线程优先级抢占

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

- 加深优先级调度认识

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优先级的调度过程。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_priority

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 thread_priority_init()函数中初始化了两个线程 t1、t2，

```
result = rt_thread_init(&thread1,
            "t1",
            thread1_entry, RT_NULL,
            &thread1_stack[0], sizeof(thread1_stack),5, 5);

    if (result == RT_EOK)
            rt_thread_startup(&thread1);

resul = rt_thread_init(&thread2,
            "t2",
            thread2_entry, RT_NULL,
            &thread2_stack[0], sizeof(thread2_stack),7, 5);

    if (result == RT_EOK)
            rt_thread_startup(&thread2);
```

两段初始化代码最显著的区别就是优先级不同。

因为优先级较高，首先得到执行的是 t1。

```
static void thread1_entry(void* parameter)
{
    for(;count<10;count++)
    {
            rt_thread_delay(3*RT_TICK_PER_SECOND);
            rt_kprintf("count = %d\n", count);
    }
}
```

t1 执行到了 rt_thread_delay(2*RT_TICK_PER_SECOND),t2 得到执行，t2 的主要任务就是获得当前系统 tick，并打印

```
static void thread2_entry(void* parameter)
{
    rt_tick_t tick;
    rt_uint32_t i;
```

```
    for(i=0; ; i++)
    {
            tick = rt_tick_get();
            rt_thread_delay(RT_TICK_PER_SECOND);
    rt_kprintf("tick = %d\n",tick);
    }
}
```

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
tick = 1
tick = 1001
count = 0
tick = 2001
tick = 3002
tick = 4002
count = 1
tick = 5002
tick = 6002
tick = 7002
count = 2
tick = 8002
tick = 9002
tick = 10002
count = 3
tick = 11003
```

```
tick = 12004
tick = 13005
count = 4
tick = 14006
......
```

结果分析

因为更高的优先级，thread1 率先得到执行，随后它调用延时，时间为 3 个系统 tick，于是 thread2 得到执行。可以从打印结果中发现一个规律，在第一次 thread2 了打印两次 thread1 会打印一次之后，接下来的话 thread2 每打印三次 thread1 会打印一次。对两个线程的入口程序进行分析可以发现，在 thread1 3 个系统 tick 的延时里，thread2 实际会得到三次执行机会，但显然在 thread1 的第一个延时内 thread2 第三次执行并没有执行结束，在第三次延时结束以后，thread2 本应该执行第三次打印计数的，但是由于 thread1 此时的延时也结束了，而其优先级相比 thread2 要高，所以抢占了 thread2 的执行而开始执行。当 thread1 再次进入延时时，之前被抢占的 thread2 的打印得以继续，然后在经过两次 1 个系统 tick 延时和两次打印计数后，在第三次系统 tick 结束后又遇到了 thread1 的延时结束，thread1 再次抢占获得执行，所以在这次 thread1 打印之前，thread2 执行了三次打印计数。