FreeRTOS任务调度

启动第一个任务流程：

创建开始任务：start_tsak

启动任务调度器：vTaskStartScheduler

1.创建空闲任务：prvIdleTask
2.创建软件定时器任务：xTimerCreateTimerTask
3.关中断（在启动第一个任务时开启）
4.初始化一些全局变量...
5.初始化任务运行时间统计功能的时基定时器
6.调用函数xPortStartScheduler完成启动任务调度器
1.配置 PendSV 和 SysTick 的中断优先级为最低优先级
2.调用函数 vPortSetupTimerInterrupt()配置 SysTick
3.调用函数 prvEnableVFP()使能 FPU
4.将 FPCCR 寄存器的[31:30]置 1，这样在进出异常时， FPU 的相关寄存器就会自动地保存和恢复
5.调用函数 prvStartFirstTask()启动第一个任务
1.复位MSP初始值
2.使能中断
3.触发SVC中断
获取当前优先级最高的任务控制块pxCurrentTCB
将该任务的寄存器值出栈至CPU寄存器中
设置PSP
返回r14　　执行第一个任务函数

任务切换流程

触发PendSV中断，主要途径

1、滴答定时器中断触发
2、调用FreeRTOS的API函数触发，如：portYIELD( )

PendSV中断函数触发，执行任务切换

当前的psp是正在运行的任务的栈指针，读取当前psp进程指针，存入r0
压栈（保存现场）
获取当前最高优先级任务的任务控制块
出栈（恢复现场）
更新切换后的任务的的栈指针给PSP
bx r14 执行新任务函数

1，开启任务调度器（熟悉）

函数：vTaskStartScheduler()，用于启动任务调度器，任务调度器启动后， FreeRTOS便会开始进行任务调度

1、创建空闲任务
2、如果使能软件定时器，则创建定时器任务
3、关闭中断，防止调度器开启之前或过程中，受中断干扰，会在运行第一个任务时打开中断
4、初始化全局变量，并将任务调度器的运行标志设置为已运行
5、初始化任务运行时间统计功能的时基定时器
6、调用函数 xPortStartScheduler()

xPortStartScheduler()该函数用于完成启动任务调度器中与硬件架构相关的配置部分，以及启动第一个任务

1、检测用户在 FreeRTOSConfig.h 文件中对中断的相关配置是否有误
2、配置 PendSV 和 SysTick 的中断优先级为最低优先级
3、调用函数 vPortSetupTimerInterrupt()配置 SysTick
4、初始化临界区嵌套计数器为 0
5、调用函数 prvEnableVFP()使能 FPU
6、调用函数 prvStartFirstTask()启动第一个任务

2，启动第一个任务（掌握）

想象下应该如何启动第一个任务？

1、找到优先级最高的任务
2、将该任务的寄存器值恢复到CPU寄存器中

2.1，prvStartFirstTask () 该函数用于初始化启动第一个任务前的环境，主要是重新设置MSP 指针，并使能全局中断

1、什么是MSP指针？
主堆栈指针（MSP）：它由 OS 内核、异常服务例程以及所有需要特权访问的应用程序代码来使用。
进程堆栈指针（PSP）：用于常规的应用程序代码（不处于异常服用例程中时）。
注意：在FreeRTOS中，中断使用MSP（主堆栈），中断以外使用PSP（进程堆栈）

2、为什么是 0xE000ED08?
因为需从 0xE000ED08 获取向量表的偏移，为啥要获得向量表呢？因为向量表的第一个是 MSP 指针！
取 MSP 的初始值的思路是先根据向量表的位置寄存器 VTOR (0xE000ED08) 来获取向量表存储的地址；在根据向量表存储的地址，来访问第一个元素，也就是初始的 MSP

2.2，vPortSVCHandler () /* SVC中断服务函数 */

1. 通过 pxCurrentTCB 获取优先级最高的就绪态任务的任务栈地址，优先级最高的就绪态任务是系统将要运行的任务 。
2. 通过任务的栈顶指针，将任务栈中的内容出栈到 CPU 寄存器中，任务栈中的内容在调用任务创建函数的时候，已初始化，然后设置 PSP 指针 。
3. 通过往 BASEPRI 寄存器中写 0，允许中断。
4、执行 bx R14，告诉处理器 ISR 完成，需要返回，此刻处理器便会使用 PSP 做为堆栈指针，进行出栈操作，将xPSR、PC、LR、R12、R3~R0 出栈，初始化的时候，PC被我们赋值成为了执行任务的函数的入口，所以呢，就正常跳入到了优先级最高的就绪状态的第一个任务的入口函数了

注意
返回R14 R14 是链接寄存器 LR，在 ISR 中（此刻我们在 SVC 的 ISR 中），它记录了异常返回值 EXC_RETURN
SVC中断只在启动第一次任务时会调用一次，以后均不调用

3，任务切换（掌握）

任务切换的本质：就是CPU寄存器的切换。

假设当由任务A切换到任务B时，主要分为两步：
第一步：需暂停任务A的执行，并将此时任务A的寄存器保存到任务堆栈，这个过程叫做保存现场；
第二步：将任务B的各个寄存器值（被存于任务堆栈中）恢复到CPU寄存器中，这个过程叫做恢复现场；
对任务A保存现场，对任务B恢复现场，这个整体的过程称之为：上下文切换

通过函数vTaskSwitchContext（）查找最高优先级任务
利用函数taskSELECT_HIGHEST_PRIORITY_TASK( )，前导置零指令找到最高优先级
利用函数listGET_OWNER_OF_NEXT_ENTRY( ) ，获取最高优先级的任务句柄

任务切换的整个过程：

1.mrs r0, psp　　把psp存到r0，当前的psp是任务A的栈指针，这时的psp指向的是任务栈A中的R0
2. ldr r3, =pxCurrentTCB
ldr r2, [r3]　　获取当前运行任务的栈顶地址即R2保存的栈顶地址，注意R3等于pxCurrentTCB的地址
3.stmdb r0!, {r4-r11, r14}　　压栈，从上往下压，将r0的值，当压栈的起始地址，开始压栈（保存现场）
4.str r0, [r2]　　将r0的值(前面的底部地址)，写到r2地址所指向的内存中（即栈顶地址指向的内存，pxTopOfStack中）
5.stmdb sp!, {r0, r3}　　/* 把R0和R3的值压入MSP */
6.bl vTaskSwitchContext　　通过该函数，获取下一个执行任务的任务控制块，赋值给pxCurrentTCB
7.ldmia sp!, {r0, r3}　　从sp(MSP)恢复r3，即把r3恢复成&pxCurrentTCB。后续就可以利用r3得到新的任务控制块了。
8.ldmia r0!, {r4-r11, r14}　　出栈，以寻址地址开始，从下往上进行出栈，将保存在这些地址的值恢复到寄存器里边去
9.msr psp, r0
bx r14　　①将r0更新给psp线程堆栈
②返回线程模式，执行新任务