第 2 章

RT-Thread 实时线程管理及调度

2.1

–

2.2

RT-Thread

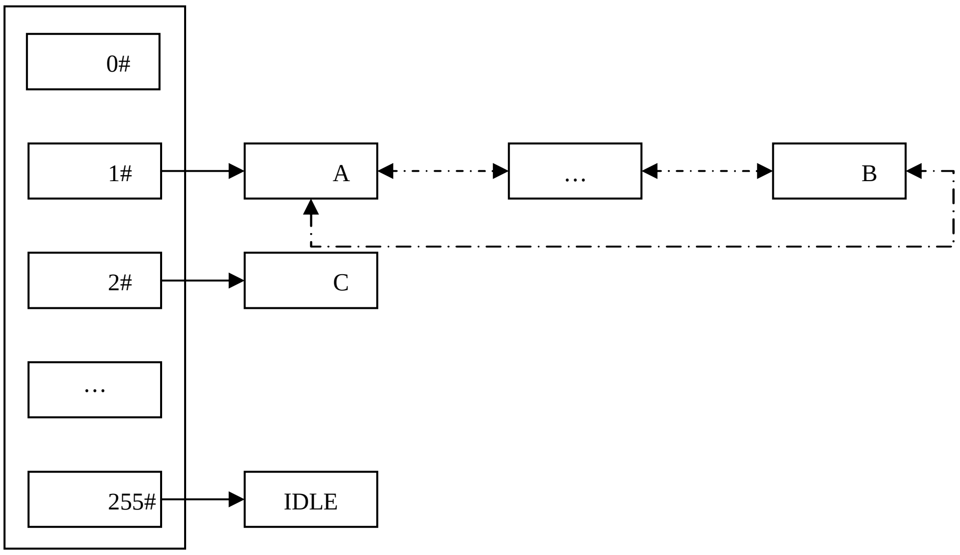
256 (0 255 255

32)

图 2 - 1 线程优先级队列

RT-Thread 256
255 idle
1# 2# A – C A B C
C A – B

图 2 - 1 线程优先级队列



RT-Thread
A B
10 7 B A
10 A

RT-Thread

RT-Thread O(1)

2.3

RT-Thread `struct rt_thread`
`rt_thread_t` C

代码 2 - 1 线程控制块

```
typedef struct rt_thread* rt_thread_t;

struct rt_thread
{
    /* rt object */
    char         name[RT_NAME_MAX];            /* the name of thread */
    rt_uint8_t   type;                         /* type of object */
    rt_uint8_t   flags;                        /* thread's flags */

    rt_list_t    list;                         /* the object list */

    rt_thread_t  tid;                          /* the thread id */
    rt_list_t    tlist;                        /* the thread list */

    /* stack point and entry */
    void*        sp;                           /* stack point */
    void*        entry;                        /* entry */
    void*        parameter;                    /* parameter */
    void*        stack_addr;                   /* stack address */
    rt_uint16_t stack_size;                    /* stack size */

    /* error code */
    rt_err_t     error;                        /* error code */

    /* priority */
    rt_uint8_t   current_priority;             /* current priority */
    rt_uint8_t   init_priority;                /* initialized priority */
#if RT_THREAD_PRIORITY_MAX > 32
    rt_uint8_t   number;
    rt_uint8_t   high_mask;
#endif
    rt_uint32_t number_mask;

#if defined(RT_USING_EVENT) || defined(RT_USING_FASTEVENT)
    /* thread event */
    rt_uint32_t event_set;
    rt_uint8_t   event_info;
#endif

    rt_uint8_t   stat;                         /* thread stat */

    rt_ubase_t   init_tick;                    /* thread's tick */
    rt_ubase_t   remaining_tick;               /* remaining tick */

    struct rt_timer thread_timer;              /* thread timer */

    rt_uint32_t user_data;                     /* user data */
};
```

user_data

2.4

RT-Thread

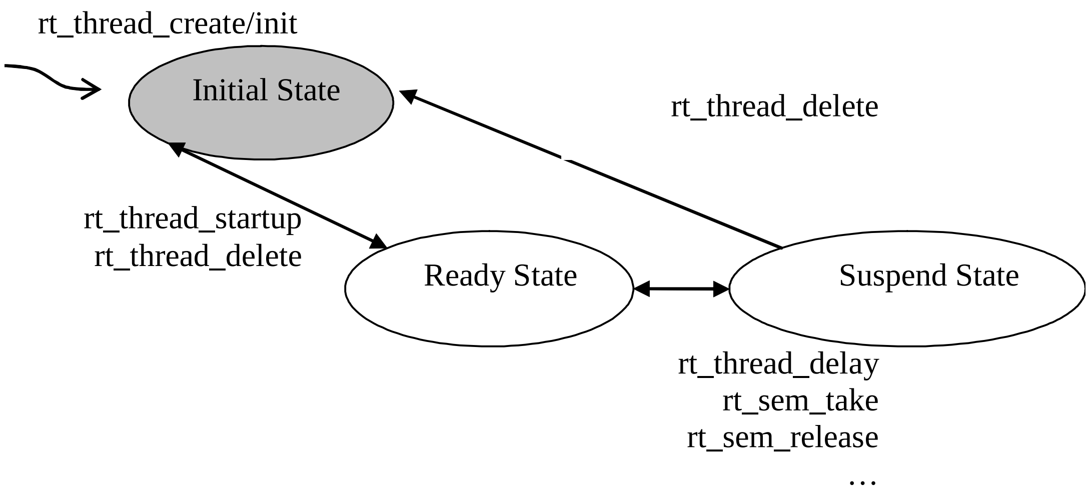
RT-Thread

RT_THREAD_INIT/CLOSE	
RT_THREAD_SUSPEND	
RT_THREAD_READY	

RT-Thread RTOS

(rt_sem_take)

图 2 - 2 线程状态转换图



rt_thread_create/init Initial State, RT_THREAD_INIT rt_thread_startup Ready State RT_THREAD_READY rt_thread_delay rt_sem_take rt_mb_recv Suspend

`State  RT_THREAD_SUSPEND`

2.5

RT-Thread

2.6

2.6.1

```
void rt_system_scheduler_init(void)
```

2.6.2

```
void rt_system_scheduler_start(void)
```

`idle`

2.6.3

```
void rt_schedule(void)
```

2.7

2.7.1

/

```
rt_thread_t rt_thread_create (const char* name,
   void (*entry)(void* parameter), void* parameter,
   rt_uint32_t stack_size,
   rt_uint8_t priority, rt_uint32_t tick)
```

RT_NAME_MAX

ARM 4

0 255

TCB

代码 2-2 创建线程

```
#include <rtthread.h>

/*        */
static void entry(void* parameter)
{
    rt_uint32_t count = 0;
    while (1)
    {
        rt_kprintf("count:%d\n", ++count);
        rt_thread_delay(50);
    }
}

/*               */
int rt_application_init()
{
    /*                 entry    */
    rt_thread_t thread = rt_thread_create("t1",
        entry, RT_NULL,
        1024, 200, 10);
    if (thread != RT_NULL)
        rt_thread_startup(thread);

    return 0;
}
```

2.7.2

rt_thread_create

rt_err_t rt_thread_delete (rt_thread_t thread)

rt_thread_init

代码 2-3 删除线程

```
#include <rtthread.h>

void thread1_entry(void* parameter)
{
    rt_uint32_t count = 0;
    while (1)
    {
        rt_kprintf("count:%d\n", ++count);
        rt_thread_delay(50);
    }
}

rt_uint32_t to_delete_thread1 = 0;
rt_thread_t thread1, thread2;
void thread2_entry(void* parameter)
{
    while (1)
    {
        if (to_delete_thread1 == 1)
        {
            /* to_delete_thread1        thread1 */
            rt_thread_delete(thread1);

            /*             */
            return ;
        }

        /* to_delete_thread1          100                */
        rt_thread_delay(100);
    }
}

int rt_application_init()
{
    /*    thread1      */
    thread1 = rt_thread_create("t1",
        thread1_entry, RT_NULL,
        1024, 200, 10);
    if (thread1 != RT_NULL) rt_thread_startup(thread1);

    /*    thread2      */
    thread2 = rt_thread_create("t2",
        thread2_entry, RT_NULL,
        1024, 120, 10);
    if (thread2 != RT_NULL) rt_thread_startup(thread2);

    return 0;
}
```

2.7.3

```
rt_err_t rt_thread_init(struct rt_thread* thread,
    const char* name,
    void (*entry)(void* parameter), void* parameter,
    void* stack_start, rt_uint32_t stack_size,
    rt_uint8_t priority, rt_uint32_t tick);
```

代码 2-4 线程初始化

```
#include <rtthread.h>

static rt_uint8_t thread_stack[512];
static struct rt_thread thread;

/*        */
static void entry(void* parameter)
{
    int i;
    rt_thread_t self;

    /*               */
    self = rt_thread_self();

    while (1)
    {
        rt_kprintf("thread[%s] count %d\n", self->name, ++i);
        rt_thread_delay(100);
    }
}

int rt_application_init()
{
    /*          */
```

```
        rt_thread_init(&thread,
            "thread1",
            entry, RT_NULL,
            &thread_stack[0], sizeof(thread_stack),
            200, 10);
        /*        */
        rt_thread_startup(&thread);

        return 0;
    }
```

2.7.4

rt_err_t rt_thread_detach (rt_thread_t thread)

rt_thread_delete rt_thread_delete
rt_thread_create rt_thread_detach
rt_thread_init

2.7.5

/ RT_THREAD_INIT
/
rt_err_t rt_thread_startup (rt_thread_t thread)

2.7.6

rt_thread_t rt_thread_self (void)

2.7.7

rt_err_t rt_thread_yield ()

代码 2-5 让出处理机

```
void funcion()
{
    ...
    rt_thread_yield();
    ...
}
```

rt_thread_yield   rt_schedule

rt_thread_yield
rt_schedule

2.7.8

rt_err_t rt_thread_sleep(rt_tick_t tick)
rt_err_t rt_thread_delay(rt_tick_t tick)

2.7.9

rt_thread_delay rt_sem_take rt_mb_recv

rt_err_t rt_thread_suspend (rt_thread_t thread)

rt_schedule

代码 2-6 挂起线程代码

```
#include <rtthread.h>

/*         */
rt_thread_t thread = RT_NULL;

/*         */
```

```
static void entry(void* parameter)
{
    while (1)
    {
        /*        */
        rt_thread_suspend(thread);

        /*    rt_thread_suspend     thread
         *
         */
        rt_schedule();

        /*                thread         */
        rt_kprintf("thread is resumed\n");
    }
}

int rt_application_init()
{
    /*        */
    thread = rt_thread_create("tid", entry, RT_NULL, 1024, 250, 20);

    /*        */
    rt_thread_startup(thread);

    return 0;
}
```

2.7.10

rt_err_t rt_thread_resume (rt_thread_t thread)

代码 2-7 恢复挂起线程

```
#include <rtthread.h>

rt_thread_t thread = RT_NULL;
void function ()
{
   ...
   rt_thread_resume(thread);
   ...
}
```

rt_err_t rt_thread_control(rt_thread_t thread, rt_uint8_t cmd, void* arg)

2.7.11

```
void rt_thread_idle_init(void)
```

2.7.12

```
void rt_thread_idle_set_hook(void (*hook)())
```

rt_thread_delay
rt_sem_take