静态线程初始化与脱离

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

- 快速熟悉静态线程相关接口
- 可以使用线程实现简单任务

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，包括静态线程的创建/删除、相关 API，为了简化起见，我们将这些 API 放在同一个线程中调用。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_static

系统依赖

在 rtconfig.h 中需要开启

- #define RT_USING_CONSOLE
  此项必须，本实验使用 rt_kpriintf 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 中定义了两个线程数据结构以及相应栈

```
static struct rt_thread thread1;
static rt_uint8_t thread1_stack[512];
static struct rt_thread thread2;
static rt_uint8_t thread2_stack[512];
```

application.c 中的 thread_detach_init()函数中初始化了两个静态线程 t1、t2，静态线程的初始化对应于动态线程的创建(create)有一点不同，由于静态线程的栈是用户提供建立的，在编译时分配，内核不会再进行动态分配空间，所以比动态线程多两个参数：线程数据结构以及线程栈的开始地址

```
result = rt_thread_init(&thread1, "t1", /* 线程名：t1 */
            thread1_entry, RT_NULL,
            /* 线程的入口是 thread1_entry，入口参数是 RT_NULL*/
            &thread1_stack[0], sizeof(thread1_stack),
            /* 线程栈是 thread1_stack */
            6, 10);
if (result == RT_EOK) /* 如果返回正确，启动线程 1 */
    rt_thread_startup(&thread1);

result = rt_thread_init(&thread2, "t2", /* 线程名：t2 */
            thread2_entry, RT_NULL,
            /* 线程的入口是 thread2_entry，入口参数是 RT_NULL*/
            &thread2_stack[0], sizeof(thread2_stack),
            /* 线程栈是 thread2_stack */
            5, 10);
if (result == RT_EOK) /* 如果返回正确，启动线程 2 */
    rt_thread_startup(&thread2);
```

下面的代码是两个线程的入口程序，在 thread2 的入口程序中将会去脱离 thread1，也就是从就绪线程队列中删除 thread1

```
static void thread1_entry(void* parameter)
{
    rt_uint32_t count = 0;
```

```
    while (1)
    {
        rt_kprintf("thread count: %d\n", count ++);
        rt_thread_delay(RT_TICK_PER_SECOND);
    }
}

static void thread2_entry(void* parameter)
{
    rt_thread_delay(RT_TICK_PER_SECOND*10);
    /*
     * 线程 2 唤醒后直接执行线程 1 脱离，线程 1 将从就绪线程队列中删除
     */
    rt_thread_detach(&thread1);

    rt_thread_delay(10);
}
```

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
thread count: 0
thread count: 1
thread count: 2
thread count: 3
thread count: 4
thread count: 5
thread count: 6
```

```
thread count: 7
thread count: 8
thread count: 9
```

结果分析

因为 thread2 拥有更高的优先级，所以在初始化两个线程成功后 thread2 首先得到执行，得到执行后 thread2 延时 10 个系统 tick，

```
rt_thread_delay(RT_TICK_PER_SECOND*10);
```

此时系统调度到 thread1 执行，thread1 执行计数打印，可以看到 10 次计数打印。当 thread2 的延时到达时，它将重新获得执行权。Thread2 调用 rt_thread_detach()函数将 thread1 脱离线程就绪队列，

```
rt_thread_detach(&thread1);
```

从而线程 1 不会再被调度执行。

总结

什么是动态线程？什么是静态线程？两者有什么区别？

RT-Thread 中支持静态和动态两种定义方式。用线程来举例的话，rt_thread_init 对应静态定义方式， rt_thread_create 对应动态定义方式。

使用静态定义方式时，必须先定义静态的线程控制块，并且定义好堆栈空间，然后调用 rt_thread_init 来完成线程的初始化工作。采用这种方式，线程控制块和堆栈占用的内存会放在 RW 段，这段内存空间在编译时就已经确定，它不是可以动态分配的，所以不能被释放，而只能使用 rt_thread_detach 函数将该线程控制块从对象管理器中脱离。

使用动态定义方式 rt_thread_create 时， RT-Thread 会动态申请线程控制块和堆栈空间。当不需要使用该线程时，调用 rt_thread_delete 函数就会将这段申请的内存空间重新释放到内存堆中（如果线程执行完毕，退出时，系统也会自动回收线程控制块和堆栈空间）