

2019 TencentOS tiny 物联网操作系统

学习永无止境~

——杰杰

本讲义所有权归杰杰所有



关于我

一个走在物联网路上的小菜鸟~

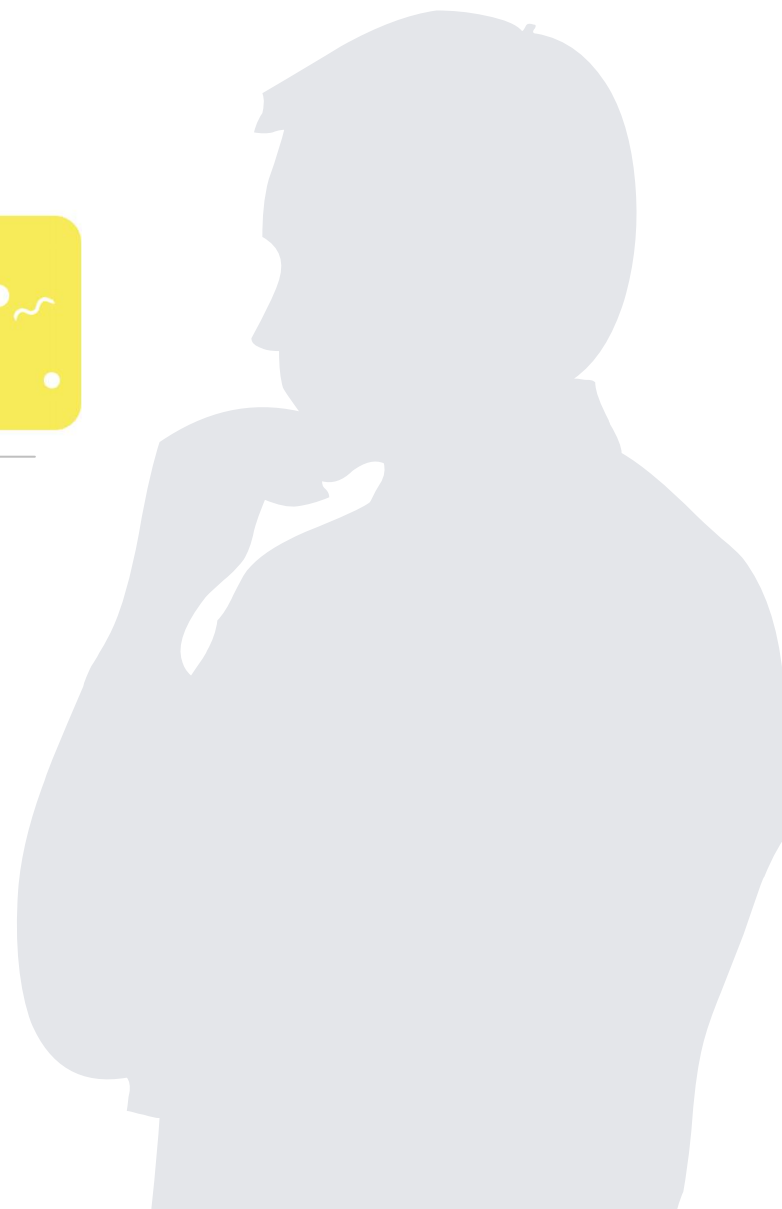
博客：<https://jiejietop.cn>

CSDN：<https://blog.csdn.net/jiejiemcu>

GitHub：<https://github.com/jiejieTop>



+ 个人公众号



目录

01

事件基本概念

02

事件数据结构

03

创建事件

04

销毁事件

05

等待事件

06

发送事件

01.事件基本概念

在操作系统中，事件是一种内核资源，主要用于任务与任务间、中断与任务间的同步，不提供数据传输功能！

与使用信号量同步有细微的差别：事件它可以实现一对多，多对多的同步。即一个任务可以等待多个事件的发生：可以是任意一个事件发生时唤醒任务进行事件处理；也可以是几个事件都发生后才唤醒任务进行事件处理。同样，也可以是多个任务同步多个事件。

每一个事件组只需要极少的RAM空间来保存事件旗标，一个事件（控制块）中包含了一个旗标，这个旗标的每一位表示一个“事件”，旗标存储在一个k_event_flag_t类型的变量中（名字叫flag，旗标简单理解就是事件标记变量），该变量在事件控制块中被定义，每一位代表一个事件，任务通过“逻辑与”或“逻辑或”与一个或多个事件建立关联，在事件发生时任务将被唤醒。

事件“逻辑或”是独立型同步，指的是任务所等待的若干事件中任意一个事件发生即可被唤醒；

事件“逻辑与”则是关联型同步，指的是任务所等待的若干事件中全部都发生时才被唤醒。

01.事件基本概念

事件是一种实现任务间通信的机制，可用于实现任务间的同步，但事件无数据传输。多任务环境下，任务、中断之间往往需要同步操作，一个事件发生会告知等待中的任务，即形成一个任务与任务、中断与任务间的同步。

事件无排队性，即多次向任务设置同一事件(如果任务还未来得及读走)，等效于只设置一次。

此外事件可以提供一对多、多对多的同步操作。

一对多同步模型：一个任务等待多个事件的触发，这种情况是比较常见的；

一对多同步模型：一个任务等待多个事件的触发，这种情况是比较常见的；

02.事件数据结构

事件控制块

TencentOS tiny 通过事件控制块操作事件，其数据类型为k_event_t，事件控制块由多个元素组成。

- pend_obj有点类似于面向对象的继承，继承一些属性，里面有描述内核资源的类型（如互斥锁、队列、互斥量等，同时还有一个等待列表list）。
- flag是旗标，一个32位的变量，因此每个事件控制块最多只能标识32个事件发生！

任务控制块与事件相关的数据结构

opt_event_pend、flag_expect、flag_match

在tos_config.h中，配置事件开关的宏定义是TOS_CFG_EVENT_EN

02.事件数据结构

在tos_event.h中，存在一些宏定义是用于操作事件的（opt选项）：

- TOS_OPT_EVENT_PEND_ANY：任务在等待任意一个事件发生，即“逻辑或”！
- TOS_OPT_EVENT_PEND_ALL：任务在等待所有事件发生，即“逻辑与”！
- TOS_OPT_EVENT_PEND_CLR：清除等待到的事件旗标，可以与TOS_OPT_EVENT_PEND_ANY、TOS_OPT_EVENT_PEND_ALL混合使用（通过“|”运算符）。

除此之外还有一个枚举类型的数据结构，用于发送事件时的选项操作，可以在发送事件时清除事件旗标的其他位（即覆盖，影响其他事件），也可以保持原本旗标中的其他位（不覆盖，不影响其他事件）。

OPT_EVENT_POST_KEP

OPT_EVENT_POST_CLR

03.创建事件

系统中每个事件都有对应的事件控制块，事件控制块中包含了事件的所有信息，比如它的等待列表、它的资源类型，以及它的事件旗标值

创建事件函数是`tos_event_create()`，传入一个事件控制块的指针`*event`，除此之外还可以指定事件初始值`init_flag`。

事件的创建实际上就是调用`pend_object_init()`函数将事件控制块中的`event->pend_obj`成员变量进行初始化，它的资源类型被标识为`PEND_TYPE_EVENT`。然后将`event->flag`成员变量设置为事件旗标初始值`init_flag`。

04.销毁事件

事件销毁函数是根据事件控制块直接销毁的，销毁之后事件的所有信息都会被清除，而且不能再次使用这个事件，当事件被销毁时，其等待列表中存在任务，系统有必要将这些等待这些任务唤醒，并告知任务事件已经被销毁了PENDING_STATE_DESTROY。然后产生一次任务调度以切换到最高优先级任务执行。

04.销毁事件

1. 调用pend_is_nopending()函数判断一下是否有任务在等待事件
2. 如果有任务在等待事件则调用pend_wakeup_all()函数将这些任务唤醒，并且告知等待任务事件已经被销毁了（即设置任务控制块中的等待状态成员变量pend_state为PEND_STATE_DESTROY）。
3. 调用pend_object_deinit()函数将事件控制块中的内容清除，最主要的是将控制块中的资源类型设置为PEND_TYPE_NONE，这样子就无法使用这个事件了。
4. 将event->flag成员变量恢复为默认值0。
5. 进行任务调度knl_sched()

05.等待事件

tos_event_pend()函数用于获取事件，通过这个函数，就可以知道事件旗标中的哪一位被置1，即哪一个事件发生了，然后任务可以对等待的事件指定“逻辑与”、“逻辑或”进行等待操作（opt_pend选项）。

并且这个函数实现了等待超时机制，且仅当任务等待的事件发生时，任务才能等待到事件。当事件未发生的时候，等待事件的任务会进入阻塞态，阻塞时间timeout由用户指定，在这段时间中，如果事件一直没发生，该任务将保持阻塞状态以等待事件发生。当其它任务或中断服务程序往其等待的事件旗标设置对应的标志位，该任务将自动由阻塞态转为就绪态。当任务等待的时间超过了指定的阻塞时间，即使事件还未发生，任务也会自动从阻塞态转移为就绪态。这样子很有效的体现了操作系统的实时性。

任务获取了某个事件时，可以选择清除事件操作

等待事件的操作不允许在中断上下文环境运行！

05.等待事件

1. 首先检测传入的参数是否正确。 , 注意opt_pend的选项必须存在TOS_OPT_EVENT_PEND_ANY 或者 TOS_OPT_EVENT_PEND_ANY 之一 , 且二者不允许同时存在 (互斥) 。
2. 调用event_is_match()函数判断等待的事件是否已发生 (即任务等待的事件与事件控制块中的旗标是否匹配) 。
3. 在event_is_match()函数中会根据等待选项opt_pend是等待任意一个事件 (TOS_OPT_EVENT_PEND_ANY) 还是等待所有事件 (TOS_OPT_EVENT_PEND_ANY) 做出是否匹配的判断 , 如果是匹配了则返回K_TRUE , 反之返回K_FALSE , 同时等待到的事件通过flag_match变量返回 (已发生匹配) 。对于等待所有时间的选项 , 当且仅当所有事件都发生是才算匹配 : (event & flag_expect) == flag_expect) , 对于等待任意一个事件的选项 , 有其中一个事件发生都算匹配 : (event & flag_expect) 。
4. 如果事件未发生则可能会阻塞当前获取的任务 , 看一下用户指定的阻塞时间timeout是否为不阻塞TOS_TIME_NOWAIT , 如果不阻塞则直接返回K_ERR_PEND_NOWAIT错误代码。
5. 如果调度器被锁了knl_is_sched_locked() , 则无法进行等待操作 , 返回错误代码K_ERR_PEND_SCHED_LOCKED , 毕竟需要切换任务 , 调度器被锁则无法切换任务。

05.等待事件

6. 将任务控制块中关于事件的变量设置一下，即设置任务期望等待的事件`k_curr_task->flag_expect`，任务匹配的事件`k_curr_task->flag_match`，以及任务等待事件的选项`k_curr_task->opt_event_pend`。
7. 调用`pend_task_block()`函数将任务阻塞，该函数实际上就是将任务从就绪列表中移除`k_rdyq.task_list_head[task_prio]`，并且插入到等待列表中`object->list`，如果等待的时间不是永久等待`TOS_TIME_FOREVER`，还会将任务插入时间列表中`k_tick_list`，阻塞时间为`timeout`，然后进行一次任务调度`kn1_sched()`。
8. 当程序能继续往下执行时，则表示任务等待到事件，又或者等待发生了超时，任务就不需要等待事件了，此时将任务控制块中的内容清空，即清空任务期望等待的事件`k_curr_task->flag_expect`，任务匹配的事件`k_curr_task->flag_match`，以及任务等待事件的选项`k_curr_task->opt_event_pend`，同时还调用`pend_state2errno()`函数获取一下任务的等待状态，看一下是哪种情况导致任务恢复运行，并且将结果返回给调用等待事件函数的任务。

注意：当等待事件的任务能从阻塞中恢复运行，也不一定是等待到事件发生，也有可能是发生了超时，因此在写程序的时候必须要判断一下等待的事件状态，如果是`K_ERR_NONE`则表示获取成功！

06.发送事件

TencentOS tiny 提供两个函数发送事件，分别是：`tos_event_post()`与`tos_event_post_keep()`，两个函数本质上都是调用同一个函数`event_do_post()`去实现发送事件的操作的，只不过选项是不同而已，使用`tos_event_post()`函数会覆盖写入指定的事件，可能影响其他已发生的事件，而`tos_event_post_keep()`函数则可以保持其他事件位不改变的同时发生事件，在实际情况中后者更常用。

此函数用于将已发生的事件写入事件旗标中指定的位，当对应的位被置1之后，等待事件的任务将可能被恢复，此时需要遍历等待在事件对象上的事件等待列表，判断是否有任务期望的事件与当前事件旗标的值匹配，如果有，则唤醒该任务。

简单来说，就是设置自己定义的事件标志位为1，并且看看有没有任务在等待这个事件，有的话就唤醒它。

06.发送事件

TencentOS tiny 中设计的很好的地方就是简单与低耦合，这两个api接口本质上都是调用event_do_post()函数去发生事件，只是通过opt_post参数不同选择不同的处理方法。

1. 首先判断一下发生事件的方式opt_post，如果是OPT_EVENT_POST_KEP则采用或运算 “|”写入事件旗标，否则直接赋值。
2. 使用TOS_LIST_FOR_EACH_SAFE遍历等待在事件对象上的事件等待列表，通过event_is_match()函数判断是否有任务期望的事件与当前事件旗标的值匹配，如果有则调用pend_task_wakeup()函数唤醒对应的任务。
3. 如果唤醒的等待任务指定了清除对应的事件，那么将清除事件的旗标值。
4. 最后进行一次任务调度knl_sched()。

09. 实验

代码获取：<https://github.com/jiejieTop/TencentOS-Demo>

或者关注公众号，在后台回复“19”



jiejieTop / TencentOS-Demo

Unwatch 1 Star 2 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

This is the TencentOS tiny Demo that was ported on the embedfire stm32f103 board. Edit

Manage topics

5 commits 1 branch 0 releases 1 contributor BSD-3-Clause

Branch: master New pull request Create new file Upload files Find File Clone or download

jiejieTop add ICP demo Latest commit 9c71463 1 hour ago

01-task	add ICP demo	1 hour ago
02-queue	add ICP demo	1 hour ago
03-msg_queue	add ICP demo	1 hour ago
04-sem	add ICP demo	1 hour ago
05-mutex	add ICP demo	1 hour ago
06-event	add ICP demo	1 hour ago
hello-world	add hello-world demo	2 days ago
stm32f1-demo	add stm32f1-demo source	2 days ago
.gitignore	add ICP demo	1 hour ago
LICENSE	Initial commit	2 days ago
README.md	update README.md	2 days ago
keilkill.bat	add ICP demo	1 hour ago



THANKS

