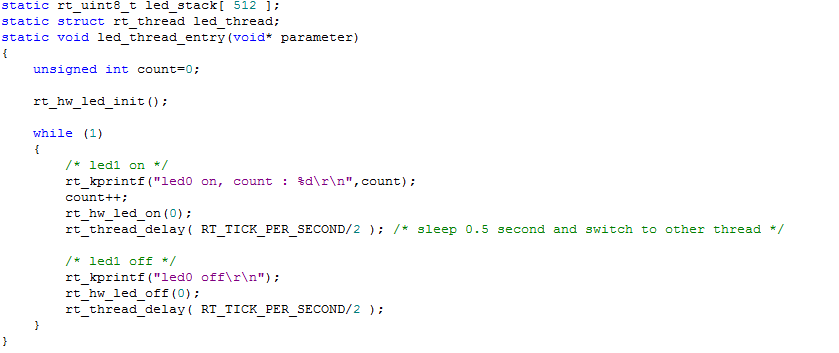
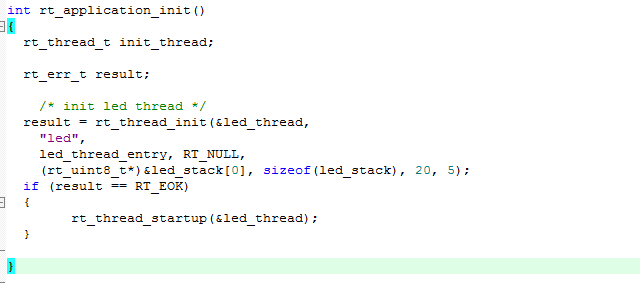
**RTT移植至stm32开发板**

在keil下搭建RT-thread最小系统工程\_CN,详情参见pdf。

**RTT创建线程**

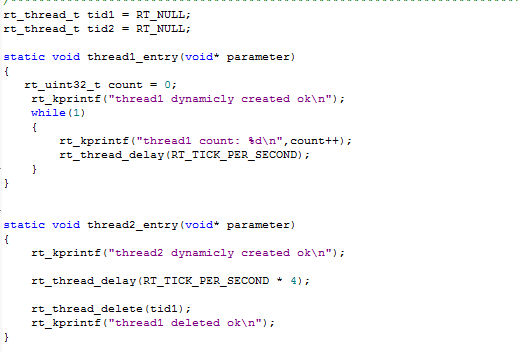
**静态线程的创建**

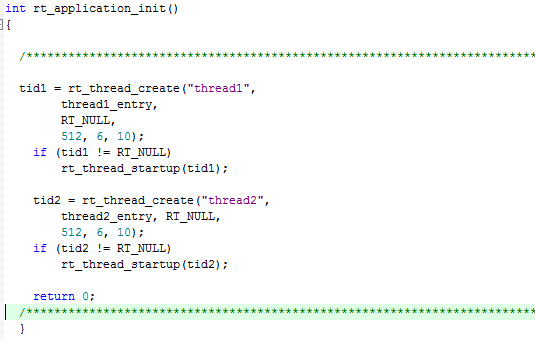


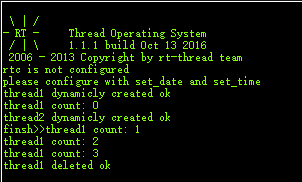


**创建动态线程**

1. 命名变量
2. 书写线程（thread——entry）
3. 在int rt\_application\_init() 函数中使用rt\_thread\_create()创建动态线程







**RTT线程管理**

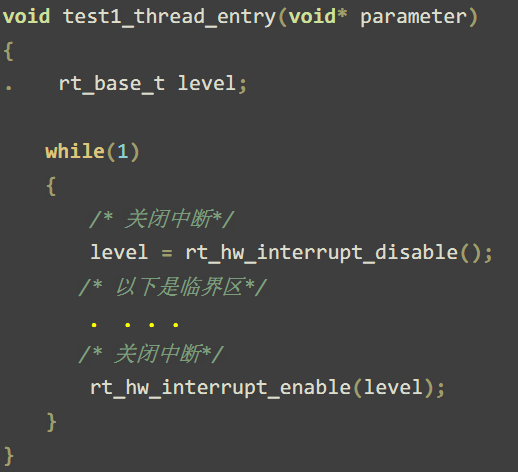
Rtt中线程可分为4种状态： 初始态，就绪态，运行态，挂起态

1. 使用rt\_thread\_create/init()函数创建动态/静态线程后，函数自动进入初始状态。
2. 调用rt\_thread\_startup()函数使线程进入就绪状态。
3. rt\_thread\_delete()函数将线程删除，使函数重新进入初始态。
4. rt\_thread\_delay()函数将线程挂起，使cpu执行其他程序，从而起到延时作用。
5. rtt函数优先级支持0-255，0为最高优先级。
6. 优先级高者抢占cpu，先执行，优先级相同时，cpu执行一个程序的时间长度由函数的时间片参数决定，顺序采用轮询的方式。如果函数在一个时间片内未执行完，下次执行时继续执行
7. 当线程a调用rt\_thread\_yield()函数时线程将让出CPU的使用权并保持当前执行状态（下次调度该线程时，从此线程中rt\_thread\_yield()函数后一句程序开始执行），若存在同优先级线程，线程将被置于线程控制块中同优先级线程的最后，若不存在同优先级线程，当前除此线程外的最高优先级线程（线程b）获得CPU的使用权，执行完线程b后，执行线程a。

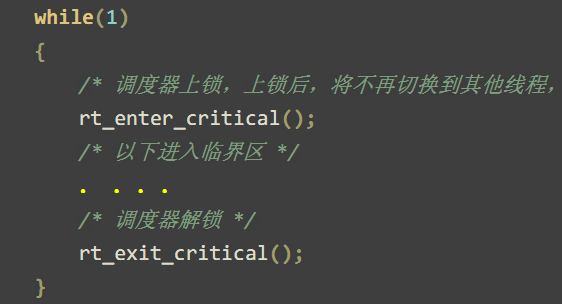
**RTT线程间同步与通信**

为解决公共资源在多个线程间公用导致的输出结果不正确的问题，引入线程间通信机制（IPC机制）。

1. 关闭中断：使用rt\_hw\_interrupt\_disable();与rt\_hw\_interrupt\_enable();函数在公用资源使用时，关闭中断，禁止系统的调度，从而保证线程不被打断。



1. 把调度器锁住也能让当前运行的任务不被换出，直到调度器解锁。但和关闭中断有一点不相同的是，对调度器上锁，系统依然能响应外部中断，中断服务例程依然有可能被运行。所以在使用调度器上锁的方式来做任务同步时，需要考虑好，任务访问的临界资源是否会被中断服务例程所修改，如果可能会被修改，那么将不适合采用此种方式作为同步的方法。 rt\_enter\_critical(); rt\_exit\_critical(); 共同使用



（3）rtt中的IPC对象包括：信号量、互斥锁、事件、消息队列、

邮箱。

（4）信号量：给予公用资源一个非负整数（即信号量，它的值由公

用资源可以同时被几个线程占用所决定），当公用资源被一个线

程占用，信号量便减一，当信号量减为0时，公用资源不能再被

其它线程占用。

信号量有静态与动态之分:

初始化—rt\_sem\_init()（对应静态信号量）；

建立—rt\_sem\_create()（对应动态信号量）；

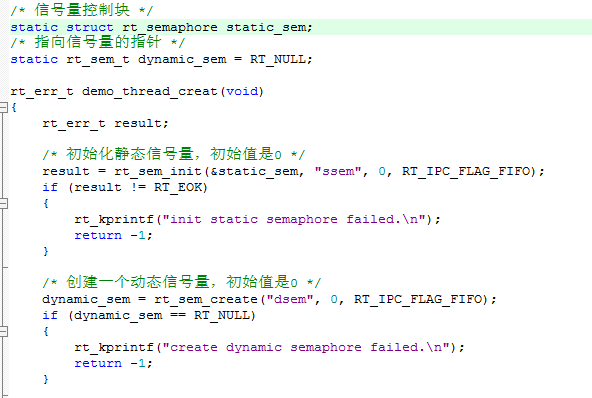
获取—rt\_sem\_take()；

释放—rt\_sem\_release()；

脱离—rt\_sem\_detach()（对应静态信号量）；

删除—rt\_sem\_delete()（对应动态信号量）；

创建一个信号量：



初始化信号量，信号量的值为0。

