Práctica 5

SISTEMAS TOLERANTES A FALLOS EN TIEMPO REAL

Sara Marcos Cornejo
José Francisco Romero Rodríguez

UNIVERSIDAD NEBRIJA

Índice:

Introducción:

La cola de mensajes, también conocida como message queue, es una estructura de datos que permite la comunicación asincrónica y la transferencia de información entre diferentes componentes de un sistema o entre sistemas distribuidos.

Es una forma eficiente de gestionar la comunicación y la sincronización entre procesos, ya que permite que los componentes se comuniquen de manera asíncrona, sin necesidad de estar simultáneamente activos y sin bloquear el proceso principal.

Implementación:

Para introducir la cola de mensajes en el código, es necesario seguir los siguientes pasos:

1. **Definición de la estructura de datos de la cola de mensajes:** Primero, debes definir la estructura de datos que se utilizará para representar la cola de mensajes. Puedes utilizar un arreglo, una lista enlazada u otra estructura según tus necesidades. Cada elemento de la estructura de datos debe contener la información necesaria para representar un mensaje, como un identificador único y los datos o referencias asociados al mensaje.
2. **Inicialización de la cola de mensajes**: Antes de utilizar la cola de mensajes, debes inicializarla. Esto implica crear la estructura de datos correspondiente y realizar cualquier otra configuración necesaria, como establecer el tamaño máximo de la cola si aplica.
3. **Productor: Agregar mensajes a la cola:** El productor es responsable de agregar mensajes a la cola de mensajes. Cuando haya un nuevo mensaje para enviar, el productor crea una nueva instancia de mensaje, asigna un identificador único y establece los datos o referencias asociados al mensaje. A continuación, el mensaje se agrega al final de la cola de mensajes.
4. **Consumidor: Extraer mensajes de la cola:** El consumidor es responsable de extraer y procesar los mensajes de la cola de mensajes. El consumidor puede ejecutarse en un proceso o hilo separado y continúa en un bucle mientras haya mensajes en la cola. En cada iteración del bucle, el consumidor extrae el primer mensaje de la cola y realiza las operaciones necesarias según el tipo de mensaje.
5. **Sincronización y gestión de concurrencia**: Es importante asegurarse de que el acceso a la cola de mensajes esté sincronizado y gestionado adecuadamente en entornos concurrentes para evitar problemas de concurrencia, como condiciones de carrera. Puedes utilizar mecanismos de sincronización como cerrojos, semáforos o mutex para proteger el acceso a la cola y garantizar que solo un hilo o proceso acceda a ella a la vez.
6. **Finalización y liberación de recursos**: Al finalizar el uso de la cola de mensajes, debes liberar los recursos asociados. Esto implica liberar la memoria utilizada por la estructura de datos de la cola y realizar cualquier otra limpieza necesaria.

El código implementado realiza las siguientes acciones:

- Se inicializa el RTOS y se configura el modo de rendimiento del Clicker2.
- Se configuran los pines GPIO de la placa y se establece la configuración de la UART para la comunicación serie.
- Se crean varias tareas que controlan los LEDs de la placa y envían mensajes a través de UART.
    - Se crea una cola para la comunicación entre las tareas de la siguiente forma:

```
OSQCreate((OS_Q *)&MyExternalQueue,
          (CPU_CHAR *)"My Queue",
          (OS_MSG_QTY )20,
          (OS_ERR *)&os_err);
```

- Se verifica si hubo errores al crear las tareas y se inicia el sistema operativo.
- Se ejecutan las tareas en paralelo, controlando los LEDs, enviando mensajes y realizando retardos.
- Las tareas TASK_LED1, TASK_LED2, TASK_LED3, TASK_LED4, TASK_LED5 y TASK_LED6 son funciones individuales que se ejecutan en paralelo en el sistema operativo. Cada tarea tiene una funcionalidad específica:

    - **TASK_LED1**: En cada iteración del bucle infinito, la tarea envía un número incrementado a través de la cola **MyExternalQueue**, envía un mensaje a través de UART indicando el valor enviado. Luego, se realiza un. Lo hace mediante la función post:

```
void TASK_LED1 (void)
{
    unsigned int var_char = 0;
    OS_ERR os_err;
    OSSchedRoundRobinCfg(DEF_DISABLED, (OS_TICK)100u, &os_err);
    while(1)
    {

        OSQPost((OS_Q    *) &MyExternalQueue,
                (void*) (unsigned int)(var_char),
                 sizeof(var_char),
                (OS_OPT       ) OS_OPT_POST_FIFO,
                (OS_ERR      *) &os_err);

        sprintf(txdata,"Practica 5, task1, variable enviada %d\r\n",var_char );
        EnviarString(txdata, Uart3);
        LED_RED = !LED_RED_Read;

        var_char++;

            OSTimeDly(1000,OS_OPT_TIME_DLY, &os_err);
    }
}
```

- **TASK_LED2**: En cada iteración del bucle infinito, la tarea espera hasta que haya un elemento disponible en la cola y luego lee el valor recibido. Después, envía un mensaje a través de UART indicando el valor recibido y realiza un retardo de 500 milisegundos antes de repetir el bucle.

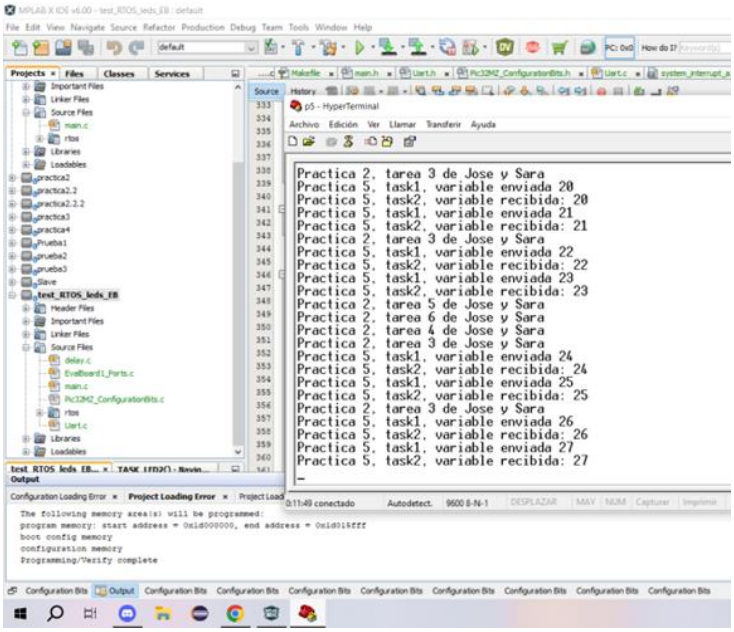
```
void TASK_LED2 (void)
{
    unsigned int prueba_size;
    OS_ERR os_err;
    unsigned int var_rec;
    while(1)
    {
        var_rec=OSQPend((OS_Q   *) &MyExternalQueue,
                (OS_TICK     ) 0,
                (OS_OPT      ) OS_OPT_PEND_BLOCKING,
                               &prueba_size,
                (CPU_TS     *) &ts,
                (OS_ERR     *) &os_err);
        sprintf(txdata,"Practica 5, task2, variable recibida: %d\r\n",var_rec );
        EnviarString(txdata, Uart3);

            OSTimeDly(500,OS_OPT_TIME_DLY, &os_err);

    }
}
```

- Las otras tareas controlan diferentes LEDs en la placa y enviando mensajes a través de UART, pero con retardos y mensajes diferentes.

En resumen, el programa utiliza el RTOS para coordinar varias tareas que se comunican a través de UART. Cada tarea tiene su propia funcionalidad y se ejecuta en paralelo, permitiendo un control simultáneo de los LEDs y la comunicación serie. La cola MyExternalQueue se utiliza para intercambiar datos entre las tareas de manera segur

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:



El programa entra en un bucle infinito donde se ejecutan las tareas y se envían mensajes a través de UART.

Podemos observar como funcionan a medida que se ejecutan las tareas, hay mensajes en el Hyperterminal enviados por las tareas TASK_LED1 y TASK_LED2. Estos mensajes indican los valores enviados y recibidos a través de la cola MyExternalQueue.

Conclusiones:

En resumen, el código proporcionado muestra la implementación de un sistema en tiempo real utilizando un RTOS. El sistema consiste en varias tareas que controlan los LEDs de la placa y se comunican entre sí a través de una cola.

Las tareas TASK_LED1 y TASK_LED2 son diferentes en su funcionamiento. La tarea TASK_LED1 controla un LED específico en la placa, envía un valor incrementado a través de la cola y muestra mensajes en el Hyperterminal indicando el valor enviado. Por otro lado, la tarea TASK_LED2 espera a que haya un elemento disponible en la cola, recibe el valor y muestra mensajes en el Hyperterminal indicando el valor recibido.

El programa se ejecuta en un bucle infinito después de la inicialización del RTOS y las configuraciones iniciales. A medida que las tareas se ejecutan en paralelo, se producen intercambios de información a través de la cola y se muestran mensajes en el Hyperterminal. Además, los estados de los LEDs en la placa se actualizan según lo indicado por las tareas.

Conectar el dispositivo a un Hyperterminal permite observar la interacción entre las tareas y el intercambio de datos a través de UART, así como los cambios en los estados de los LEDs. Esto proporciona una forma de monitorear y depurar el sistema en tiempo real mientras se ejecuta.

En general, este código ejemplifica la estructura y la implementación básica de un sistema en tiempo real utilizando un RTOS y destaca la comunicación entre tareas y la sincronización mediante el uso de colas. Además, muestra cómo se pueden visualizar los resultados y el estado del sistema a través de una conexión con un Hyperterminal.