**RT-Thread常见问题**

**1.**我创建了一个线程，然后在finsh中用list\_thread(),结果这个线程的error总是-2，也就是-RT\_ETIMEOUT，线程是个while循环，执行一段程序，然后rt\_thread\_delay(2)。程序运行没发现问题，可是list\_thread()总是显示错误呢？

因为程序在执行rt\_thread\_delay/rt\_thread\_sleep函数时，当指定的时间超时结束时，系统会把这个线程给唤醒，然后统一对这类线程赋了一个-RT\_ETIMEOUT的错误号。唤醒的线程继续执行后续的rt\_thread\_delay/rt\_thread\_sleep函数，在末尾的时候会把线程本身的错误号清成0。所以，有的时候会发现错误号是-RT\_ETIMEOUT，但它并不一定代表是错误。

**2.**rt\_kprintf()是RTT控制台的主要输出手段，也可在自行编写的应用中输出调试信息、运行状态等信息，很方便。 但是该函数的缓冲区“static char rt\_log\_buf[RT\_CONSOLEBUF\_SIZE]”并未做独占保护，任何对 rt\_kprintf()的调用都将直接通过vsnprintf（）将新数据覆盖到rt\_log\_buf内。上述问题在多任务调用时应会出现。   
是否考虑使用信号量？

rt\_kprintf允许在中断服务例程中使用，并且是“即时”打印出来，所以不能够使用信号量。

rt\_kprintf使用了一个静态的数组(rt\_log\_buf)做为缓冲，可以对它进行关中断保护，但是这样对系统的性能会影响非常大，所以 rt\_kprintf被\*特别实现\*成一个完全不关中断的log输出。只有当用户确实需要输出严格一致的log时(例如用于测试)，可以自行关闭中断。

**3**.程序开始运行时串口输出： (thread->stat == RT\_THREAD\_INIT) assert failed at rt\_thread\_startup:200   
这句是什么原因造成的？200代表什么？

  rt\_thread\_startup是函数名 200是文件中的行号，即thread.c中的第200行。当出现了这类assert问题时，可以在相应的函数、相应的行号中查找是因为什么导致代码不能执行通过，一般都是OS的代码检测到错误而停下来了。

**4.**线程退出或删除，它真正是在哪里删除的？

线程自动运行结束退出后，线程状态会变成init状态不再参与调度，当系统空闲调度到idle线程时，idle线程会做真正的删除动作，包括线程的TCB和栈。线程删除的过程也类似，先更改为init状态，然后由idle线程做真正的删除动作。

**5.**什么是动态线程？什么事静态线程？两者有什么区别？

RT-Thread中支持静态和动态两种定义方式。用线程来举例的话，rt\_thread\_init对应静态定义方式，rt\_thread\_create对应动态定义方式。

使用静态定义方式时，必须先定义静态的线程控制块，并且定义好堆栈空间，然后调用rt\_thread\_init来完成线程的初始化工作。采用这种方式，线程控制块和堆栈占用的内存会放在RW段，这段空间在编译时就已经确定，它不是可以动态分配的，所以不能被释放，而只能使用rt\_thread\_detach函数将该线程控制块从对象管理器中脱离。

使用动态定义方式rt\_thread\_create时，RT-Thread会动态申请线程控制块和堆栈空间。在编译时，编译器是不会感知到这段空间的，只有在程序运行时，RT-Thread才会从系统堆中申请分配这段内存空间，当不需要使用该线程时，调用rt\_thread\_delete函数就会将这段申请的内存空间重新释放到内存堆中。

这两种方式各有利弊，静态定义方式会占用RW/ZI空间，但是不需要动态分配内存，运行时效率较高，实时性较好。 动态方式不会占用额外的RW/ZI空间，占用空间小，但是运行时需要动态分配内存，效率没有静态方式高。

总的来说，这两种方式就是空间和时间效率的平衡，可以根据实际环境需求选择采用具体的分配方式。