RTT时间触发（TT）线程拓展测试记录

一、测试环境

1、硬件环境：

STM32H743开发板，SRAM：1060KB。

2、关键参数设置

跳表层数：5。

二、测试计划

1、测试代码

1.1 TT（时间触发）线程入口函数代码

```
static void TT_thread_entry(void *parameter)
{
        rt_thread_t cur_thread = rt_thread_self();
        rt_base_t cycle = cur_thread->thread_exec_cycle, offset = cur_thread-
>thread_exec_offset, cycle_count = -1;
        rt_base_t bound = offset / 100 + 1, cnt = 0;
    while(++cnt <= bound)
        {
            set_TT_running(1);
        rt_tick_t delta = rt_get_global_time();
                rt_base_t cur_cycle_count = (delta - TT_start_time - offset) /
cycle;
                if((delta - TT_start_time - offset) % cycle != 0 || (cycle_count
!= -1 && cur_cycle_count != cycle_count + 1))
                {
                        rt_base_t error = delta -
get_first_TT_Thread_start_time();
                        if(error == 1)
                        {
                                one_tick_error++;
                        }
                        else if(error == 2)
                        {
                                two_tick_error++;
                        }
                        else
                        {
                                if(error != upper_bound)
                                {
                                        upper_bound = error;
                                }
                                fatal_error_count++;
                        }
```

```
                }

                rt_kprintf("State: %d %d | %d %d %d %d %d | %d %d | %d %d %d %d
%d | %d\n",
                                        get_first_TT_Thread_start_time() -
TT_start_time, delta - TT_start_time,
                                        cycle_count, cur_cycle_count, cnt,
cycle, offset,
                                        total_TT_thread,
get_running_TT_Thread_count(),
                                        global_exec_count, one_tick_error,
two_tick_error, fatal_error_count, upper_bound,
                                        get_serial_error()
                                    );

                cycle_count = cur_cycle_count;
                global_exec_count++;
            set_TT_running(0);
        rt_thread_yield();
    }
}
```

入口函数的主要目的是：

A、使得TT线程的生存周期不一致；

B、比较TT线程每一次的运行时间和指定时间，记录错误类型和数目，显示每一次TT线程运行时的状态。

1.2 ET线程代码

```
static void ET_thread_entry(void *parameter)
{
        rt_base_t num = 0;
        while(1)
        {
                rt_kprintf("ET_thread low : %d\n", num);
                num++;
                num %= 100;
                rt_thread_mdelay(50);
        }
}

static void ET_high_thread_entry(void *parameter)
{
        rt_base_t num = 0;
        while(1)
        {
                rt_kprintf("ET_thread high : %d\n", num % 100);
                num++;
                num %= 100;
                rt_thread_mdelay(500);
        }
}
```

设置了两个ET线程入口函数，均为无限循环执行，且没有任何主动挂起措施。高优先级ET线程入口函数执行一次之后挂起500毫秒，低优先级线程入口函数执行一次之后挂起50毫秒。

1.3 随机数代码

```
unsigned long int rand_next = 1;
void srand(unsigned int seed)
{
    rand_next = seed;
}
int rand ()
{
    rand_next = rand_next * 1103515245 + 12345;
    return ((unsigned int)(rand_next / 65536) % 32768);
}
```

每一次测试使用的随机数都是合格的随机数。

1.4 线程生成代码

```
        while(1)
        {
                ++i;
                global_offset = rand() % 6000;
                global_cycle = 6000;

                TT_thread = rt_TT_thread_create("TT_thread",
                            TT_thread_entry, RT_NULL,
                            THREAD_STACK_SIZE,
                            THREAD_PRIORITY, THREAD_TIMESLICE,
                                                    6000,
                                                    global_offset, 10);

                rt_kprintf("%d : ", i);
                if (TT_thread != RT_NULL)
                {
                        rt_kprintf("%d : %d\n", ++cnt, global_offset);
                        rt_thread_startup(TT_thread);
                }
                else
                {
                        rt_kprintf("%d faild\n", i);
                }
                rt_thread_mdelay(100);
        }
```

这一代码不断尝试创建和运行新的TT线程，线程的周期均为6000，偏移随机，最大运行时长为10。

2、错误类型

I型错误：实际执行时间晚于指定执行时间1个tick（1ms）

II型错误：实际执行时间晚于指定执行时间2个tick（2ms）

III型错误：不同于I型错误和II型错误的其他错误。

三、测试结果

1、正确性测试

1.1 测试目的

在没有任何干扰，线程数目较少的情况下，测试TT（时间触发）线程能够在指定的时间点触发运行。

1.2 测试计划

随机生成50个TT线程，所有线程只创建一次，且无限循环；

运行至少20000次，记录错误类型及数目。

1.3 测试结果

编号	线程类型	周期	偏移	时间片	描述
0~50	TT	6000	随机	10	无限循环执行

运行时长	运行次数	TT线程累计成功创建个数	同时运行的TT线程个数	错误类型及数目
14小时	438032	50	50	无错误

1.4 结果分析

50个TT线程在所有指定时间点按时触发执行，没有错误。

2、ET线程干扰测试

2.1 测试目的

在ET（时间触发）线程的干扰中，少量的TT（时间触发）线程能够在指定的时间点触发运行。

2.2 测试计划

随机生成50个TT线程，所有线程只创建一次，且无限循环；

生成一个无限循环的ET线程；

运行至少20000次，记录错误类型及数目。

2.3 测试结果

编号	线程类型	周期	偏移	时间片	描述
0~50	TT	6000	随机	10	
51	ET				不断执行计算

运行时长	运行次数	TT线程累计成功创建个数	同时运行的TT线程个数	错误类型及数目
5小时	145818	50	50	无错误

2.4 结果分析

在ET线程的干扰下，TT线程仍然能够在指定的时刻触发执行。

3、时间冲突测试

3.1 测试目的

由于TT线程之间不能有时间上的冲突，因此在TT线程申请时，必须要进行冲突检查，本次测试是为了检验冲突检测模块的运行是否正确。

3.2 测试计划

生成三个线程，TT线程1与TT线程2冲突，TT线程2与TT线程3冲突；

观察创建于运行情况。

3.3 测试结果

编号	线程类型	周期	偏移	时间片	描述
1	TT	600	380	20	
2	TT	1000	200	20	
3	TT	1500	720	20	
运行时长	**运行次数**				
<1分钟	15				

TT线程1与TT线程3创建成功，TT线程2显示时间冲突，创建失败。

3.4 结果分析

TT线程1首先创建，不会产生时间冲突；

TT线程2与TT线程1冲突，创建失败；

TT线程3与TT线程2冲入，但是TT线程2没有被成功创建，因此创建成功；

时间冲突检测模块运行正确。

4、TT线程超时测试

4.1 测试目的

TT线程必须声明自己的最长运行时间，如果超时，则有可能影响其他TT线程，因此超时的TT线程必须被杀死。本次测试是为了检验超时检测模块运行是否正确。设置超时钩子函数，测试钩子函数的设置和删除逻辑是否正确。

4.2 测试计划

生成五个线程，TT线程2与TT线程3的时间片极短；

设置一个钩子函数，在其中执行打印操作。

观察所有线程创建于运行情况。

4.3 测试结果

编号	线程类型	周期	偏移	时间片	描述
1	TT	500	370	10	
2	TT	1000	100	1	
3	TT	1000	600	2	
4	TT	2500	0	10	
5	TT	3000	200	10	
运行时长	**运行次数**				
<1分钟	18				

TT线程1，4，5正常运行，TT线程2,3在第一次运行中退出。钩子函数执行正常。

4.4 结果分析

超时检测模块运行正确，超时线程被杀死，且不影响正常TT线程。超时异常钩子函数的设置、执行逻辑正确。

5、压力测试

5.1 测试目的

在不断创建和终止TT线程的情况下，测试系统能否正常运行，线程能否在指定时间点触发执行。

5.2 测试计划

不断随机生成TT线程；

TT线程的入口函数是有限循环；

测试TT线程执行时间正确性；

5.3 测试结果

编号	线程类型	周期	偏移	时间片	描述
inf	TT线程	6000	随机	10	验证正确性

运行时长	运行次数	TT线程累计成功创建个数	同时运行的TT线程个数	错误类型及数目
19小时	3355765	165815	301	无错误

5.4 结果分析

在将系统资源使用到极致，并且不断生成、启动、销毁TT线程的情况下，系统未出现异常，且TT线程没有出现错误，说明系统实现具有稳定性和正确性。

6、ET线程干扰压力测试

5.1 测试目的

在有ET线程不断干扰的情况下，不断创建和终止TT线程，测试系统能否正常运行，线程能否在指定时间点触发执行。

5.2 测试计划

不断随机生成TT线程；

生成10个ET线程，其中5个优先级为25，另外5个优先级为24

TT线程的入口函数是有限循环；

测试TT线程执行时间正确性。

5.3 测试结果

编号	线程类型	周期	偏移	时间片	描述
1-5	ET线程（优先级25）	-	-	5	无限循环，每次执行之后等待50毫秒
6-10	ET线程（优先级24）	-	-	5	无限循环，每次执行之后等待500毫秒
inf	TT线程	6000	随机	10	有限循环，循环次数是1~60的随机数

运行时长	运行次数	TT线程累计成功创建个数	同时运行的TT线程个数	错误类型及数目
2小时	389221	19351	270左右	无错误
17小时	2759182	136043	277左右	无错误
18小时	2885174	142532	279左右	无错误

5.4 结果分析

在有较多ET线程的干扰且系统资源使用到极致的情况下，TT线程能够正常创建、启动、挂起和销毁，系统运行平稳无异常，说明系统实现可靠且正确。