FreeRTOS内核笔记（一）：基本知识和命名规则

FreeRTOS　内核笔记（一）：基本知识和命名规则

FreeRTOS内核笔记

查看库存笔记发现 之前学习FreeRTOS 内核的笔记，趁着休假整理下发出来：

简单来说，FreeRTOS是一种多任务 实时操作系统

命名规则：

x : portBASE_TYPE类型变量,数据结构，任务句柄，队列句柄等定义的变量名.
p : 指针变量的前缀.
prv（private）：私有函数的前缀（加上了 static）
e: 枚举变量会加上前缀 e。
v: 即 void 类型，函数名称前加上字母 v，则代表函数返回void。函数前加x则代表函数返回x类型。
在函数名/宏定义　中加入了函数所在的文件名：如

1. vTaskPrioritySet()函数的返回值为 void 型，在 task.c这个文件中定义。
2. xQueueReceive()函数的返回值为 portBASE_TYPE 型，在 queue.c 这个文件中定义。
3. vSemaphoreCreateBinary()函数的返回值为 void 型，在 semphr.h 这个文件中定义。
4. pxPortInitialiseStack（），这个函数前缀是px，表示函数的返回值是结构体类型的指针，并且这个函数是port文件，作用是初始化栈

c	函数返回值（变量值）类型是char类型
s	函数返回值（变量值）类型是short类型
l	函数返回值（变量值）类型是long类型
x	函数返回值（变量值）类型是数据结构，任务句柄，队列句柄等定义的变量名类型
u	函数返回值（变量值）类型是无符号类型
p	函数返回值（变量值）类型是指针类型
prv	函数是私有函数，不能被外界调用。
v	函数返回值类型是void类型

详细内容可参见FreeRTOS内核说明文档（中文）：https://download.csdn.net/download/howiexue/9978417

Variable Names

Variables are pre-fixed with their type. 'c' for char, 's' for short, 'l' for long and 'x' for portBASE_TYPE and any other type (structures, task handles, queue handles, etc.).

If a variable is unsigned it is also prefixed with a 'u'. If a variable is a pointer it is also prefixed with a 'p'. Therefore a variable of type unsigned char will be prefixed with 'uc', and a variable of type pointer to char will be prefixed 'pc'.

Function Names

Functions are prefixed with both the type they return and the file they are defined within. For example:

- vTaskPrioritySet() returns a void and is defined within task.c.
- xQueueReceive() returns a variable of type portBASE_TYPE and is defined within queue.c.
- vSemaphoreCreateBinary() returns a void and is defined within semphr.h.

File scope (private) functions are prefixed with 'prv'.

常用宏定义

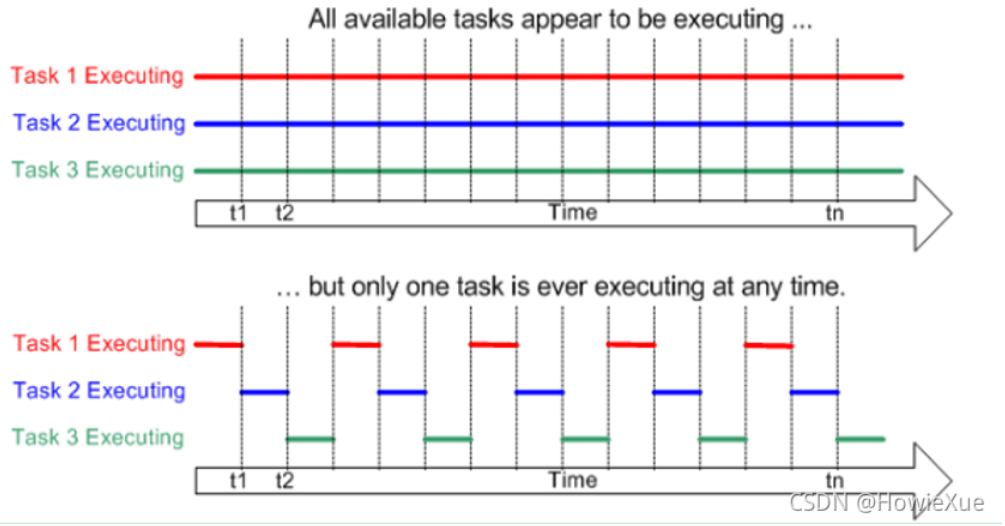
Table 23 Macro prefixes

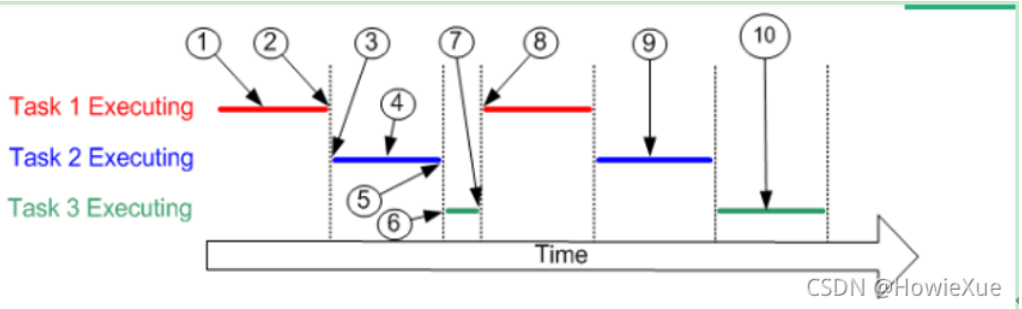
Prefix	Location of macro definition
port (for example, portMAX_DELAY)	portable.h
task (for example, taskENTER_CRITICAL())	task.h
pd (for example, pdTRUE)	projdefs.h
config (for example, configUSE_PREEMPTION)	FreeRTOSConfig.h
err (for example, errQUEUE_FULL)	projdefs.h

Table 24 Common macro definitions

Macro	Value
pdTRUE	1
pdFALSE	0
pdPASS	1
pdFAIL	0

Thread运行状态：





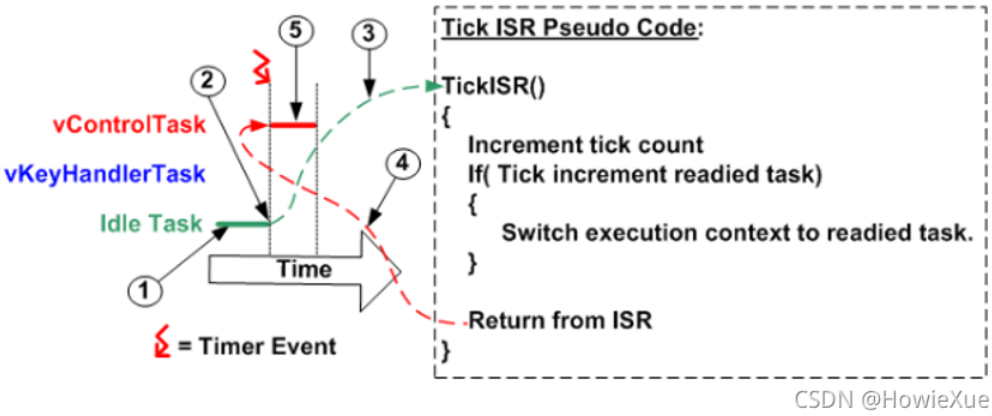
1. Task1正在运行
2. 内核挂起Task1（suspended）
3. 恢复任务Task2
4. Task2正在执行
5. 内核挂起Task2
6. 恢复Task3
7. Task3视图访问同样的处理器外设，发现它被锁定，Task3不能继续，所以自己挂起自己。

8. 内核回复Task1

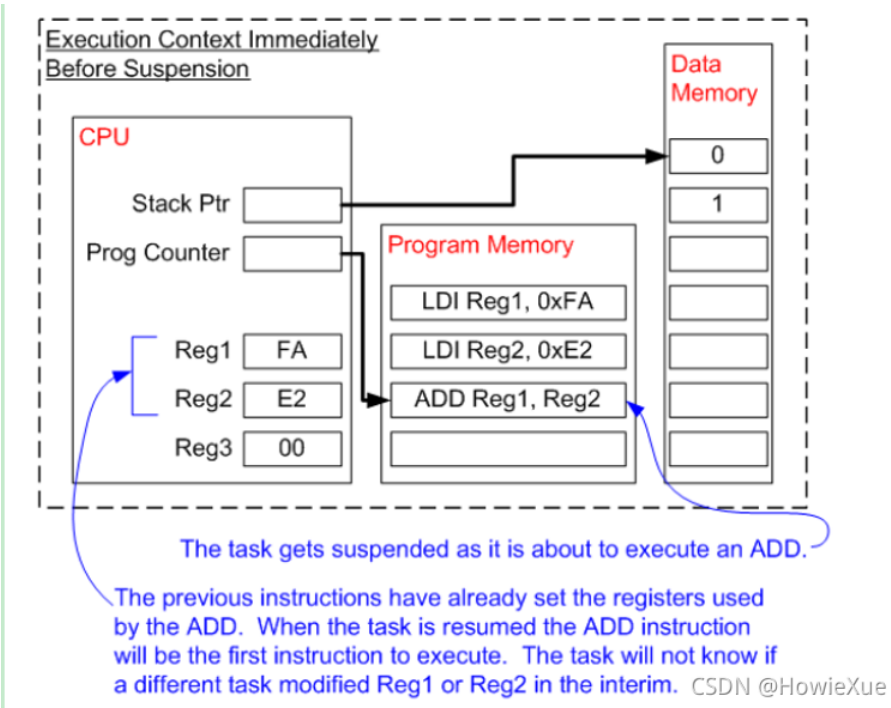
RTOS Tick

RTOS使用tick count变量来度量时间。Tick值通过定时器中断来累加，每次tick增加后，Kernel都会检查看现在是否唤醒、阻塞一个任务。

睡眠时，一个task将指定多长时间后它会醒来。
阻塞时，一个任务将指定一个希望等待多久的时间。

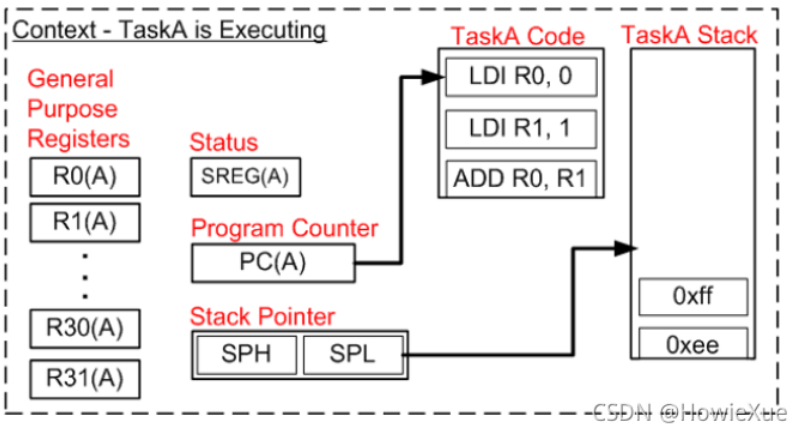


Context切换:

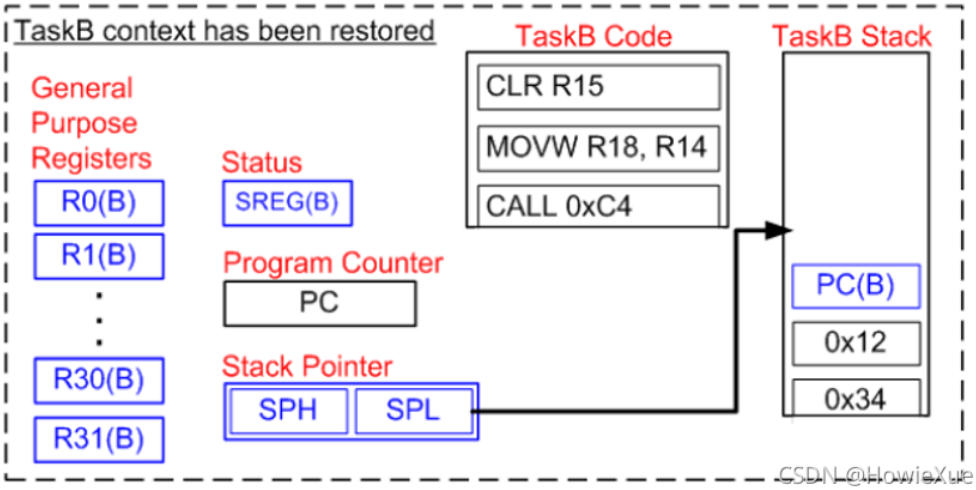


每个Task有不同的Stack空间，用来保存上下文和数据处理。

TaskA 的上下文如下图所示：

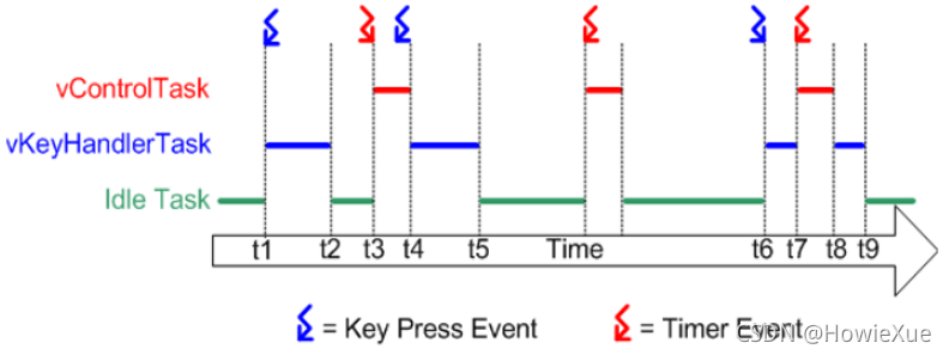


Tick 的ISR修改Stack Pointer 指向了TaskB的stack，TaskB的栈顶指针被取得：TaskB任务恢复



实时调度器Scheduler

Scheduler是RTOS的核心，FreeRTOS自己有idle Task，就是在没有其他Task执行的时候，会一直执行idle Task



1. 在t1时刻，键盘按下（事件）出现，则vKeyHandlerTask任务现在可以执行，因为它比idle Task有更高的优先级，所以CPU事件给它
2. 在t2时刻，vKeyHandlerTask已经完成了对按键的处理，它不能继续执行（直到另一个按键按下），所以它必须挂起自己，idle task恢复执行
3. 在t3时刻，定时器时间到，执行vControlTask，并且作为最高优先级被立即Scheduled
4. 在t3和t4之间，当vControlTask还在执行的时候，一个键按下，vKeyHandlerTask不能被执行，因为优先级比Control低，
5. 在t4时刻，vControlTask完成处理，将自己挂起，而这时vKey Task现在是最高优先级的Task，这时才被cpu scheduled运行