PRÁCTICA 3. *Bitmap* de prioridades y *Ready List*

OBJETIVOS

En esta práctica se trabajará con los dos procesos del sistema operativo en tiempo real que gestionan el conjunto de Tasks que precisan un cambio de estado *Ready -> Running*

TAREAS PREVIAS

- ¿Qué es el *bitmap* de prioridades? Defina las características de la tabla que lo compone, entradas y tamaño.

- ¿Qué es la *Ready List*? Explique las características que definen la tabla que lo compone, entradas y tamaño.

  La ready list es la estructura de datos donde se colocan las tareas que han cambiado su estado a ready y como su nombre indica están esperando para poder entrar a ejecución.

- ¿Qué parámetro debe retornar el bitmap?

- ¿Qué Task está asociada a la última casilla del bitmap?

  Siempre ha de ser la idle task para estar disponible para ejecutarse siempre la ultima.

- Dada una tabla de bitmap compuesta por dos vectores de entrada: Bitmap[0] y Bitmap[1]. Cada vector que compone la tabla esta formada por una palabra de 8-bit. Suponga los siguientes valores decimales en cada vector:

  Bitmap[0] = 82 (dec)

  Bitmap[1] = 14 (dec)

  a) Determine el valor que retornará el bitmap de prioridades

  b) ¿Existe algún fallo en el valor decimal reportado en los vectores anteriores? Razone la respuesta

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)

La Task de monitorización debe disponer de:

- Nivel de prioridad: 1
- Periodicidad mínima/máxima: 10ms/5
- El usuario deberá monitorizar la tabla bitmap y reportar por UART, en valor binario y decimal, el estado dinámico de cada casilla que compone el vector. El vector dispondrá de un tamaño por defecto de 32-bit. De este modo, podrá visualizarse el estado actual de las Tasks en estado Ready.

```
OSTaskCreate(
        (OS_TCB *) & monitor_TCB,
        (CPU_CHAR *) "Tarea Monitorizacion",
        (OS_TASK_PTR) Task_monitor,
        (void *) 0,
        (OS_PRIO) 1,
        (CPU_STK *) & monitor_STK[0],
        (CPU_STK_SIZE) 0u,
        (CPU_STK_SIZE) 1024u,
        (OS_MSG_QTY) 0u,
        (OS_TICK) 10u, //10*System Tick period
        (void *) 0,
        (OS_OPT) (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
        (OS_ERR *) & os_err);
```

- Tenga en cuenta que el sistema operativo define el nombre de la tabla bitmap como
- `OSPrioTbl`
- La dirección del vector [0], por tanto, podrá accederse mediante
- `&OSPrioTbl[0]`

Aquí sacamos cada posición de la tabla a través de sus punteros, tenemos que aplicarle una máscara ya que

```
OS_ERR os_err;
CPU_DATA *cpudataptr;
cpudataptr = &OSPrioTbl[0];
unsigned int casilla[8];
while (1) {

    cpudataptr = &OSPrioTbl[0]; //Puntero a la tabla
    casilla[0] = [(*cpudataptr)&0x80000000] >> 31;
    casilla[1] = [(*cpudataptr)&0x40000000] >> 30;
    casilla[2] = [(*cpudataptr)&0x20000000] >> 29;
    casilla[3] = [(*cpudataptr)&0x10000000] >> 28;
    casilla[4] = [(*cpudataptr)&0x8000000] >> 27;
    casilla[5] = [(*cpudataptr)&0x4000000] >> 26;
    casilla[6] = [(*cpudataptr)&0x2000000] >> 25;
    casilla[7] = [(*cpudataptr)&0x1000000] >> 24;
    casilla[8] = [(*cpudataptr)&0x800000] >> 23;
```

Posteriormente imprimimos cada una de estas posiciones y aplicamos la temporización que se nos ha requerido en el apartado anterior, como hicimos en la práctica anterior.

```
sprintf(txdata, " Task Monitor\r\n");
sprintf("Direccion: &u", cpudataptr);
sprintf(txdata, "Contenido Tabla: &u"); // imprimir la tabla con una sola posición porque solo hay 8 posiciones
sprintf(txdata, "Posicion 0 &u", casilla[0]);
sprintf(txdata, "Posicion 1 &u", casilla[1]);
sprintf(txdata, "Posicion 2 &u", casilla[2]);
sprintf(txdata, "Posicion 3 &u", casilla[3]);
sprintf(txdata, "Posicion 4 &u", casilla[4]);
sprintf(txdata, "Posicion 5 &u", casilla[5]);
sprintf(txdata, "Posicion 6 &u", casilla[6]);
sprintf(txdata, "Posicion 7 &u", casilla[7]);
sprintf(txdata, "Posicion 8 &u", casilla[8]);
OSTimeDly(8000, OS_OPT_TIME_DLY, &os_err); //8000 ticks del RTOS -> 8s
```

Posteriormente solo enviamos por la UART y ajustamos la temporización requerida.