Práctica 4

SISTEMAS TOLERANTES A FALLOS EN TIEMPO REAL

Sara Marcos Cornejo
José Francisco Romero Rodríguez

UNIVERSIDAD NEBRIJA

Índice:

Introducción:

En sistemas operativos de tiempo real, es común que varias tareas compartan recursos como pueden ser puertos de comunicación, memorias compartidas o dispositivos de entrada/salida. Para evitar problemas de acceso no controlado a estos recursos, se utilizan objetos de sincronización como los mutex.

El mutex es un objeto de sincronización que permite garantizar la exclusión mutua de un recurso compartido entre dos o más tareas. Esto significa que solo una tarea a la vez puede acceder al recurso compartido protegido por el mutex, evitando así la interferencia de otras tareas.

En esta práctica utilizaremos el entorno de desarrollo MPLAB y una placa DSPIC para implementar un ejemplo sencillo de uso de mutex en un sistema operativo de tiempo real. A través de la conexión de la placa DSPIC a un ordenador, podremos imprimir en tiempo real el comportamiento de las tareas, observando el efecto que tienen las diferentes configuraciones de los mutex sobre la prioridad y el rendimiento del sistema.

Implementación:

Para introducir el mutex en el código, es necesario seguir los siguientes pasos:

1. Inicialización del mutex: En primer lugar, es necesario inicializar el mutex antes de su uso. Para ello, se utiliza la función OSMutexCreate(), que recibe como argumento un puntero a una variable de tipo OS_MUTEX que se utilizará para referirse al mutex creado.
2. Bloqueo del mutex: Una vez inicializado, el mutex se utiliza para proteger el acceso a un recurso compartido. Para ello, se utiliza la función OSMutexPend(), que bloquea la tarea que intenta acceder al recurso compartido hasta que el mutex esté disponible.
3. Acceso al recurso compartido: Una vez que la tarea ha obtenido el control del mutex, puede acceder al recurso compartido protegido por él.
4. Liberación del mutex: Una vez que la tarea ha terminado de utilizar el recurso compartido, debe liberar el mutex para permitir que otras tareas puedan acceder al recurso compartido. Para ello, se utiliza la función OSMutexPost().

En el código presentado, se está utilizando un Mutex (semáforo binario) para controlar el acceso a un recurso compartido, que en este caso es la impresión de información en una UART.

Primero, se crea el Mutex utilizando la función OSMutexCreate(), la cual recibe como parámetros la dirección de memoria de la estructura que representa el Mutex, una cadena de caracteres para identificarlo y un puntero a una variable de error.

Luego, en cada tarea se utiliza la función OSMutexPend() para esperar por el Mutex y bloquear el acceso al recurso compartido. Esta función también recibe como parámetros la dirección de memoria del Mutex, el tiempo de espera (en este caso 0 para indicar que la tarea se bloqueará hasta que el Mutex esté disponible), opciones de espera y punteros a una variable de marca temporal y de error.

Una vez que se ha obtenido el Mutex, se realiza la acción que requiere el recurso compartido (en este caso, imprimir información en la UART) y luego se libera el Mutex mediante la función OSMutexPost(). Esta función recibe como parámetros la dirección de memoria del Mutex, opciones de liberación y un puntero a una variable de error.
Cabe destacar que, al utilizar un Mutex, se garantiza que sólo una tarea a la vez podrá acceder al recurso compartido, lo que previene condiciones de carrera y garantiza la integridad de los datos compartidos.

```
//Mutex
OS_MUTEX MyMutex;
CPU_TS ts;
```

```
//Mutex create
OSMutexCreate((OS_MUTEX *) &MyMutex,
              (CPU_CHAR * ) "Mutex para control de recursos compartidos",
              (OS_ERR *   ) &os_err);
```

Pend:

```
void TASK_LED1 (void)
{
    OS_ERR os_err;
    //OSSchedRoundRobinCfg(DEF_ENABLED, (OS_TICK)100u, &os_err);

    while(1)
    {

        OSMutexPend((OS_MUTEX * ) &MyMutex,
                    (OS_TICK    ) 0,
                    (OS_OPT     ) OS_OPT_PEND_BLOCKING,
                    (CPU_TS *   ) 0,
                    (OS_ERR *   ) &os_err);

        sprintf(txdata,"Task1.prio:%u address:%lu priod: %u \r\n ", MyMutex.OwnerOriginalPrio, MyMutex.OwnerTCBPtr, task_led1_TCB.Prio, "\r\n");
        EnviarString(txdata, Uart3);
        LED_RED = !LED_RED_Read;

        OSMutexPost((OS_MUTEX * ) &MyMutex,
                    (OS_OPT     ) OS_OPT_POST_NONE,
                    (OS_ERR *   ) &os_err);

        OSTimeDly(500,OS_OPT_TIME_DLY, &os_err);  //500 ticks del RTOS -> 500ms
    }
}
```

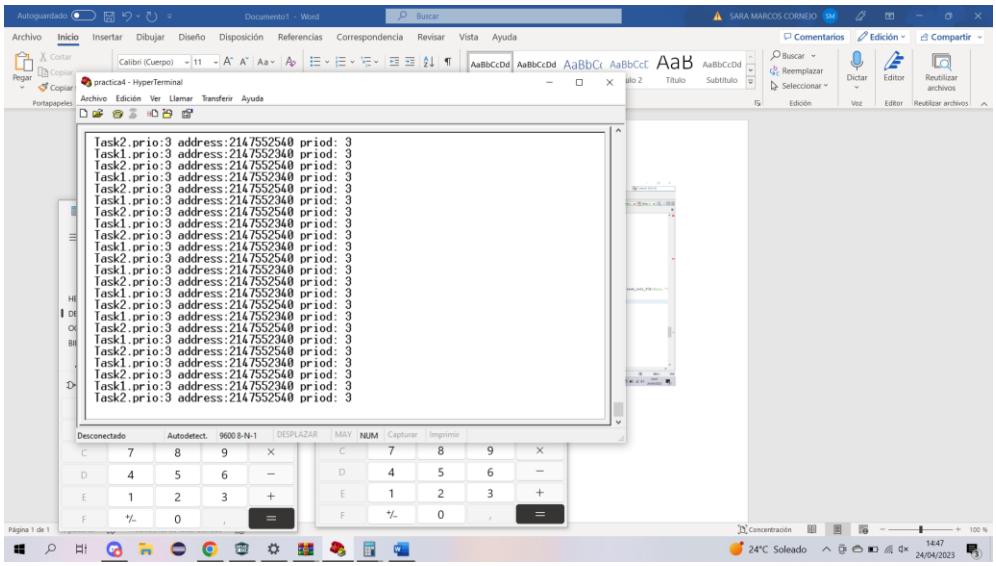
Post:

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


    while(1)
    {
        OSMutexPend((OS_MUTEX * ) &MyMutex,
                    (OS_TICK    ) 0,
                    (OS_OPT     ) OS_OPT_PEND_BLOCKING,
                    (CPU_TS *   ) 0,
                    (OS_ERR *   ) &os_err);
        sprintf(txdata,"Task2.prio:%u address:%lu priod: %u \r\n ", MyMutex.OwnerOriginalPrio, MyMutex.OwnerTCBPtr, task_led2_TCB.Prio, "\r\n");
        EnviarString(txdata, Uart3);
        LED_ORANGE = !LED_ORANGE_Read;
        OSMutexPost((OS_MUTEX * ) &MyMutex,
                    (OS_OPT     ) OS_OPT_POST_NONE,
                    (OS_ERR *   ) &os_err);
        OSTimeDly(500,OS_OPT_TIME_DLY, &os_err);  //500 ticks del RTOS -> 500ms
        //delay · 2 para marcar la periodicidad
    }
}
```

Aquí podemos observar el HyperTerminal, primero debemos asegurarnos de que la placa esté conectada correctamente a nuestro ordenador y que el puerto serie esté disponible. Una vez que hayamos verificado esto, podemos abrir HyperTerminal en nuestro ordenador y configurar la conexión serial.

Al imprimir:



La comparación que se está haciendo en este caso es entre task_led1 y task_led2, y se está hablando de la posibilidad de una inversión de prioridades en caso de que hubiera tareas de prioridades diferentes a su izquierda o derecha.

En este caso, ambas tareas tienen la misma prioridad, que es de 3. Por lo tanto, no hay posibilidad de que ocurra una inversión de prioridades en este caso específico.

Sin embargo, en general, si hubiera tareas con prioridades diferentes a su izquierda o derecha, podría haber una inversión de prioridades en algunas circunstancias. Esto se debe a que una tarea de baja prioridad puede bloquear un recurso que necesita una tarea de alta prioridad, lo que hace que la tarea de alta prioridad se bloquee y deba esperar a que la tarea de baja prioridad termine de utilizar el recurso.

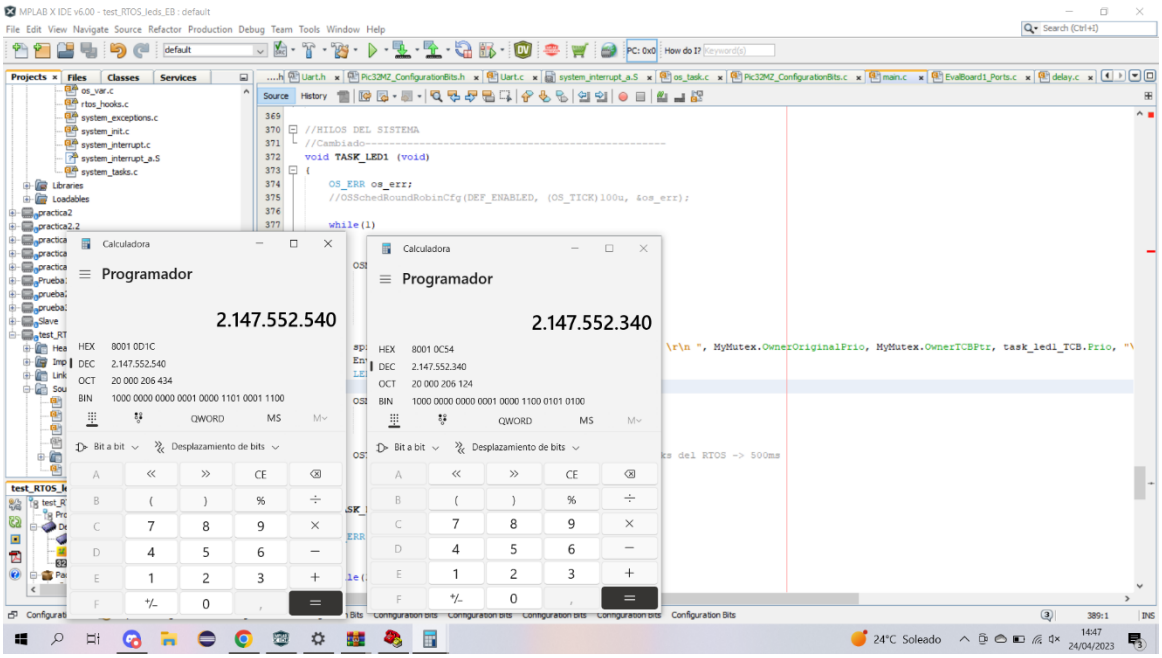
En esta situación, si la tarea de baja prioridad continúa ejecutándose durante un período prolongado de tiempo, la tarea de alta prioridad puede quedar inactiva y no cumplir con su plazo de entrega. Esto se conoce como inversión de prioridades.

La dirección de la tarea 1 es 2147552340 y la dirección de la tarea 2 es 2147552540, lo que significa que hay una diferencia de 200 entre ellas. Esta diferencia se debe a la forma en que el compilador asigna la memoria para las tareas en la memoria RAM.

Cuando se compila el código, el compilador asigna un espacio en la memoria para cada tarea. La ubicación exacta en la memoria se determina según varios factores, como el tamaño de la tarea, la ubicación de otras tareas y las variables globales, entre otros.

En este caso, es posible que la tarea 1 haya sido asignada primero y, por lo tanto, se colocó en una ubicación de memoria más baja. La tarea 2 se asignó después y se colocó en una ubicación de memoria más alta, lo que resulta en una diferencia de 200 entre las direcciones.

Es importante tener en cuenta que esta diferencia de dirección no afecta el funcionamiento del programa ni de las tareas. Las tareas se comunican a través del uso de punteros, y el sistema operativo en tiempo real se encarga de gestionar y coordinar las tareas según sus prioridades y el uso de recursos compartidos.



Conclusiones:

En esta práctica hemos visto cómo implementar y utilizar un mutex para controlar el acceso a recursos compartidos entre varias tareas en un sistema operativo en tiempo real (RTOS). Hemos utilizado MPLAB y una placa dspic para imprimir mensajes en una terminal de comunicación.

El mutex nos permite garantizar que solo una tarea a la vez accede a un recurso compartido, evitando así conflictos y errores. Hemos creado el mutex y lo hemos utilizado en dos tareas diferentes que comparten acceso a la misma UART, para asegurarnos de que no haya dos tareas escribiendo en la UART al mismo tiempo.

También hemos observado las prioridades de las tareas y cómo pueden invertirse en ciertas situaciones, como cuando una tarea de menor prioridad tiene acceso a un recurso compartido que necesita una tarea de mayor prioridad. Se ha tratado el tema acerca de cómo solucionar este problema utilizando técnicas como la inversión de prioridades.

En general, esta práctica nos ha permitido entender mejor el uso de mutex en RTOS y cómo pueden ayudarnos a garantizar la correcta ejecución de múltiples tareas en un sistema.