PRÁCTICA 5. Objetos de Kernel (Semaphore, Mutex, Queues)

OBJETIVOS

En esta práctica se pide trabajar con los conceptos teóricos y prácticos asociados a los objetos de Kernel de un RTOS, a nivel de sincronización de Tasks y control de recursos compartidos.

TAREAS PREVIAS

- ¿Qué es un semáforo binario? Describa posibles aplicaciones de uso de este elemento

  Un semáforo binario es un objeto de kernel que simplemente señala cuando un recurso compartido está siendo empleado y no puede ser empleado por otros procesos, esto es cierto que puede generar problemas de inversión de prioridades, ya que el semáforo no automatiza la herencia de prioridades y el mutex sí.

- ¿Qué es un mutex? Razone las ventajas de uso respecto a un semáforo binario y que tipo de operación extra presenta.

  Como se ha comentado previamente, los dos objetos son objetos de control de acceso a recursos compartidos y zonas críticas, la diferencia radica en que el mutex es capaz de solucionar el problema de la inversión de prioridades por medio de la herencia de esta.

- ¿Qué es la inversión de prioridades?

  La inversión de prioridades es una situación en la que una tarea de baja prioridad se comporta como una de alta, esto debido a una mala gestión de un recurso compartido, que quiere ser empleado por ambas, pero la de inferior prioridad se comporta y adquiere este recurso.

- Respecto a la pregunta anterior

  - ¿Cómo se denomina esta zona cuando hacemos uso de un semáforo?

    Zona de inversión de prioridades no acotada
  - ¿Cómo se denomina esta zona cuando hacemos uso de un mutex?
  - Zona de inversión de prioridades acotada.

- Explique que es una cola de mensajes en el ámbito de los RTOS.

  Una cola de mensajes es un objeto de kernel mediante el cual se pueden realizar comunicaciones entre tareas, esto debe ser así ya que estas tareas no comparten espacio de memoria y deben comunicarse a través de objetos globales de kernel

- ¿De qué manera puede comunicarse información local de una Task con otra de nuestra aplicación haciendo uso de una cola de mensajes?

  La cola de mensajes es un objeto global de kernel en el cual las tareas pueden depositar información que posteriormente otra pueda recoger, esto puede servir tanto para el traspaso de información de una tarea a otra como de objeto de sincronización, al poner una tarea a la escucha y espera de que otra encole algún mensaje.

APLICACIÓN

- Se deberán crear 6 Tasks de aplicación, con los parámetros de la Práctica 4.

- Mediante un mutex, se deberá proteger un recurso virtual compartido entre Task4 y Task5. Suponed que este recurso compartido es una operación compuesta por el acceso al LED + delay_ms(). Mediante LED + delay_ms(), simularemos el acceso a un recurso compartido que toma cierto tiempo en procesarse. Durante este tiempo, no debe existir ningún otro acceso.

  Primero creamos el mutex de manera global

```
OSMutexCreate(
        (OS_MUTEX *) & mutext4t5,
        (CPU_CHAR *) "Mutex tareas 4 y 5",
        (OS_ERR*) & os_err);
```

A continuación, dentro del hilo indicamos que el mutex debe
2bloquearse antes de acceder al recurso compartido (nuestro led), esto en ambas funciones, al terminar de emplear el recurso deberá liberarlo (post).

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

    while (1) {
        OSMutexPend(
                (OS_MUTEX *) & mutext4t5,
                (OS_TICK) 0,
                (OS_OPT) OS_OPT_PEND_BLOCKING,
                (CPU_TS *) 0,
                (OS_ERR *) & os_err
                );
        LED_BLUE2 = !LED_BLUE2_Read;

        OSTimeDly(2000, OS_OPT_TIME_DLY, &os_err);

        OSMutexPost(
                (OS_MUTEX *) & mutext4t5,
                (OS_OPT) OS_OPT_PEND_BLOCKING,
                (OS_ERR *) & os_err
                );
    }
}
```

```
void TASK_LED5(void) {
    OS_ERR os_err;

    while (1) {
        OSMutexPend(
                (OS_MUTEX *) & mutext4t5,
                (OS_TICK) 0,
                (OS_OPT) OS_OPT_PEND_BLOCKING,
                (CPU_TS *) 0,
                (OS_ERR *) & os_err
                );
        LED_BLUE2 = !LED_BLUE2_Read;

        OSTimeDly(5000, OS_OPT_TIME_DLY, &os_err);

        OSMutexPost(
                (OS_MUTEX *) & mutext4t5,
                (OS_OPT) OS_OPT_PEND_BLOCKING,
                (OS_ERR *) & os_err
                );
    }
}
```

- Mediante un semáforo binario, se deberá sincronizar Task2 con Task3, creando un sistema de sincronización bilateral entre ambas. Ambas Tasks deben reportar en que parte del hilo se encuentran mediante un mensaje de UART, de modo que el usuario compruebe que existe sincronización entre ambas tareas de manera correcta.
- Esto, lo logramos mediante el uso de dos semáforos, esto, para garantizar que ninguna de las dos tareas se interrumpen entre sí , y se cumpla un bucle constante y perfecto, que haga siempre vaya una detrás de la otra.

```
void TASK_LED2(void) {
    OS_ERR os_err;
    while (1) {
        sprintf(txdata, "TASK2 entrando en hilo  \r\n");

        OSSemPend(&semt2t3_one,
                0,
                OS_OPT_PEND_BLOCKING,
                0,
                &os_err
                );
        sprintf(txdata, "TASK3 Tras Pend  \r\n");


        LED_RED = !LED_RED_Read;

        OSSemPost(&semt2t3_two,
                OS_OPT_PEND_BLOCKING,
                &os_err
                );
        sprintf(txdata, "TASK3 Tras Primer Post  \r\n");

        OSSemPost(&semt2t3_one,
                OS_OPT_PEND_BLOCKING,

                 &os_err
                 );
        sprintf(txdata, "TASK3 Tras Segundo Post  \r\n");

        OSTimeDly(500, OS_OPT_TIME_DLY, &os_err);
    }
     void TASK_LED3(void) {
        OS_ERR os_err;
        while (1) {
            sprintf(txdata, "TASK3 entrando en hilo  \r\n");
            OSSemPend(&semt2t3_two,
                    0,
                    OS_OPT_PEND_BLOCKING,
                    0,
                    &os_err
                    );
            sprintf(txdata, "TASK3 Tras Pend  \r\n");


            LED_RED = !LED_RED_Read;

            OSSemPost(&semt2t3_one,
                    OS_OPT_PEND_BLOCKING,
                    &os_err
                    );
            OSSemPost(&semt2t3_two,
                    OS_OPT_PEND_BLOCKING,
                    &os_err
                    );
            OSTimeDly(500, OS_OPT_TIME_DLY, &os_err);
        }
```