Práctica 2. Internal Tasks con uC-OS-III

OBJETIVOS

En esta práctica se pide trabajar con las tareas internas del sistema operativo. En particular, se trabajará con las siguientes Tasks:

- The Tick Task -> *OS_TickTask()*

- The Idle Task -> *OS_IdleTask()*

TAREAS PREVIAS

- Explique las características principales de cada una de las tasks, y su función como parte de la planificación del RTOS.

- The Tick Task -> *OS_TickTask()*

  La función OS_Tick_Task es el hilo característico del Kernel del RTOS que sirve para llevar las temporizaciones de las tareas que se encuentran aguardando a que un determinado numero de ticks del sistema corran, para después poder volver a ella si el planificador lo permite.

- The Idle Task -> *OS_IdleTask()*
  La función IDLE se emplea por el Kernel cuando no hay ninguna tarea en ejecución o lista para pasar a ejecución.

- Explique la razón por la cual estas Tasks internas son de tipo Obligatorio

  La tarea Tick Task no se puede deshabilitar ya que es la encargada de mantener la temporización de el resto de tareas, por lo que ninguna otra tarea tendría sentido sin que esta existiera.
  La tarea Idle Task no se puede deshabilitar ya que en algún momento el RTOS estará inactivo, y pasaría a esta tarea, por lo que necesitamos un lugar donde ir en los momentos de no actividad, también es útil a la hora de computar estadísticas.

- ¿Qué prioridad presenta *OS_TickTask()*? ¿Es configurable? Razone la respuesta

```
#define  OS_CFG_TICK_RATE_HZ            1000u
#define  OS_CFG_TICK_TASK_PRIO           8-4u
#define  OS_CFG_TICK_TASK_STK_SIZE      2048u
#define  OS_CFG_TICK_WHEEL_SIZE          17u
```

  Originalmente se encontraba a 4, como es la más prioritaria se ha modificado a 2

```
#define  OS_CFG_TICK_RATE_HZ            1000u
#define  OS_CFG_TICK_TASK_PRIO           2u
#define  OS_CFG_TICK_TASK_STK_SIZE      2048u
#define  OS_CFG_TICK_WHEEL_SIZE          17u
```

-

- ¿Qué prioridad presenta *OS_IdleTask()*? ¿Es configurable? Razone la respuesta

Por defecto se configurar como la menos prioritaria (PRIO_MAX-1)

```
#define OS_CFG_PRIO_MAX                  8u
```

Como la prioridad máxima es 7 (0-7) esta idle task tendrá la prioridad numero 7.

- Considere que el *RTOS* se ha iniciado con 16 Tasks de aplicación, y nos encontramos en la marca de tiempos *OSTickCtr* = 24. Se ejecuta Task1, la cual presenta una llamada a *OSTimeDly(5)*, exactamente para la marca de tiempos indicada anteriormente. Responda a las siguientes preguntas:

    - ¿Qué valor es recomendable asignar a OS_CFG_TICK_WHEEL_SIZE?
      Task_App/4 =16/4=4
      Se necesitaría 4

    - ¿A qué posición de tabla se asignará el TCB de Task1?
      Entra en la tabla según la temporización de su tiempo de ejecución.
    - Represente la estructura de la tabla *TickWheel[]* junto con los parámetros del TCB involucrados (apuntamientos, valores de Tick Match/Remain..) teniendo en cuenta los datos anteriores
      Tick Spoke Y posición

Tick Spoke:	Posición Memoria:
0x00001	TABLA[0] 0x00002C
0x00002	TABLA[1] 0x00002B
0x00003	TABLA[2] 0x00002D
0x00004	TABLA[3] 0x00002E
0x00005	TABLA[4] 0x00002F
0x00006	TABLA[5] 0x00003A
0x00007	TABLA[6] 0x00003B
0x00008	TABLA[7] 0x00003C
0x00009	TABLA[8] 0x00003D
0x00010	TABLA[9] 0x00003E
0x00011	TABLA[10] 0x00003F
0x00012	TABLA[11] 0x00004A
0x00013	TABLA[12] 0x00004B
0x00014	TABLA[13] 0x00004C
0x00015	TABLA[14] 0x00004D
0x00016	TABLA[15] 0x00004E

APLICACIÓN

El usuario deberá crear 4 Tasks de aplicación, las cuales deberán disponer de prioridad número 3. Además, se deberá tener en cuenta los siguientes requisitos:

- Periodicidad_task1: 1000ms

```
void TASK_LED1 (void)
{
    OS_ERR os_err;

    while(1)
    {
        //sprintf(txdata,"TASK1\r\n");
        //EnviarString(txdata, Uart3);
        LED_GREEN = !LED_GREEN_Read;
        OSTimeDly(1000,OS_OPT_TIME_DLY, &os_err);
```

- Periodicidad_task2: 2000ms

```
void TASK_LED2 (void)
{
    OS_ERR os_err;

    while(1)
    {
        //sprintf(txdata,"TASK1\r\n");
        //EnviarString(txdata, Uart3);
        LED_GREEN = !LED_GREEN_Read;
        OSTimeDly(2000,OS_OPT_TIME_DLY, &os_err);  //2000 ticks del RTOS -> 2000ms

    }
```

- Periodicidad_task3: 4000ms

```
void TASK_LED3 (void)
{
    OS_ERR os_err;

    while(1)
    {
        //sprintf(txdata,"TASK1\r\n");
        //EnviarString(txdata, Uart3);
        LED_GREEN = !LED_GREEN_Read;
        OSTimeDly(4000,OS_OPT_TIME_DLY, &os_err);  //4000 ticks del RTOS -> 100ms

    }
```

- Periodicidad_task4: 8000ms

```
void TASK_LED4 (void)
{
    OS_ERR os_err;

    while(1)
    {
        //sprintf(txdata,"TASK1\r\n");
        //EnviarString(txdata, Uart3);
        LED_GREEN = !LED_GREEN_Read;
        OSTimeDly(8000,OS_OPT_TIME_DLY, &os_err);
```

- Las Tasks (1,2,3,4) NO deben imprimir ningún tipo de mensaje
- Las Tasks (1,2,3,4) deberán hacer parpadear un LED, como parte del hilo

El usuario deberá crear una Task extra, llamada *Task_monitor*, encargada de monitorear las variables TCB de las 4 Tasks anteriores, asociadas con la task interna *OS_IdleTask()*

- *.TickSpokePtr posición a la que va ligada de la tabla cada entrada*
- *.TickNextPtr puntero a siguiente*
- *.TickPrevPtr Puntero al anterior*
- *.TickRemain Cantidad de ticks para terminar la tarea*
- *.TickMatch Punto en el cual la tarea tiene que salir de ejecución.*

Como parámetros de ejecución de *Task_monitor*:

- Periodicidad *Task_monitor*: 50ms
- Prioridad *Task_monitor*: 2
- Se deberá imprimir un mensaje que refleje el estado de las variables anteriores. Se recomienda no incluir \n, para poder ver la actualización en la misma línea.

```
void Task_monitor(void){
    OS_ERR os_err;
    while(1){
       sprintf(txdata,"Data Task1-----------------------------------------------");
        sprintf(txdata,"TickSpokePTR : %u ", &task_led1_TCB.TickSpokePtr );
        sprintf(txdata,"TickNextPTR : %u ", &task_led1_TCB.TickNextPtr);
        sprintf(txdata,"TickPrevPTR : %u ", &task_led1_TCB.TickPrevPtr);
        sprintf(txdata,"TickRemain : %u ", &task_led1_TCB.TickRemain);
         sprintf(txdata,"TickmATCH : %u ", &task_led1_TCB.TickCtrMatch);


         sprintf(txdata,"Data Task2-----------------------------------------------");
        sprintf(txdata,"TickSpokePTR : %u ", &task_led2_TCB.TickSpokePtr );
        sprintf(txdata,"TickNextPTR : %u ", &task_led2_TCB.TickNextPtr);
        sprintf(txdata,"TickPrevPTR : %u ", &task_led2_TCB.TickPrevPtr);
        sprintf(txdata,"TickRemain : %u ", &task_led2_TCB.TickRemain);
         sprintf(txdata,"TickmATCH : %u ", &task_led2_TCB.TickCtrMatch);


         sprintf(txdata,"Data Task3-----------------------------------------------");
        sprintf(txdata,"TickSpokePTR : %u ", &task_led3_TCB.TickSpokePtr );
        sprintf(txdata,"TickNextPTR : %u ", &task_led3_TCB.TickNextPtr);
        sprintf(txdata,"TickPrevPTR : %u ", &task_led3 TCB.TickPrevPtr);

         sprintf(txdata,"Data Task4-----------------------------------------------");
         sprintf(txdata,"TickSpokePTR : %u ", &task_led4_TCB.TickSpokePtr );
         sprintf(txdata,"TickNextPTR : %u ", &task_led4_TCB.TickNextPtr);
         sprintf(txdata,"TickPrevPTR : %u ", &task_led4_TCB.TickPrevPtr);
         sprintf(txdata,"TickRemain : %u ", &task_led4_TCB.TickRemain);
          sprintf(txdata,"TickmATCH : %u ", &task_led4_TCB.TickCtrMatch);
```

Para la periodicidad se añade la instrucción OSTimeDly(50,OS_OPT_TIME_DLY, &os_err);