TI DSP, MCU, Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee (이상훈)
gcccompil3r@gmail.com
학생 – 김형주
mihaelkel@naver.com

FreeRTOS 멀티-태스킹

```
#include "HL_sys_common.h"
#include "FreeRTOS.h"
#include "os_task.h"
#include "HL_het.h"
#include "HL_gio.h"
xTaskHandle xTask1Handle;
xTaskHandle xTask2Handle;

void vTask1(void *pvParameters)
{
    for(;;)
    {
        gioSetBit(hetPORT1, 17, gioGetBit(hetPORT1, 17) ^ 1);
        vTaskDelay(100);
    }
}
void vTask2(void *pvParameters)
{
    for(;;)
    {
        gioSetBit(gioPORTB, 4, gioGetBit(gioPORTB, 4) ^ 1);
        vTaskDelay(100);
    }
}
void main(void)
{
    gioSetDirection(hetPORT1, 0xFFFFFFFF);
    gioInit();
    if(xTaskCreate(vTask1, "Task1", configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) != pdTRUE)
    {
        while(1);
    }
    if(xTaskCreate(vTask2, "Task2", configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) != pdTRUE)
    {
        while(1);
    }
    vTaskStartScheduler();
    while(1);
}
```

xTaskCreate 분석

```
#define xTaskCreate                     MPU_xTaskCreate
```

```
1  BaseType_t MPU_xTaskCreate( TaskFunction_t pxTaskCode, const char * const pcName, const uint16_t usStackDepth,
2  void * const pvParameters, UBaseType_t uxPriority, TaskHandle_t * const pxCreatedTask )
3  {
4      BaseType_t xReturn;
5      BaseType_t xRunningPrivileged = prvRaisePrivilege();
6      xReturn = xTaskCreate( pxTaskCode, pcName, usStackDepth, pvParameters, uxPriority, pxCreatedTask );
7      portRESET_PRIVILEGE( xRunningPrivileged );
8      return xReturn;
9  }
```

Colored by Color Scripter

4 : 리턴할 변수 xReturn을 선언한다

5 : 현재 process의 privileged state를 xRunningPrivileged 변수에 저장한다.

6 : pxTaskCode에 저장되어 있는 함수포인터를 task로 만든다.

7 : xRunningPrivileged에 저장해두었던 privileged state를 다시 불러온다.

8 : xReturn 값을 리턴한다.

xTaskCreate 분석

```
1  #if( configSUPPORT_DYNAMIC_ALLOCATION == 1 )
2      BaseType_t xTaskCreate(    TaskFunction_t pxTaskCode,
3                             const char * const pcName,
4                             const uint16_t usStackDepth,
5                             void * const pvParameters,
6                             UBaseType_t uxPriority,
7                             TaskHandle_t * const pxCreatedTask ) /*lint !e971 Unqualified char types are allowed
   for strings and single characters only. */
8      {
9      TCB_t *pxNewTCB;
10     BaseType_t xReturn;
11         /* If the stack grows down then allocate the stack then the TCB so the stack
12         does not grow into the TCB.  Likewise if the stack grows up then allocate
13         the TCB then the stack. */
14         #if( portSTACK_GROWTH > 0 )
15         {
16             /* Allocate space for the TCB.  Where the memory comes from depends on
17             the implementation of the port malloc function and whether or not static
18             allocation is being used. */
19             pxNewTCB = ( TCB_t * ) pvPortMalloc( sizeof( TCB_t ) );
20             if( pxNewTCB != NULL )
21             {
22                 /* Allocate space for the stack used by the task being created.
23                 The base of the stack memory stored in the TCB so the task can
24                 be deleted later if required. */
25                 pxNewTCB->pxStack = ( StackType_t * ) pvPortMalloc( ( ( ( size_t ) usStackDepth ) * sizeof(
   StackType_t ) ) ); /*lint !e961 MISRA exception as the casts are only redundant for some ports. */
26                 if( pxNewTCB->pxStack == NULL )
27                 {
28                     /* Could not allocate the stack.  Delete the allocated TCB. */
29                     vPortFree( pxNewTCB );
30                     pxNewTCB = NULL;
31                 }
32             }
33         }
34         #else /* portSTACK_GROWTH */
35
```

```
        {
        StackType_t *pxStack;
            /* Allocate space for the stack used by the task being created. */
            pxStack = ( StackType_t * ) pvPortMalloc( ( ( ( size_t ) usStackDepth ) * sizeof( StackType_t ) ) );
/*lint !e961 MISRA exception as the casts are only redundant for some ports. */
            if( pxStack != NULL )
            {
                /* Allocate space for the TCB. */
                pxNewTCB = ( TCB_t * ) pvPortMalloc( sizeof( TCB_t ) ); /*lint !e961 MISRA exception as the casts
are only redundant for some paths. */
                if( pxNewTCB != NULL )
                {
                    /* Store the stack location in the TCB. */
                    pxNewTCB->pxStack = pxStack;
                }
                else
                {
                    /* The stack cannot be used as the TCB was not created.  Free
                    it again. */
                    vPortFree( pxStack );
                }
            }
            else
            {
                pxNewTCB = NULL;
            }
        }
        #endif /* portSTACK_GROWTH */
        if( pxNewTCB != NULL )
        {
            #if( tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE != 0 )
            {
                /* Tasks can be created statically or dynamically, so note this
                task was created dynamically in case it is later deleted. */
                pxNewTCB->ucStaticallyAllocated = tskDYNAMICALLY_ALLOCATED_STACK_AND_TCB;
            }
            #endif /* configSUPPORT_STATIC_ALLOCATION */
            prvInitialiseNewTask(  pxTaskCode,  pcName,  ( uint32_t ) usStackDepth,  pvParameters,  uxPriority,
pxCreatedTask, pxNewTCB, NULL );
            prvAddNewTaskToReadyList( pxNewTCB );
            xReturn = pdPASS;
        }
        else
        {
            xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
        }
        return xReturn;
    }
#endif /* configSUPPORT_DYNAMIC_ALLOCATION */
```

```
BaseType_t xReturn;
TCB_t *pxNewTCB;
StackType_t *pxStack;
```

위의 세 변수를 선언하고,

```
pxStack = ( StackType_t * ) pvPortMalloc( ( ( ( size_t ) usStackDepth ) * sizeof( StackType_t ) ) ); /*lint !e961 MISRA exception as the casts are only redundant for some ports. */
```

parameter로 입력했던 stack 크기만큼 할당해준다.

```
if( pxStack != NULL )
{
/* Allocate space for the TCB. */
pxNewTCB = ( TCB_t * ) pvPortMalloc( sizeof( TCB_t ) ); /*lint !e961 MISRA exception as the casts are only redundant for some paths. */
```

stack memory 할당이 성공하면, pxNewTCB 라는 변수명으로 TCB(Task Control Block) 메모리를 할당한다.
TCB에는 후에 태스크의 정보들이 저장된다.(태스크 id, 태스크 상태, CPU 스케줄링 정보 등)

```
#if( tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE != 0 )
{
/* Tasks can be created statically or dynamically, so note this
task was created dynamically in case it is later deleted. */
pxNewTCB->ucStaticallyAllocated = tskDYNAMICALLY_ALLOCATED_STACK_AND_TCB;
}
#endif /* configSUPPORT_STATIC_ALLOCATION */
```

stack이 동적으로 할당되었는지, 정적으로 할당되었는지를 pxNewTCB->ucStaticallyAllocated 변수에 저장한다.
이는 후에 free를 하기 위한 목적으로 예상된다.

```
prvInitialiseNewTask( pxTaskCode, pcName, ( uint32_t ) usStackDepth, pvParameters, uxPriority, pxCreatedTask, pxNewTCB, NULL );
prvAddNewTaskToReadyList( pxNewTCB );
xReturn = pdPASS;
```

task를 만들고, ReadyList에 올리고, pdPASS를 리턴한다.