Práctica 3

SISTEMAS TOLERANTES A FALLOS EN TIEMPO REAL

Sara Marcos Cornejo
José Francisco Romero Rodríguez

UNIVERSIDAD NEBRIJA

Índice:

Introducción:

Esta práctica se enfoca en la implementación de semáforos en sistemas operativos en tiempo real (RTOS). Un semáforo es un mecanismo de sincronización utilizado para controlar el acceso a recursos compartidos en un sistema multiusuario. En un sistema operativo en tiempo real, el uso de semáforos es una técnica común para prevenir la competencia de recursos compartidos y garantizar una ejecución coherente y ordenada de las tareas del sistema.

En esta práctica, se presentará un código de ejemplo que utiliza semáforos en un sistema operativo en tiempo real para controlar el acceso a un recurso compartido por dos tareas. Se explorarán las funciones básicas de semáforos, incluyendo la creación, pending y post, y se demostrará cómo se pueden utilizar estas funciones para garantizar una ejecución ordenada y sincronizada de tareas en un sistema multiusuario.

Procedimiento:

```
//Semaphore
OS_SEM MySemaphore;

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
    }
}
```

El código presentado implementa el uso de semáforos en el sistema operativo de tiempo real MLAPB con DSPIC para sincronizar dos tareas (TASK_LED1 y TASK_LED2) que comparten un recurso (el LED).

En primer lugar, se crea el semáforo utilizando la función "OSSemCreate", donde se especifica un nombre para identificar el semáforo, el valor inicial (0 en este caso, lo que significa que el recurso está ocupado), y se guarda en la variable "MySemaphore".

Luego, en cada tarea, se utiliza la función "OSSemPend" para esperar a que el semáforo esté disponible (valor = 1). Una vez que está disponible, la tarea realiza su acción (en este caso, cambiar el estado del LED y enviar un mensaje a través del puerto serie) y luego utiliza la función "OSSemPost" para liberar el semáforo (establecer su valor de nuevo en 1) para que la otra tarea pueda realizar su acción.

Se utiliza la variable "ts" para obtener un timestamp que se utiliza en la función "OSSemPend" para conocer el momento en que se liberó el recurso compartido. Finalmente, ambas tareas incluyen una función "OSTimeDly" que introduce un delay en la ejecución de la tarea para marcar la periodicidad de la acción a realizar.

La tarea "TASK_LED1" enciende y apaga un LED rojo cada 500 ms y luego libera el semáforo para que la tarea "TASK_LED2" pueda tomar el control. La tarea "TASK_LED2" hace lo mismo con un LED naranja y luego libera el semáforo para que la tarea "TASK_LED1" pueda tomar el control nuevamente.

Resultado:

El resultado es que las dos tareas se ejecutan alternativamente, con una tarea esperando a que la otra termine antes de continuar. El uso del semáforo asegura que cada tarea tenga acceso exclusivo al recurso compartido (en este caso, los LED), evitando posibles conflictos de acceso concurrente.

La inversión de prioridades es un problema común en sistemas multiproceso o multitarea, en el que un proceso o tarea de baja prioridad impide que un proceso o tarea de alta prioridad avance, debido a que el proceso de baja prioridad tiene el control de un recurso compartido necesario por el proceso de alta prioridad.

Los semáforos son una herramienta comúnmente utilizada para resolver este problema. La idea básica es que un semáforo se utiliza para proteger un recurso compartido, y solo un proceso a la vez puede tener acceso al recurso. Si un proceso intenta acceder al recurso mientras está bloqueado por otro proceso, se pone en espera hasta que el recurso esté disponible.

Sin embargo, la prioridad de inversiones puede ocurrir incluso con semáforos, si no se utilizan adecuadamente. Esto puede suceder cuando un proceso de baja prioridad adquiere un semáforo, bloqueando el acceso al recurso compartido, y un proceso de alta prioridad intenta acceder al recurso pero debe esperar a que se libere el semáforo. Si un tercer proceso de prioridad media intenta adquirir el mismo semáforo, queda bloqueado por el proceso de baja prioridad, y esto puede generar un bloqueo del sistema, lo que se conoce como "inversión de prioridad".

Para solucionar este problema, se pueden utilizar diferentes técnicas, como la herencia de prioridad o la eliminación de la inversión de prioridad mediante la utilización de semáforos binarios o mutex. En la herencia de prioridad, si un proceso de baja prioridad tiene el recurso bloqueado y un proceso de alta prioridad necesita acceder a ese recurso, el sistema eleva temporalmente la prioridad del proceso de baja prioridad a la del proceso de alta prioridad, permitiendo que el proceso de alta prioridad acceda al recurso. Por otro lado, los semáforos binarios o mutex permiten que solo un proceso a la vez tenga acceso al recurso compartido, evitando la competencia por el acceso al recurso y eliminando así la inversión de prioridad.

Conclusiones:

En conclusión, la implementación de semáforos en un sistema operativo en tiempo real es fundamental para garantizar la sincronización y la protección de recursos compartidos entre tareas. El uso de semáforos permite evitar condiciones de carrera y garantizar la exclusión mutua, asegurando que solo una tarea tenga acceso a un recurso compartido en un momento determinado.

Además, es importante tener en cuenta la prioridad de inversiones que puede surgir cuando se utilizan semáforos. La prioridad de inversiones se produce cuando una tarea de baja prioridad adquiere un semáforo que una tarea de alta prioridad necesita para continuar. Para evitar este problema, se pueden utilizar técnicas como la herencia de prioridad o la prioridad de herencia inversa.

En resumen, el uso adecuado de los semáforos en un sistema operativo en tiempo real puede mejorar la eficiencia y la seguridad de un sistema, asegurando una mejor gestión de los recursos y evitando problemas potenciales de exclusión mutua y prioridad de inversiones.