PRÁCTICA 4. The *Ready List*

OBJETIVOS

En esta práctica se trabajará con la Ready List, en particular estudiar como el RTOS actualiza la tabla OSRdyList[] para gestionar las Tasks en proceso *Ready -> Running*

TAREAS PREVIAS

- ¿Qué es la *Ready List*?

  LA Ready list es una estructura de datos propia del RTOS, en la que se van encolando las tareas que han terminado su tiempo de espera y ya están listas para ser ejecutadas, como su propio nombre indica

- ¿Cuál es el nombre (data type) que define el tipo de estructura de OSRdyList[]?
  Si vamos haciendo un traceback de las definiciones e instanciaciones de esta estructura de datos encontrasmos estas sentencias:

```
typedef  struct  os_rdy_list          OS_RDY_LIST;

struct  os_rdy_list {
    OS_TCB                *HeadPtr;
    OS_TCB                *TailPtr;
    OS_OBJ_QTY             NbrEntries;
};
```

  Esta última siendo la definición inicial de esta estructura, la cual contiene un puntero al head tcb detro suya, el análogo al tail pointer y un número que indica la cantidad de entradas que esta posee.

- Explique los 3 diferentes elementos que componen la estructura anterior (.TailPtr, .HeadPtr, .Entries)
  Como bien se ha indicado antes, el primero argumento sirve para indicar qué tarea, a través de su tcb, se encuentra a la cola o en la última posición de etsa estructura de ddatos, viendola siempre como una lista ordenada. Lo mismo sucedería con el head pointer, pero esta vzz apuntando a la primera posición, y el parámetro entries simplemente sería un acumulador que indica el número total de entradas que esta estructura de datos posee, contabilizando así las tareas correspondientes.

- Si la OSRdyList presenta 16 vectores [0],…,[15]. ¿Qué se deduce de nuestra aplicación?
  Que nuestro sistema tiene 16 tareas incluyendo el propio rtos y la idle task

- Respecto a la pregunta anterior, ¿Con quién está asociado el vector 15 [15]?
  La Idle task por ser siempre la de menor prioridad

- Si todos los elementos .HeadPtr y .TailPtr se encuentran apuntando a NULL. ¿En qué estado se encuentra el sistema?
  Se acaba de iniciar el rtOS, por lo que ninguna tarea ha podido entrar al planificador aún

- Explique que elementos adicionales se necesitan emplear para especificar el orden de

ejecución de cada Task, cuando existen más de 2 Tasks para un mismo nivel de prioridad en estado *Ready*
Se tiene que tener en cuenta el planificador, ya que si se está en una modalidad de round robin por ejemplo, las tareas entrarán y saldrán de ejecución rápidamente en función del time cuanta y lógicamente los recursos que cada tarea necesite, y si otra tarea que esté en ready requiere esos mismos recursos, el planificador deberá seleccionar cual de las dos tareas pasa a ejecución.

- Si mi sistema presenta un número máximo de prioridades OS_CFG_PRIO_MAX. ¿De cuántos vectores se compondrá la OSRdyList[]?
  DE OS_CFG_PRIO_MAX -1 , YA QUE ESE 1 REPRESENTA AL PROPIO RTOS, y si quisiaramos meter en ese grupo también la idle task , se le restaría un a prioridad más, ya qe ambas siempre van a querer estra ejecutandose.

- Justifique si las siguientes expresiones pueden ser ciertas o falsas:
  - OSRdyList[OS_CFG_PRIO_MAX - 1].Entries = 2
  - OSRdyList[OS_CFG_PRIO_MAX - 1].HeadPtr = &OS_TCB_Task1
  - OSRdyList[OS_CFG_PRIO_MAX - 1].TailPtr = &OS_TCB_IDLE_Task
  - OSRdyList[OS_CFG_PRIO_MAX - 3].Entries = 3
  - OSRdyList[OS_CFG_PRIO_MAX - 2].HeadPtr = &OS_TCB_Task2
  - OSRdyList[OS_CFG_PRIO_MAX - 2].TailPtr = &OS_TCB_IDLE_Task

APLICACIÓN

Se deberán crear 6 Tasks de aplicación y 1 Task de monitorización. La configuración de cada Hilo de aplicación es la siguiente:

TASK	PRIO.	PERIODICIDAD	FUNCIÓN
TASK_1	2	1 seg	LED + delay_ms(1000)
TASK_2	2	500ms	LED
TASK_3	3	1 seg	LED + delay_ms(500)
TASK_4	4	2 seg	LED + delay_ms(2000)
TASK_5	5	5 seg	LED + delay_ms(500)
TASK_6	6	8 seg	LED + delay_ms(1000)

```
OSTaskCreate(
            (OS_TCB *) & task_led1_TCB,
            (CPU_CHAR *) "Tarea 1. Control led y mensaje uart",
            (OS_TASK_PTR) TASK_LED1,
            (void *) 0,
            (OS_PRIO) 3,
            (CPU_STK *) & task_led1_STK[0],
            (CPU_STK_SIZE) 0u,
            (CPU_STK_SIZE) 1024u,
            (OS_MSG_QTY) 0u,
            (OS_TICK) 0u, //10*System Tick period
            (void *) 0,
            (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
            (OS_ERR *) & os_err);
OSTaskCreate(
            (OS_TCB *) & task_led2_TCB,
            (CPU_CHAR *) "Tarea 2. Control led y mensaje uart",
            (OS_TASK_PTR) TASK_LED2,
            (void *) 0,
            (OS_PRIO) 3,
            (CPU_STK *) & task_led2_STK[0],
            (CPU_STK_SIZE) 0u,
            (CPU_STK_SIZE) 1024u,
            (OS_MSG_QTY) 0u,
            (OS_TICK) 0u, //10*System Tick period
            (void *) 0,
            (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
            (OS_ERR *) & os_err);

OSTaskCreate(
            (OS_TCB *) & task_led3_TCB,
            (CPU_CHAR *) "Tarea 3. Control led y mensaje uart",
            (OS_TASK_PTR) TASK_LED3,
            (void *) 0,
            (OS_PRIO) 3,
```

```
OSTaskCreate(
        (OS_TCB *) & task_led3_TCB,
        (CPU_CHAR *) "Tarea 3. Control led y mensaje uart",
        (OS_TASK_PTR) TASK_LED3,
        (void *) 0,
        (OS_PRIO) 3,
        (CPU_STK *) & task_led3_STK[0],
        (CPU_STK_SIZE) 0u,
        (CPU_STK_SIZE) 1024u,
        (OS_MSG_QTY) 0u,
        (OS_TICK) 0u, //10*System Tick period
        (void *) 0,
        (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
        (OS_ERR *) & os_err);
```

```
OSTaskCreate(
        (OS_TCB *) & task_led4_TCB,
        (CPU_CHAR *) "Tarea 4. Control led y mensaje uart",
        (OS_TASK_PTR) TASK_LED4,
        (void *) 0,
        (OS_PRIO) 3,
        (CPU_STK *) & task_led4_STK[0],
        (CPU_STK_SIZE) 0u,
        (CPU_STK_SIZE) 1024u,
        (OS_MSG_QTY) 0u,
        (OS_TICK) 0u, //10*System Tick period
        (void *) 0,
        (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
        (OS_ERR *) & os_err);
```

```
OSTaskCreate(
        (OS_TCB *) & task_led5_TCB,
        (CPU_CHAR *) "Tarea 5. Control led y mensaje uart",
        (OS_TASK_PTR) TASK_LED5,
        (void *) 0,
        (OS_PRIO) 3,
        (CPU_STK *) & task_led5_STK[0],
        (CPU_STK_SIZE) 0u,
        (CPU_STK_SIZE) 1024u,
        (OS_MSG_QTY) 0u,
        (OS_TICK) 0u, //10*System Tick period
        (void *) 0,
        (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
        (OS_ERR *) & os_err);
```

```
OSTaskCreate(
        (OS_TCB *) & task_led6_TCB,
        (CPU_CHAR *) "Tarea 6. Control led y mensaje uart",
        (OS_TASK_PTR) TASK_LED6,
        (void *) 0,
        (OS_PRIO) 3,
        (CPU_STK *) & task_led6_STK[0],
        (CPU_STK_SIZE) 0u,
        (CPU_STK_SIZE) 1024u,
        (OS_MSG_QTY) 0u,
        (OS_TICK) 0u, //10*System Tick period
        (void *) 0,
        (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
        (OS_ERR *) & os_err);
```

```
OSTaskCreate(
        (OS_TCB *) & task_monitor_TCB,
        (CPU_CHAR *) "Tarea Monitor. Control led y mensaje uart",
        (OS_TASK_PTR) TASK_MONITOR,
        (void *) 0,
        (OS_PRIO) 1,
        (CPU_STK *) & task_monitor_STK[0],
        (CPU_STK_SIZE) 0u,
        (CPU_STK_SIZE) 1024u,
        (OS_MSG_QTY) 0u,
        (OS_TICK) 10u,
        (void *) 0,
        (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
        (OS_ERR *) & os_err);
```

La Task de monitorización debe disponer de:

- Nivel de prioridad: 1
- Periodicidad mínima/máxima: 10ms/50ms

El usuario deberá mostrar las Tasks en estado Ready por medio de la lectura de cada vector que compone la tabla OSRdyList[], así como el conjunto de elementos que definen su estructura (.Entries, .HeadPtr, .TailPtr)

Puesto que los TCBs son estructuras almacenadas en la Stack de cada Hilo, y OSRdyList[] almacena la dirección de estos TCBs, es necesario almacenar la dirección asociada a cada hilo y reemplazarla por el nombre del propio hilo o emplear el símbolo & a la hora de referenciar el TCB

Se da libertad para definir la estructura/tabla que se desee para imprimir los resultados.

```
if (OSRdyList[1].TailPtr == &task_monitor_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[1] apunta a task monitor");
}

if (OSRdyList[2].HeadPtr == &task_led1_TCB && OSRdyList[2].TailPtr
        == &task_led2_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[2] apunta a task 2");
} else {

    sprintf(txdata, "HeaderPointer de OSRdy[3] apunta a null");
}

if (OSRdyList[2].HeadPtr == &task_led2_TCB && OSRdyList[2].TailPtr == &task_led1_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[1] apunta a task monitor");
} else {

    sprintf(txdata, "HeaderPointer de OSRdy[3] apunta a null");
}

if (OSRdyList[2].TailPtr == &task_led2_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[1] apunta a task monitor");
}
```

```
if (OSRdyList[1].HeadPtr == &task_monitor_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[1] apunta a task monitor");
}

if (OSRdyList[2].HeadPtr == &task_led1_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[2] apunta a task 1");
}
if (OSRdyList[2].HeadPtr == &task_led2_TCB) {
    if (OSRdyList[2].TailPtr == &task_led2_TCB) {
        sprintf(txdata, "HeaderPointer de OSRdy[2] apunta a task 2");
    }
}
```

```
if (OSRdyList[3].HeadPtr == &task_led3_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[3] apunta a task 3");
} else {

    sprintf(txdata, "HeaderPointer de OSRdy[3] apunta a null");
```

```
if (OSRdyList[4].HeadPtr == &task_led4_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[4] apunta a task 4");
} else {

    sprintf(txdata, "HeaderPointer de OSRdy[4] apunta a null");

}

if (OSRdyList[5].HeadPtr == &task_led5_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[5] apunta a task 5");
} else {

    sprintf(txdata, "HeaderPointer de OSRdy[5] apunta a null");

}

if (OSRdyList[6].HeadPtr == &task_led6_TCB) {
    sprintf(txdata, "HeaderPointer de OSRdy[6] apunta a task 6");
} else {

    sprintf(txdata, "HeaderPointer de OSRdy[6] apunta a null");
```

Aquí, vamos haciendo las comparaciones y comprobaciones de cada una de las posiciones de la ready list, para así poder comprobar punto a punto qué tarea se encuentra en la primera posición de esa ready list en función de su prioridad , para el caso contrario se indica que ese puntero de listo por prioridades apunta a null, indicando que no existe ninguna tarea de esa peioridad en dicha estructura de datos y por tanto en la ready list.