RT-Thread RTOS

1

事件集

实时操作系统培训

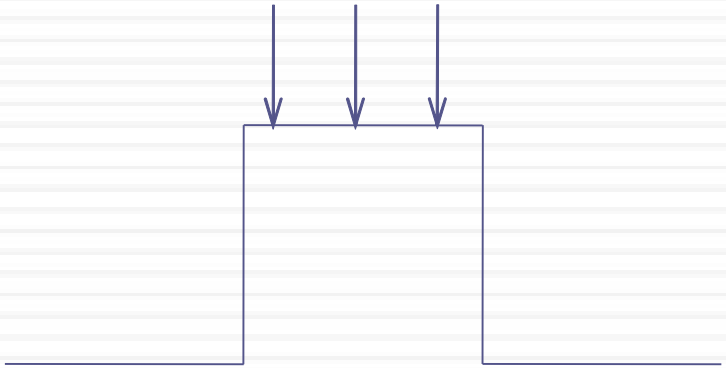
RT-Thread线程间同步：事件

- 在嵌入式实时内核中，事件是指一种表明预先定义的系统事件已经发生的机制。
- 事件机制用于任务与任务之间、任务与ISR之间的同步。其主要的特点是可实现一对多的同步。

- 事件也称为事件集：
  - 每个事件用一个bit位代表；
  - bit位置1代表相应的事件已经触发；
  - bit位置0代表相应的事件并未触发；

事件特性

- 单一事件可以看成是硬件当中一个水平触发的GPIO中断：
  - 当激活时，中断立刻触发；
  - 当IO水平依然拉高时，再进行触发将不起作用；



  - 事件集也类似：
    - 当事件位置位且未清除时，
    - 再进行触发将没有效用；

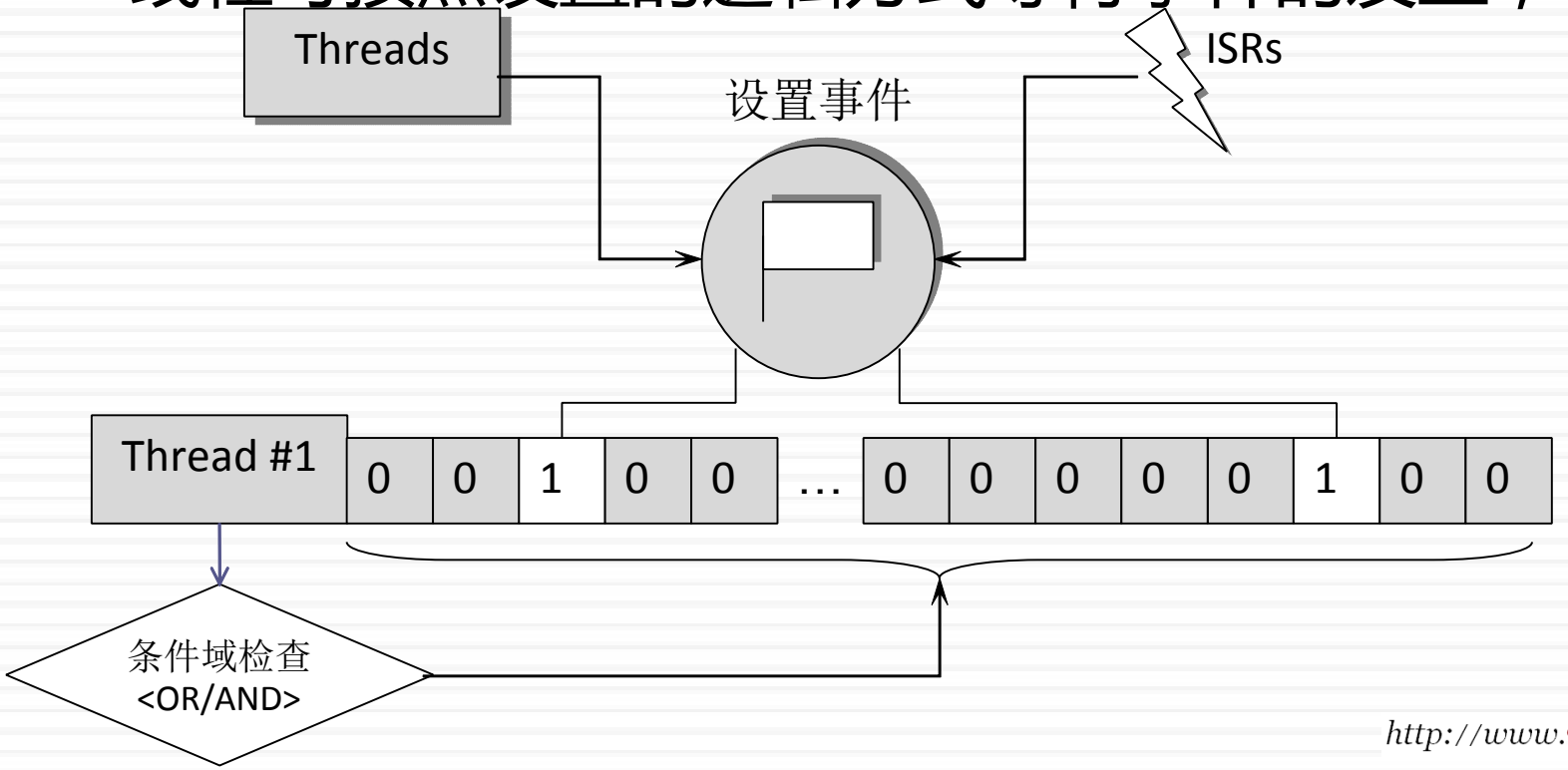
事件控制块


```
struct rt_event
{
    struct rt_ipc_object parent;
    rt_uint32_t set; /* 事件集合 */
};
/* rt_event_t是指向事件结构体的指针 */
typedef struct rt_event* rt_event_t;
```

- 事件从IPC类中进行派生；
- set是32位整数，一个事件对象总计可以支持32个事件。

事件工作示意图

- 线程或中断服务例程可以发送（设置）相应事件位；
- 线程可按照设置的逻辑方式等待事件的发生；



事件：创建

- 动态接口：
- `rt_event_t rt_event_create (const char* name, rt_uint8_t flag);`

- 静态接口：
- `rt_err_t rt_event_init(rt_event_t event, const char* name, rt_uint8_t flag);`

- 在使用一个事件集前，需求初始化事件对象或创建一个事件对象。flag的参数类似semaphore中的描述，允许取值： RT_IPC_FLAG_FIFO或RT_IPC_FLAG_PRIO；

事件：删除

- `rt_err_t rt_event_delete (rt_event_t event);`
- `rt_err_t rt_event_detach(rt_event_t event);`

- 当一个事件对象不再使用时，可以把它从系统对象容器中脱离或从系统中删除掉（释放事件对象占有的内存空间）；
- 如果事件对象的等待队列中有线程挂起，那么这些线程将都被唤醒，并返回-RT_ERROR错误。

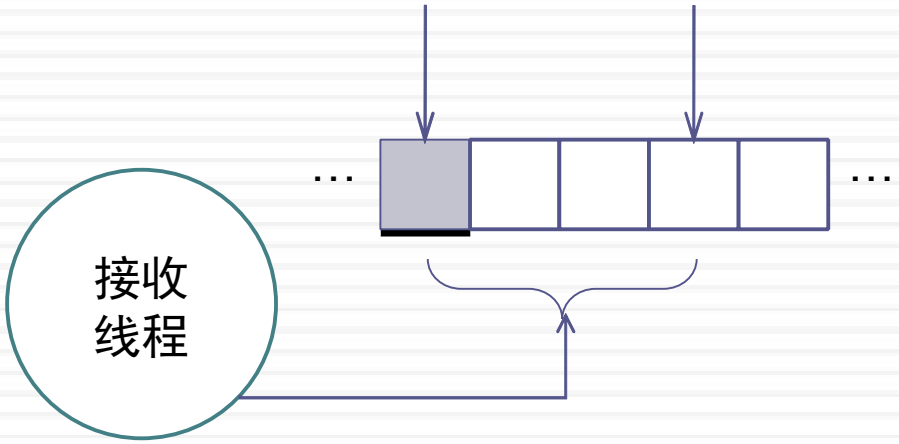
事件：接收

- `rt_err_t rt_event_recv(rt_event_t event, rt_uint32_t set, rt_uint8_t option, rt_int32_t timeout, rt_uint32_t* recved);`

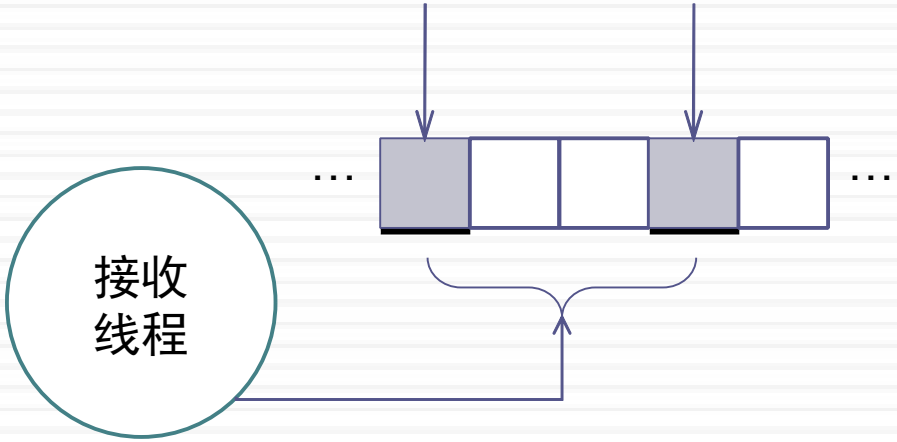
- 当线程需要接收事件（等待事件触发）时，它可以调用这个接口接收事件：
  - 参数set指明了它关心哪些事件（位）；
  - 参数option可取值：
    - RT_EVENT_FLAG_AND-- 逻辑与
    - RT_EVENT_FLAG_OR -- 逻辑或
    - RT_EVENT_FLAG_CLEAR -- 接收后将清除相应的事件位；
  - 参数timeout指定最大等待的时间；
  - 参数recved返回接收到的事件位；

事件：接收

- 逻辑或



- 逻辑与



事件：发送

- `rt_err_t rt_event_send(rt_event_t event, rt_uint32_t set);`

- 可以通过这个接口，把指定的事件发送（触发）到事件对象上；
- 如果存在线程等待在这个事件对象上，并且触发事件满足它关心的条件，这个线程将被唤醒；

典型使用场合

- 水平触发特性
  - Event中每个事件位仅代表置一或置零，并没有累加或排队效应，它可以应用于一些水平触发的例子上：
  - 例如：
    - 上层应用把数据报文放到一个缓冲队列中；
    - 然后通过发送事件的方式唤醒发送任务；
    - 如果发送任务此时正在处理报文，那么仅设置一个标志（事件位）表明有数据可能等待发送。

典型使用场合

- 多事件触发特性
  - Event由于其能够最大容纳32个事件位特性，它也适合于线程等待多事件触发场合。
  - 例如：
    - 对于应用线程来说，可能包括多种类型的数据，例如用户数据，控制相应，状态监控数据等；有时也包括一些针对发送线程的特殊操作，例如停止、启动等；
    - 每类数据可以采用不同的事件位标识，相应事件位触发代表不同的数据需要发送；
    - 发送线程采用逻辑或的方式等待接收事件，以达到一个线程等待多种事件的方式。

实验：事件的使用

- 会创建3个线程#1、#2、#3及初始化一个事件对象：
  - 线程 #1分别按照逻辑与、逻辑或的方式接收事件3、5的触发；
  - 线程 #2定时发送事件 3
  - 线程 #3定时发送事件 5

实验：事件的使用


```
#include <rtthread.h>

#define THREAD_STACK      512
#define THREAD_PRIORITY   20
#define THREAD_TIMESLICE  5

/* 事件控制块 */
static struct rt_event event;
```

实验：事件的使用


```
/* 线程1入口函数 */
static void thread1_entry(void *param)
{
    rt_uint32_t e;
    while (1){
        /* 接收第一个事件  */
        if (rt_event_recv(&event, ((1 << 3) | (1 << 5)),
                RT_EVENT_FLAG_AND | RT_EVENT_FLAG_CLEAR,
                RT_WAITING_FOREVER, &e) == RT_EOK) {
                rt_kprintf("t1: AND recv event 0x%x\n", e);
        }
        /* 等待10个OS Tick */
        rt_thread_delay(10);
        /* 接收第二个事件  */
        if (rt_event_recv(&event, ((1 << 3) | (1 << 5)),
                RT_EVENT_FLAG_OR | RT_EVENT_FLAG_CLEAR,
                RT_WAITING_FOREVER, &e) == RT_EOK){
                rt_kprintf("t1: OR recv event 0x%x\n", e);
        }
        rt_thread_delay(5);
    }
}
```

实验：事件的使用


```
/* 线程2入口函数 */
static void thread2_entry(void *param)
{
    while (1){
        /* 发送事件1 */
        rt_kprintf("t2: send event1\n");
        rt_event_send(&event, (1 << 3));
        /* 等待10个OS Tick */
        rt_thread_delay(10);
    }
}
/* 线程3入口函数 */
static void thread3_entry(void *param)
{
    while (1){
        /* 发送事件2 */
        rt_kprintf("t3: send event2\n");
        rt_event_send(&event, (1 << 5));
        /* 等待20个OS Tick */
        rt_thread_delay(20);
    }
}
```

实验：事件的使用


```
int rt_application_init()
{
    rt_thread_t tid;

    /* 初始化事件对象 */
    rt_event_init(&event, "event", RT_IPC_FLAG_FIFO);
    /* 创建线程1 */
    tid = rt_thread_create("t1", thread1_entry, RT_NULL,
        THREAD_STACK, THREAD_PRIORITY, THREAD_TIMESLICE);
    if (tid != RT_NULL) rt_thread_startup(tid);
    /* 创建线程2 */
    tid = rt_thread_create("t2", thread2_entry, RT_NULL,
        THREAD_STACK, THREAD_PRIORITY, THREAD_TIMESLICE);
    if (tid != RT_NULL) rt_thread_startup(tid);
    /* 创建线程3 */
    tid = rt_thread_create("t3", thread3_entry, RT_NULL,
        THREAD_STACK, THREAD_PRIORITY, THREAD_TIMESLICE);
    if (tid != RT_NULL) rt_thread_startup(tid);
    return 0;
}
```