，包括线程让出 API。请读者注意，本实验本身不具有实际的工程参考价值，只是帮助读者快速了解线程 API 的用法。

源程序说明

本实验对应 1_kernel_thread_yield

系统依赖

在 rtconfig.h 中需要开启

- #define RT_USING_HEAP
  此项可选，开启此项可以创建动态线程和动态信号量，如果使用静态线程和静态信号量，则此项不是必要的
- #define RT_USING_CONSOLE
  此项必须，本实验使用rt_kprintf向串口打印按键信息，因此需要开启此项

主程序说明

在 applications/application.c 中的定义了两个线程 tid1、tid2，

下面的代码是 thread1 的入口程序，thread1 得到执行后就会计数并打印出来，打印后就调用 rt_thread_yield()，让出处理机，thread2 的入口程序与之几乎相同。

```
/* 线程 1 入口 */
static void thread1_entry(void* parameter)
{
    rt_uint32_t count = 0;

    while (1)
    {
        /* 打印线程 1 的输出 */
        rt_kprintf("thread1: count = %d\n", count ++);

        /* 执行 yield 后应该切换到 thread2 执行 */
        rt_thread_yield();
    }
}

/* 线程 2 入口 */
static void thread2_entry(void* parameter)
{
    rt_uint32_t count = 0;

    while (1)
    {
        /* 执行 yield 后应该切换到 thread1 执行 */
        rt_thread_yield();

/* 打印线程 2 的输出 */
        rt_kprintf("thread2: count = %d\n", count ++);
    }
}
```

编译调试及观察输出信息

编译请参见《RT-Thread 配置开发环境指南》完成编译烧录，参考《Realtouch 开发板使用手册》完成硬件连接，连接扩展板上的串口和 jlink。

运行后可以看到如下信息：

```
 \ | /
- RT -     Thread Operating System
 / | \     1.1.0 build Aug 10 2012
 2006 - 2012 Copyright by rt-thread team
thread1: count = 0
thread1: count = 1
thread2: count = 0
thread1: count = 2
thread2: count = 1
thread1: count = 3
thread2: count = 2
thread1: count = 4
thread2: count = 3
thread2: count = 4
```

结果分析

从打印结果中看到，thread1 连续打印了两次，我们可以就此分析一下。thread1 首先执行，打印计数结果，然后让出，

```
rt_thread_yield();
```

系统将启动调度 thread2 得到执行，thread2 执行过程中遇到 yield()函数，将执行交还给 thread1，此时 thread2 中 yield()函数下的计数打印语句并未得到执行，thread1 再次执行计数打印，然后 yield()，当执行权再次回到 thread2 手上的时候，thread2 则需要弥补之前所遗留的痛，打印计数咯就。所以我们可以看到 thread1 打印了两次而 thread2 只打印了一次。可能你发现接下来，thread1 和 thread2 均只打印了一次，这又是什么原因呢。由前三个打印结果可以分析出，两个线程在规定的时间片内可以

完成计数打印和 yield()两个操作，所以在 thread2 第一次得到控制权时候，它完成了打印操作，然后又一次执行了 yield()，所以当它在此获得控制权时就会直接打印计数了。所以说 thread1 打印两次计数结果，thread2 打印一次的情况只会在开始时候出现一次。