**任务4-4 线程主动让出CPU资源**

任务描述：在任务4-3中，我们发现，当LED和BEEP两个线程的优先级别一样时，它们轮流使用CPU资源，且每次使用CPU资源时大概打印出2次日志（相当于执行一次时间片大概可以执行2次循环体）。读者也应该已经发现，程序打印中存在混乱，这主要是由于线程在打印字符串时，在字符串还没有输出完整时，线程由于时间片用完而被强制挂起而导致的。

本任务，我们让线程一次时间片只执行一次循环体后主动让出CPU资源。

一、编写代码

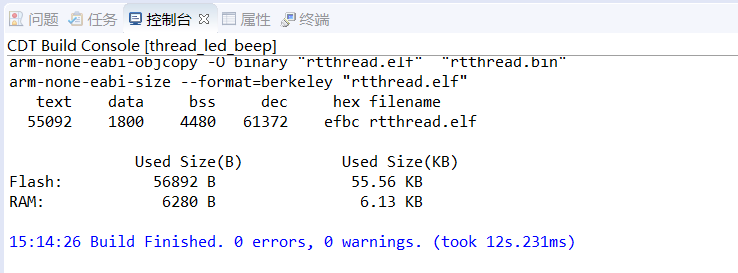
复制“操作指导\4-4”目录下的main.c文件到项目到applications目录下，

三、编译下载

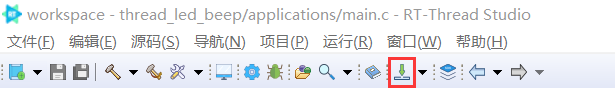
1、编译，点击下图构建按钮重新编译



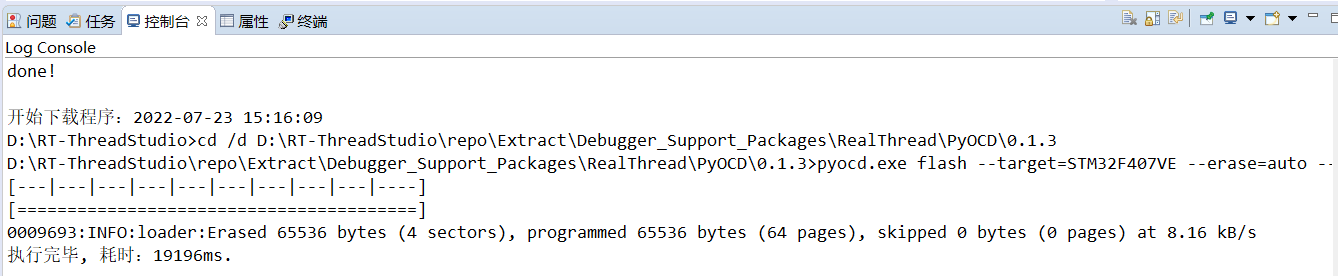
2、编译完成后，如下所示：



3、下载，点击如下图中的下载按钮进行程序下载



下载完成结果如下：



**四、程序测试**

（1）下载并启动系统，观察到终端打印如图4-15（a）所示，虽然在创建线程时，LED和BEEP线程的优先级不一样，但在线程启动后，由于main程序通过rt\_thread\_control()函数把两个线程的优先级重新设置为相同的值21，使得两个线程的运行优先级一样。由于两个线程的优先级相同，所以两个线程通过时间片轮流使用CPU资源。

又由于两个线程都在打印日志信息后通过rt\_thread\_yield()主动让出CPU资源，线程的时间片足够打印一行信息，剩下的时间把CPU让出，让下一个具有相同优先级的线程使用。因此观察到两个线程轮流打印日志信息。

（2）修改代码中的序处理宏定义如下：

//#define THREAD\_CONTROL

#define USE\_THREAD\_YIELD

构建后下载并启动系统，观察到终端打印如图4-15（b）所示，虽然BEEP线程在打印完日志后主动让出PCU资源，但由于BEEP线程的优先级19比LED线程的优先级20要高，系统当前就绪队列的最高优先级为19，调度器在最高优为19的就绪队列中只找到BEEP线程，因此，BEEP线程让出CPU资源后，又重新得到运行。

直到BEEP线程退出，系统当前就绪队列的最高优先级变为20，LED线程才可到执行。因此先观察到BEEP线程的日志信息，等BEEP线程退出后，才观察到LED线程的日志信息。

（3）修改代码中的序处理宏定义如下：

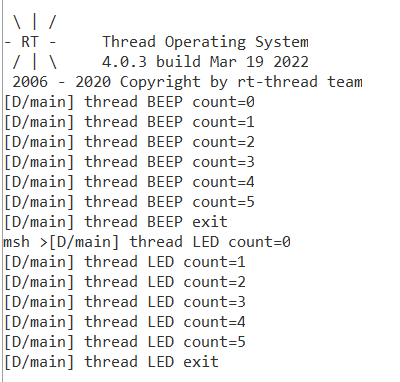
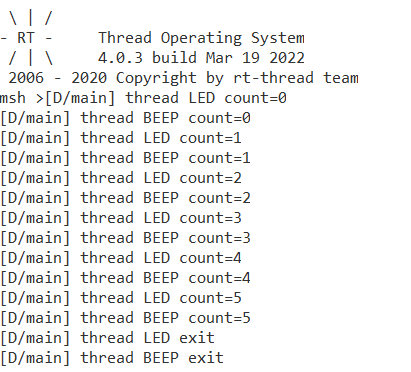
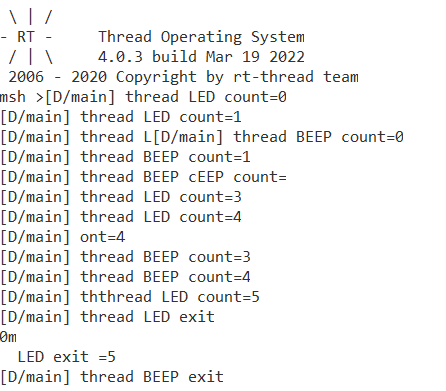
#define THREAD\_CONTROL

//#define USE\_THREAD\_YIELD

上述宏定义把BEEP线程和LED线程的优先级都为21，用rt\_schedule()函数代替rt\_thread\_yield()。

构建后下载并启动系统，观察到终端打印如图4-15（c）所示，我们发现，线程并没像（a）那样，每输出一行信息就让出CPU。

调用rt\_schedule()函数不能成功让出CPU的原因是，rt\_schedule()只是让调度器重新选择一个最高优先级的就绪线程来执行，并不影响线程在就绪队列中的位置。对于本例，当前系统就绪线程中，最高优先级为21，当优先级为21的LED/BEEP线程调用rt\_schedule()时，此时调度器找不到更高优先级的线程，且此时LED/BEEP线程时间片还没有用完，所以继续运行LED/BEEP线程。

（a） (b) (c)

图4-15 线程让出CPU实验