线程优先级反转之优先级继承算法

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

- 本实验的主要设计目的是帮助读者了解线程优先级反转的一种解决方法。

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

优先级继承是指将占有共享资源的的低优先级线程的优先级临时提高到等待该共享资源的所有线程中，优先级最高的那个线程的优先级，当高优先级线程由于等待共享资源被阻塞时，拥有共享资源的线程的优先级会被自动提升，从而避免出现优先级反转问题。

在 RT-Thread 的 mutex 设计中已经实现了优先级继承这种算法，因此，使用 mutex 即可以使用优先级继承的算法解决优先级反转问题。

源程序说明

本实验对应 1_kernel_thread_prioinherit

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

如上节一样，还是定义了三个线程，t1,t2,worker,建立过程也完全相同，所以不再赘述。

下面的代码是三个线程的入口程序：

```
static void thread1_entry(void* parameter)
{
    rt_err_t result;

    result = rt_mutex_take(mutex, RT_WAITING_FOREVER);
    result = rt_mutex_take(mutex, RT_WAITING_FOREVER);
    rt_kprintf("thread1: got mutex\n");

    if (result != RT_EOK)
    {
        return;
    }

    for(t1_count = 0; t1_count < 5;t1_count ++)
    {
        rt_kprintf("thread1:count: %d\n", t1_count);
    }

    rt_kprintf("thread1: released mutex\n");
    rt_mutex_release(mutex);
    rt_mutex_release(mutex);
}

static void thread2_entry(void* parameter)
{
    rt_err_t result;
    rt_thread_delay(5);
```

```
    result = rt_mutex_take(mutex, RT_WAITING_FOREVER);
    rt_kprintf("thread2: got mutex\n");
    for(t2_count = 0; t2_count < 5;t2_count ++)
    {
     rt_kprintf("thread2: count: %d\n", t2_count);
    }
}

static void worker_thread_entry(void* parameter)
{
    rt_thread_delay(5);
    for(worker_count = 0; worker_count < 5; worker_count++)
    {
     rt_kprintf("worker:count: %d\n", worker_count);
     rt_thread_delay(5);
    }
}
```

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 11 2012
 2006 - 2012 Copyright by rt-thread team
thread1: got mutex
thread1:count: 0
thread1:count: 1
thread1:count: 2
thread1:count: 3
```

```
thread1:count: 4
thread1: released mutex
thread2: got mutex
thread2: count: 0
thread2: count: 1
thread2: count: 2
thread2: count: 3
thread2: count: 4
worker:count: 0
worker:count: 1
worker:count: 2
worker:count: 3
worker:count: 4
….
```

结果分析

在上一节已经分析过由于信号量可能引入的优先级反转问题，在 RT-Thread 中实现的是优先级继承算法。优先级继承通过在线程 A 被阻塞期间提升线程 C 的优先级到线程 A 的优先级别从而解决优先级翻转引起的问题。如图 1-1 所示

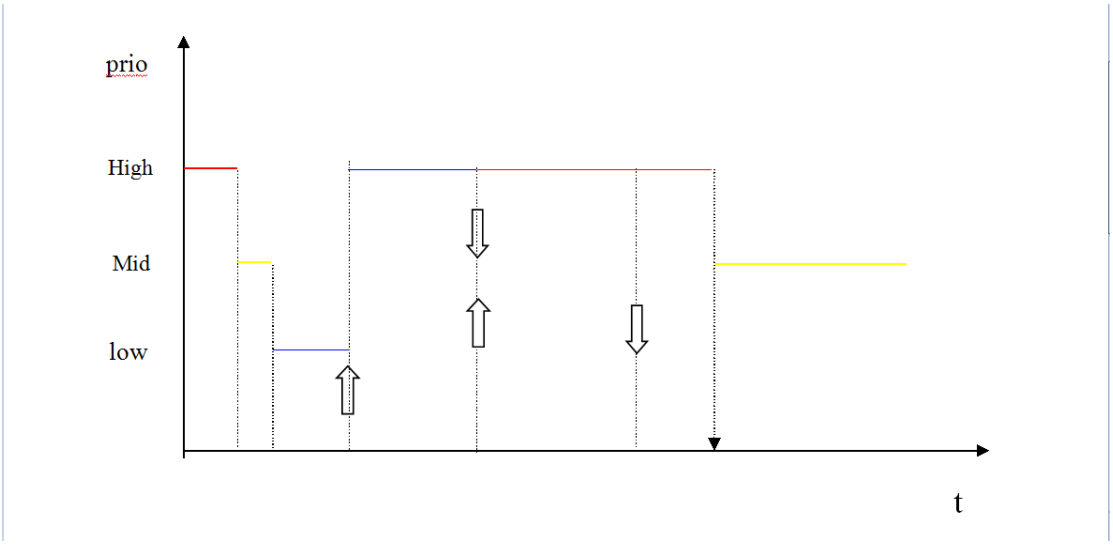

图 1-1

根据打印结果，我们可以看到 thread2 和 worker 线程虽然优先级比 thread1 要高，但是这两个线程均在进程开始出就执行了延时函数，于是轮

到 thread1 执行，然后 thread1 获得互斥量，thread2 延时结束后，虽然它的优先级高于 thread1，但是它所需的互斥量被 thread1 占有了，它无法获得所需的互斥量以便继续运行。在此时，系统的优先级继承算法也会起作用，将 thread1 的优先级提升到与 thread2 一致，验证方法是在 thread1 release 互斥量之前插入 tid2->currentpriority 是否等于 tid1->currentpriority 的判断语句，当然此时的结果是相等的。当 thread1 优先级被提升到和 thread2 一样后，worker 线程优先级因为低于 thread1 的优先级而不再能够抢占 thread1，从而保证避免优先级反转现象发生。

所以说，优先级反转的问题可以通过优先级继承来解决，在 RT-Thread 的 mutex 中实现了优先级继承算法。