时间片的初步实现

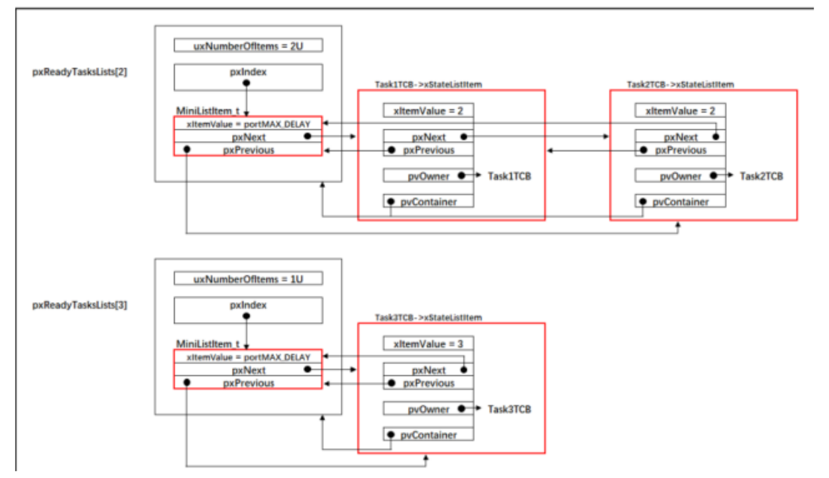
基本概念

所谓时间片就是同一个优先级下可以有多个任务，
每个任务轮流地享有CPU使用权，享有CPU的时间我们叫时间片。

实现方法

关键点在于我们切换任务的一个细节

根据前面所述：我们创建了任务就绪列表，用于管理不同优先级的任务，在同一优先级之下，我们可能会挂载多个任务（左图1）同时我们已经配置好了sysTick中断，在中断当中我们会更新时间，然后判断是否有任务需要解锁，随后根据优先级的高低比较并进行任务的切换



1、优先级2的就绪列表下，挂载了两个任务，优先级3的就绪列表下挂载一个任务

```c
/* 将任务从延时列表移除，消除等待状态 */
( void ) uxListRemove( &( pxTCB->xStateListItem ) );

/* 将解除等待的任务添加到就绪列表 */
prvAddTaskToReadyList( pxTCB );

#if (  configUSE_PREEMPTION == 1 )
{   /*判断阻塞到期新解锁的任务优先级是否更大，来切换任务*/
    if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
    {
        xSwitchRequired = pdTRUE;
    }
}
#endif /* configUSE_PREEMPTION */
```

2、时基更新函数多了返回值（用于判断是否要进行任务切换）具体实现为：多了一个预编译指令，解锁阻塞到时的任务并判断其优先级和当前执行任务的优先级高低，进行若解锁的任务优先级更高任务则进行任务切换

```c
if( xTaskIncrementTick() != pdFALSE )
{
  /* 任务切换，即触发PendSV */
        //portNVIC_INT_CTRL_REG = portNVIC_PENDSVSET_BIT;
        taskYIELD();
}
```

任务切换的主要方法

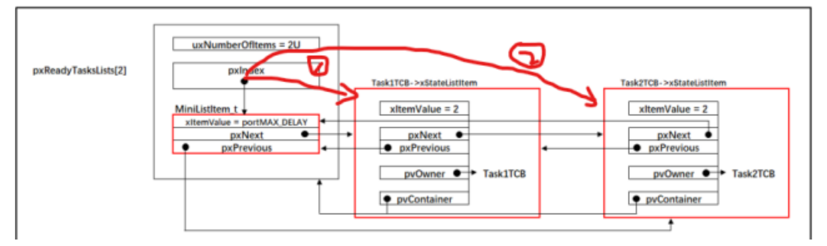
根据前述，任务切换是通过挂起一个pendSV请求
（1）寻进行旧栈保存
（2）找最高优先级
（3）将任务栈更新为所找到的最高优先级就绪列表

前面我们对最高优先级的寻找做出了解释，根据优先级位图的原理，我们还采取了优化的方法啊，但是我们还有一个内部细节没有讲述

```c
#define taskSELECT_HIGHEST_PRIORITY_TASK()                    \
{                                                             \
UBaseType_t uxTopPriority;                                    \
                                                              \
  /* 寻找包含就绪任务的最高优先级的队列 */                      \
  portGET_HIGHEST_PRIORITY( uxTopPriority, uxTopReadyPriority );  \
  /* 获取优先级最高的就绪任务的TCB，然后更新到pxCurrentTCB */    \
  listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists[ uxTopPriority ] ) );  \
}
```

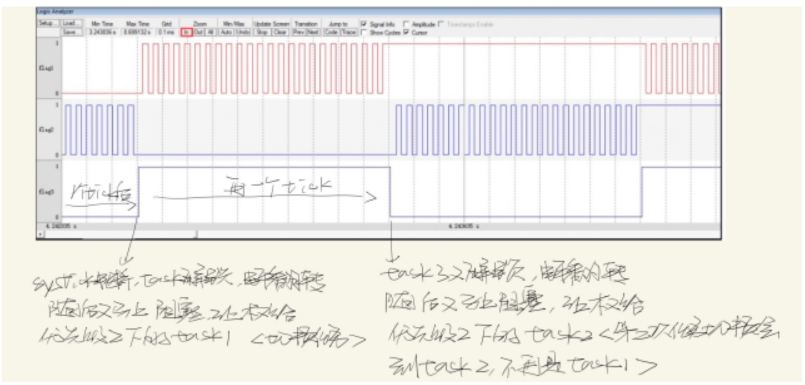
第一步，寻找最高优先级我们已经做出讲解
而第二步完成的操作为：每次进入这个函数的时候都将当前就绪列表某一优先级成员的pxIndex指针指向下一个节点（见左图），随后将pxIndex所指向的TCB返回

```c
/* 获取链表节点的OWNER，即TCB */
#define listGET_OWNER_OF_NEXT_ENTRY( pxTCB, pxList )          \
{ /*获取就绪列表的当前优先级指针*/                             \
  List_t * const pxConstList = ( pxList );                    \
  /* 让其索引指针指向下一个节点
    如果当前链表有N个节点，当第N次调用该函数时，pxInedex则指向第N个节点 */\
  ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext;  \
  /* 当前链表为空 */                                           \
  if( ( void * ) ( pxConstList )->pxIndex == ( void * ) &( ( pxConstList )->xListEnd ) )  \
  {                                                           \
    ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext;  \
  }                                                           \
  /* 获取索引指针所指向的节点的OWNER，即TCB */                   \
  ( pxTCB ) = ( pxConstList )->pxIndex->pvOwner;              \
}
```



每次指向下一节点（毗邻的同等优先级的任务），返回其TCB
Eg：第一次进入的时候会指向第一个节点如①所示，返回Task1TCB，下一次进入的指向如②所示，返回Task2TCB

因此，多次进行任务切换相当于在同一个优先级下不同任务轮流执行



仅对本例程而言：task优先级3、task1、task2优先级2

因为task3执行的是设置标志位并将task3设为阻塞状态（根据前面所述，设为阻塞状态会将任务从就绪列表移除，并移到延时列表当中，释放CPU使用权，同时进行任务切换）
因此，例程当中，task设置flag后就设为阻塞状态，随后切换任务，切换到了优先级为2的就绪列表下执行

一开始，第一次进入就绪列表优先级2的链表，执行task1，在task3解锁前，会一直执行，随后当延时时间到了（1个tick），task3解锁，切换到task3执行（因为task3优先级更高），在task3内部，我们翻转flag，再次将task3设为阻塞，随后就又切换到优先级2下执行，注意，此时已经是第二次进入优先级为2的列表，因此pxindex会如前述指向下一个任务，随后就会执行task2

由此往复，在sysTick时钟下，不断解锁task3后翻转flag，再置为阻塞，切换到优先级2下执行（每次切换都会执行下一任务）也就实现了**同一优先级的轮流执行**

小结

时间片得以实现后，我们就可以将想要“并行”的任务设置成同一优先级，在任务调度器的作用下，同一优先级的任务就可以进行轮流执行相当于“并行”