FreeRTOS（22）---FreeRTOS 系統節拍時鐘分析

其他 • 發表 2019-01-02

FreeRTOS 系統節拍時鐘分析

FreeRTOS 系統節拍時鐘分析

作業系統的執行是由系統節拍時鐘驅動的。

在 FreeRTOS 中，我們知道系統延時和阻塞時間都是以系統節拍時鐘週期為單位。在配置檔案 FreeRTOSConfig.h，改變巨集 configTICK_RATE_HZ 的值，可以改變系統節拍時鐘的中斷頻率，也間接的改變了系統節拍時鐘週期（T=1/f）。比如設定巨集 configTICK_RATE_HZ 為 100，則系統節拍時鐘週期為 10ms，設定巨集 configTICK_RATE_HZ 為 1000，則系統節拍時鐘週期為 1ms。

系統節拍中斷服務程式會呼叫函式 xTaskIncrementTick()來完成主要工作，如果該函式返回值為真（不等於 pdFALSE），說明處於就緒態任務的優先順序比當前執行的任務優先順序高。這會觸發一次 PendSV 中斷，進行上下文切換。我們重點看一下函式 xTaskIncrementTick()做了哪些事情，以及什麼情況下返回真值。

排程器正常情況

排程器正常（沒有掛起），即變數 uxSchedulerSuspended 的值為 pdFALSE。變數 uxSchedulerSuspended 是定義在 tasks.c 檔案中的靜態變數，記錄排程器執行狀態。當呼叫 API 函式 vTaskSuspendAll()掛起排程器時，會將變數 uxSchedulerSuspended 增 1。所以變數 uxSchedulerSuspended 為真時，表示排程器被掛起。

排程器正常情況下，首先將變數 xTickCount 增 1。變數 xTickCount 也是在 tasks.c 檔案中定義的靜態變數，它在啟動排程器時被清零，在每次系統節拍時鐘發生中斷後加 1，用來記錄系統節拍時鐘中斷的次數。核心會將所有阻塞的任務跟這個變數比較，以判斷是否超時（超時意味著可以解除阻塞）。

變數 xTickCount 的資料型別跟具體硬體有關，32 位架構硬體一般是無符號 32 位變數、8 位或 16 位架構一般是無符號 16 位變數。即便是 32 位變數，xTickCount 累加到 0xFFFFFFFF 後也會溢位。因此，在程式中要判斷變數 xTickCount 是否溢位。如果溢位（xTickCount 為 0），則呼叫巨集 taskSWITCH_DELAYED_LISTS()交換延時列表指標和溢位延時列表指標。這個牽扯的有點廣，我們慢慢說明。

為瞭解決 xTickCount 溢位問題，FreeRTOS 使用了兩個延時列表：xDelayedTaskList1 和 xDelayedTaskList2。並使用延時列表指標 pxDelayedTaskList 和溢位延時列表指標 pxOverflowDelayedTaskList 分別指向上面的延時列表 1 和延時列表 2（在建立任務時將延時列表指標指向延時列表）。順便說一下，上面的兩個延時列表指標變數和兩個延時列表變數都是在 tasks.c 中定義的靜態區域性變數。

比如我們使用 API 延時函式 vTaskDelay(xTicksToDelay) 將任務延時 xTicksToDelay 個系統節拍週期，延時函式會以當前的系統節拍中斷次數 xTickCount 為參考，這個值加上引數規定的延時時間 xTicksToDelay，即 xTickCount+ xTicksToDelay，就是下次喚醒任務的時間。xTickCount+xTicksToDelay 會被記錄到任務 TCB 中，隨著任務一起掛接到延時列表。如果核心判斷出 xTickCount+ xTicksToDelay 溢位（大於 32 位可以表示的最大值），就將當前任務掛接到列表指標 pxOverflowDelayedTaskList 指向的列表中，否則就掛接到列表指標 pxDelayedTaskList 指向的列表中。任務按照延時時間，順序的插入到延時列表中。

所以當系統節拍中斷次數計數器 xTickCount 溢位時，必須將延時列表指標 pxDelayedTaskList 和溢位延時列表指標 pxOverflowDelayedTaskList 交換以便正確處理延時的任務。巨集 taskSWITCH_DELAYED_LISTS()的程式碼如下所示：

```
#definetaskSWITCH_DELAYED_LISTS()                                                \
{                                                                                 \
        List_t *pxTemp                                                            \
                                                                                   \
        /* The delayed tasks list should beempty when the lists are switched. */  \
        configASSERT( ( listLIST_IS_EMPTY( pxDelayedTaskList) ) );                  \
                                                                                  \
        pxTemp = pxDelayedTaskList;                                               \
        pxDelayedTaskList = pxOverflowDelayedTaskList;                             \
        pxOverflowDelayedTaskList = pxTemp;                                        \
        xNumOfOverflows++;                                                        \
        prvResetNextTaskUnblockTime                                                \
}
```

這段程式碼完成兩部分工作，第一是將延時列表指標 pxDelayedTaskList 和溢位延時列表指標 pxOverflowDelayedTaskList 交換；第二是呼叫函式 prvResetNextTaskUnblockTime()重新獲取下一次解除阻塞的時間，這個時間儲存在靜態變數 xNextTaskUnblockTime 中，該變數也是定義在 tasks.c 中。下面檢查延時列表任務是否到期時，會用到這個變數。

接下來函式會檢查延時列表，檢視延時的任務是否到期。前面我們說過，延時的任務根據延時時間先後，順序的插入到延時列表中，延時時間短的在前，延時時間長的在後，並且下一個要被喚醒任務的時間數值儲存在變數 xNextTaskUnblockTime 中。所以使用 xTickCount 與 xNextTaskUnblockTime 比較就可以知道是否有任務可以被喚醒。

```
if( xConstTickCount >=xNextTaskUnblockTime )
{
   /* 延時的任務到期，需要被喚醒 */
}
```

如果任務被喚醒，則將任務從延時列表中刪除，重新加入就緒列表。如果新加入就緒列表的任務優先順序大於當前任務優先順序，則會觸發一次上下文切換。

FreeRTOS 支援多個任務共享同一個優先順序，如果設定為搶佔式排程（巨集 configUSE_PREEMPTION 設定為 1）並且巨集 configUSE_TIME_SLICING 也為 1（或未定義），則相同優先順序的多個任務間進行任務切換。

最後還會呼叫時間片鉤子函式 vApplicationTickHook()。可以看到時間片鉤子函式實在中斷服務函式中呼叫的，所以這個鉤子函式必須簡潔、不可以呼叫不帶中斷保護的 API 函式。

排程器掛起情況

如果排程器掛起，正在執行的任務會一直繼續執行，核心不再排程（意味著當前任務不會被切換出去），直到該任務呼叫了 xTaskResumeAll()函式。

在排程器掛起階段內，FreeRTOS 使用靜態變數 **uxPendedTicks** 記錄掛起期間,系統節拍中斷的次數。當呼叫恢復排程器函式 xTaskResumeAll()時，會執行 uxPendedTicks 次本函式(xTaskIncrementTick())。變數 **uxPendedTicks** 同樣是在 tasks.c 中定義的。

自動任務切換

函式的最後幾行程式碼頗讓人難以理解，其中區域性變數 xSwitchRequired 是本函式的返回值，在文章開始也說過：「如果該函式返回值為真，說明處於就緒態任務的優先順序高於當前執行任務的優先順序，則會觸發一次 PendSV 中斷，進行上下文切換」，現在如果變數 xYieldPending 為真，則

返回值也會為真，函式結束後會進行上下文切換。這個變數 xYieldPending 的作用是什麼？又是在什麼時候被賦值為真呢？還真要從頭說起。

```
if( xYieldPending != pdFALSE )
{
    xSwitchRequired = pdTRUE;
}
```

帶中斷保護的 API 函式，都會有一個引數 pxHigherPriorityTaskWoken。如果 API 函式導致一個任務解鎖，並且解鎖的任務優先順序高於當前執行的任務，則 API 函式將 *pxHigherPriorityTaskWoken 設定成 pdTRUE。在中斷退出前，老版本的 FreeRTOS 需要手動觸發一次任務切換。

```
   BaseType_txHigherPriorityTaskWoken = pdFALSE;
   /*收到一幀資料，向命令列直譯器任務傳送通知*/
   vTaskNotifyGiveFromISR(xCmdAnalyzeHandle,&xHigherPriorityTaskWoken);

   /*是否需要強制上下文切換*/
   portYIELD_FROM_ISR(xHigherPriorityTaskWoken );
```

從 FreeRTOSV7.3.0 起，pxHigherPriorityTaskWoken 成為一個可選引數，並可以設定為 NULL。如果將引數 xHigherPriorityTaskWoken 設定為 NULL，並且帶中斷保護的 API 函式導致更高優先順序任務解鎖，任務什麼時候、怎麼切換呢？

原來從 FreeRTOSV7.3.0 起，核心增加了一個靜態變數 xYieldPending，這個變數也是在 tasks.c 中定義的。如果將變數 xYieldPending 設定為 pdTRUE，則會在下一次系統節拍中斷服務函式中，觸發一次任務切換，見本小節第一段程式碼描述。

讓我們看一下這個過程是如何實現的。

對於佇列以及使用佇列機制的訊號量、互斥量等，在中斷服務程式中呼叫了這些 API 函式，將任務從阻塞中解除，則需要呼叫函式 xTaskRemoveFromEventList()將任務的事件列表項從事件列表中移除。在移除事件列表項的過程中，會判斷解除的任務優先順序是否大於當前任務的優先順序，如果解除的任務優先順序更高，會將變數 xYieldPending 設定為 pdTRUE。在下一次系統節拍中斷服務函式中，觸發一次任務切換。程式碼如下所示：

```
  if(pxUnblockedTCB->uxPriority > pxCurrentTCB->uxPriority)
 {
      /*任務具有更高的優先順序，返回 pdTRUE。告訴呼叫這個函式的任務，它需要強制切換上下文。*/
      xReturn= pdTRUE;
```

```
        /*帶中斷保護的 API 函式的都會有一個引數引數"xHigherPriorityTaskWoken"，如果使用者沒有使用這個引數，這裡設定任務切換標誌。在下個系統中斷服務例程中，會檢查 xYieldPending 的值，如果為 pdTRUE 則會觸發一次上下文切換。*/
        xYieldPending= pdTRUE;
    }
```

對於 FreeRTOSV8.2.0 新推出的任務通知，也提供了帶中斷保護版本的 API 函式。按照邏輯推斷，這些 API 函式的引數 xHigherPriorityTaskWoken 也可以不使用，變數 xYieldPending 也應該作用於這些 API 函式。但事實是，在 FreeRTOSV9.0 之前的版本，FreeRTOS 都沒有實現這個功能，如果使用這些 API 函式解除了一個更高優先順序任務，必須手動的進行上下文切換。這可能是一個 BUG，因為在 FreeRTOS V9.0 版本中，已經修復了這個問題，可以使用變數 xYieldPending 自動切換上下文。這個 BUG 由 QQ 暱稱為「所長」的網友遇到。

在 V9.0 以及以上版本中，如果在中斷中釋放的通知引起更高優先順序的任務解鎖，API 函式會判斷引數 xHigherPriorityTaskWoken 是否有效，有效則將*xHigherPriorityTaskWoken 設定為 pdTRUE，此時需要手動切換上下文；否則，將變數 xYieldPending 設定為 pdTRUE，在下一次系統節拍中斷服務函式中，觸發一次任務切換。程式碼如下所示：

```
if( pxTCB->uxPriority >pxCurrentTCB->uxPriority )
{
    /*如果解除阻塞的任務優先順序大於當前任務優先順序,則設定上下文切換標識,
      等退出函式後手動切換上下文,或者在系統節拍中斷服務程式中自動切換上下文*/
    if(pxHigherPriorityTaskWoken != NULL )
    {
        *pxHigherPriorityTaskWoken= pdTRUE;    /* 設定手動切換標誌*/
    }
    else
    {
        xYieldPending= pdTRUE;                 /* 設定自動切換標誌*/
    }
}
```

函式 xTaskIncrementTick()完整程式碼如下所示，根據上面的講解以及程式碼的註釋，理解這些程式碼應該不是難事。

```
BaseType_t xTaskIncrementTick( void )
{
TCB_t * pxTCB;
TickType_t xItemValue;
```

```
BaseType_t xSwitchRequired = pdFALSE;

    /* 每當系統節拍定時器中斷髮生,移植層都會呼叫該函式.
       函式將系統節拍中斷計數器加 1,
       然後檢查新的系統節拍中斷計數器值是否解除某個任務.*/
    if(uxSchedulerSuspended == ( UBaseType_t ) pdFALSE )
    {   /* 排程器正常情況 */
        const TickType_txConstTickCount = xTickCount + 1;

        /* 系統節拍中斷計數器加 1,如果計數器溢位(為 0),
           交換延時列表指標和溢位延時列表指標 */
        xTickCount = xConstTickCount;
        if( xConstTickCount == ( TickType_t ) 0U )
        {
            taskSWITCH_DELAYED_LISTS();
        }

        /* 檢視是否有延時任務到期.任務按照喚醒時間的先後順序儲存在佇列中,
           這意味著只要佇列中的最先喚醒任務沒有到期,其它任務一定沒有到期.*/
        if( xConstTickCount >=xNextTaskUnblockTime )
        {
            for( ;; )
            {
                if( listLIST_IS_EMPTY( pxDelayedTaskList) != pdFALSE )
                {
                    /* 如果延時列表為空,設定 xNextTaskUnblockTime 為最大值 */
                   xNextTaskUnblockTime = portMAX_DELAY;
                    break;
                }
                else
                {
                    /* 如果延時列表不為空,獲取延時列表第一個列表項值,
                       這個列表項值儲存任務喚醒時間.
                       喚醒時間到期,延時列表中的第一個列表項所屬的任務要被移除阻塞狀態 */
                    pxTCB = ( TCB_t * )listGET_OWNER_OF_HEAD_ENTRY( pxDelayedTaskList );
                    xItemValue =listGET_LIST_ITEM_VALUE( &( pxTCB->xStateListItem ) );

                    if( xConstTickCount < xItemValue )
```

```
            {
                /* 任務還未到解除阻塞時間?將當前任務喚醒時間設定為下次解除阻塞時間. */
               xNextTaskUnblockTime = xItemValue;
                break;
            }

            /* 從阻塞列表中刪除到期任務 */
            ( void ) uxListRemove( &( pxTCB->xStateListItem ) );

            /* 是因為等待事件而阻塞?是的話將到期任務從事件列表中刪除 */
            if(listLIST_ITEM_CONTAINER( &( pxTCB->xEventListItem ) ) != NULL )
            {
                ( void ) uxListRemove( &( pxTCB->xEventListItem ) );
            }

            /* 將解除阻塞的任務放入就緒列表 */
            prvAddTaskToReadyList( pxTCB );

            #if (  configUSE_PREEMPTION == 1 )
            {
                /* 使能了搶佔式核心.
                   如果解除阻塞的任務優先順序大於當前任務,觸發一次上下文切換標誌 */
                if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
                {
                    xSwitchRequired= pdTRUE;
                }
            }
            #endif /*configUSE_PREEMPTION */
        }
    }
}

/* 如果有其它任務與當前任務共享一個優先順序,
   則這些任務共享處理器(時間片) */
#if ( (configUSE_PREEMPTION == 1 ) && ( configUSE_TIME_SLICING == 1 ) )
{
    if(listCURRENT_LIST_LENGTH( &pxReadyTasksLists[pxCurrentTCB->uxPriority] )
        > ( UBaseType_t ) 1 )
```

```
            {
                xSwitchRequired = pdTRUE;
            }
            else
            {
               mtCOVERAGE_TEST_MARKER();
            }
        }
        #endif /* ( (configUSE_PREEMPTION == 1 ) && ( configUSE_TIME_SLICING == 1 ) ) */

        #if (configUSE_TICK_HOOK == 1 )
        {
            /* 呼叫時間片鉤子函式*/
            if( uxPendedTicks == ( UBaseType_t ) 0U )
            {
                vApplicationTickHook();
            }
        }
        #endif /*configUSE_TICK_HOOK */
    }
    else
    {   /* 排程器掛起狀態,變數 uxPendedTicks 用於統計排程器掛起期間,系統節拍中斷次數.
           當呼叫恢復排程器函式時,會執行 uxPendedTicks 次本函式(xTaskIncrementTick()):
           恢復系統節拍中斷計數器,如果有任務阻塞到期,則刪除阻塞狀態 */
        ++uxPendedTicks;

        /* 呼叫時間片鉤子函式*/
        #if (configUSE_TICK_HOOK == 1 )
        {
            vApplicationTickHook();
        }
        #endif
    }

    #if (configUSE_PREEMPTION == 1 )
    {   /* 如果在中斷中呼叫的 API 函式喚醒了更高優先順序的任務,
           並且 API 函式的引數 pxHigherPriorityTaskWoken 為 NULL 時,
           變數 xYieldPending 用於上下文切換標誌 */
```

```
        if( xYieldPending!= pdFALSE )
        {
            xSwitchRequired = pdTRUE;
        }
    }
    #endif /*configUSE_PREEMPTION */

    return xSwitchRequired;
}
```

標籤： 任務 函式 變數 中斷 列表 延時 切換 系統