2. シグナル待ちの中を覗いてみる
〔freeRTOSを解析する〕
2020年9月13日

■osSignalWaitを追いかける

osSignalWait つぎの経路でportYIELD が呼び出され、portYIELD の中からPendSV が発行されている。
osSignalWait --> xTaskNotifyWait --> portYIELD_WITHIN_API (=portYIELD)

PendSV の割り込み優先度は15 に設定されているが、portYIELD は、taskENTER_CRITICALとtaskEXIT_CRITICALに挟まれている。
taskENTER_CRITICAL では、BASEPRI の上位4ビットが8 に設定され、優先度8～15 の例外は禁止されている。
従ってportYIELD 呼び出しの時点ではPendSV が実行されない。
portYIELD の直後にtaskEXIT_CRITICAL が呼び出されているので、PendSV はその直後に実行されると思われる。

```
BaseType_t xTaskNotifyWait( uint32_t ulBitsToClearOnEntry, uint32_t ulBitsToClearOnExit, uint32_t *pulNotificationValue,
        TickType_t xTicksToWait )
{
BaseType_t xReturn;

        taskENTER_CRITICAL();
        {
                /* Only block if a notification is not already pending. */
                if( pxCurrentTCB->ucNotifyState != taskNOTIFICATION_RECEIVED )
                {
                        /* Clear bits in the task's notification value as bits may get
                        set     by the notifying task or interrupt.  This can be used to
                        clear the value to zero. */
                        pxCurrentTCB->ulNotifiedValue &= ~ulBitsToClearOnEntry;

                        /* Mark this task as waiting for a notification. */
                        pxCurrentTCB->ucNotifyState = taskWAITING_NOTIFICATION;

                        if( xTicksToWait > ( TickType_t ) 0 )
                        {
                                prvAddCurrentTaskToDelayedList( xTicksToWait, pdTRUE );
                                traceTASK_NOTIFY_WAIT_BLOCK();

                                /* All ports are written to allow a yield in a critical
                                section (some will yield immediately, others wait until the
                                critical section exits) - but it is not something that
                                application code should ever do. */
                                portYIELD_WITHIN_API();
                        }
                        else
                        {
                                mtCOVERAGE_TEST_MARKER();
                        }
                }
                else
                {
                        mtCOVERAGE_TEST_MARKER();
                }
        }
        taskEXIT_CRITICAL();

        taskENTER_CRITICAL();
        {
                traceTASK_NOTIFY_WAIT();

                if( pulNotificationValue != NULL )
                {
```

```
                        /* Output the current notification value, which may or may not
                        have changed. */
                        *pulNotificationValue = pxCurrentTCB->ulNotifiedValue;
                }

                /* If ucNotifyValue is set then either the task never entered the
                blocked state (because a notification was already pending) or the
                task unblocked because of a notification.  Otherwise the task
                unblocked because of a timeout. */
                if( pxCurrentTCB->ucNotifyState != taskNOTIFICATION_RECEIVED )
                {
                        /* A notification was not received. */
                        xReturn = pdFALSE;
                }
                else
                {
                        /* A notification was already pending or a notification was
                        received while the task was waiting. */
                        pxCurrentTCB->ulNotifiedValue &= ~ulBitsToClearOnExit;
                        xReturn = pdTRUE;
                }

                pxCurrentTCB->ucNotifyState = taskNOT_WAITING_NOTIFICATION;
        }
        taskEXIT_CRITICAL();

        return xReturn;
}
```

```
/* Scheduler utilities. */
#define portYIELD()                                                                     \
{                                                                                       \
        /* Set a PendSV to request a context switch. */                         \
        portNVIC_INT_CTRL_REG = portNVIC_PENDSVSET_BIT;                         \
                                                                                \
        /* Barriers are normally not required but do ensure the code is completely      \
        within the specified behaviour for the architecture. */                 \
        __asm volatile( "dsb" ::: "memory" );                                   \
        __asm volatile( "isb" );                                                \
}
```

```
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY   15
```

■taskENTER_CRITICAL
taskENTER_CRITICAL --> vPortEnterCritical
BASEPRI = 0x80

```
portFORCE_INLINE static void vPortRaiseBASEPRI( void )
{
uint32_t ulNewBASEPRI;

        __asm volatile
        (
                "       mov %0, %1                                              \n" \
                "       msr basepri, %0                                         \n" \
                "       isb                                                     \n" \
                "       dsb                                                     \n" \
                :"=r" (ulNewBASEPRI) : "i" ( configMAX_SYSCALL_INTERRUPT_PRIORITY ) : "memory" );
}
```