

CÁTEDRA DE SISTEMAS OPERATIVOS II

Departamento de Computación
FCEFN - UNC

Trabajo Practico Nro4

Agustín Colazo

Córdoba, 2 de abril de 2017

Índice

1. Introducción
2. Suposiciones
3. Requerimientos
4. Diseño de la solución
5. Implementación
6. Resultados
7. Conclusiones
8. Fuentes

Introducción

Este trabajo se desarrollo sobre una placa de desarrollo K64F. El trabajo consiste en generar datos que simularan ser procedentes de un sensor de temperatura. Y también generar cadenas de tamaño variable y caracteres pseudoaleatorios al pulsar un botón, que simulara ser una entrada ingresada en un teclado por un usuario. Estos datos deben ser enviados por comunicación serial a una terminal, que recibirá e imprimirá estos datos.

Suposiciones

- Uso de una placa de desarrollo K64F.
- Bootloader de Segger para K64F.
- La K64F estará conectada por un puerto serial a un dispositivo que recibirá los datos.
- Uso del sistema operativo FreeRTOS con Tracealyzer habilitado.
- Uso del programa Tracealyzer.

Requerimientos

- Enviar los datos por comunicación serial.
- La tasa de baudios debe ser 115200.
- Habrá un bit de parada y ocho bits de datos.
- Al presionar un botón, se genera una cadena con caracteres mayúsculas aleatorios, que sera enviado.
- Las cadenas generadas deben ser de tamaños variables.
- Se puede enviar la cadena vacía.
- La acción anterior se debe ejecutar al presionar el botón, no debe activarse por mantenerse pulsado. Tampoco debe activarse al soltarlo.
- Se generara un dato aleatorio que simulara ser procedente de un sensor de temperatura, este dato debe ser de un byte.
- El dato recibido del sensor se procesa en la placa para obtener la temperatura en centígrados.
- La temperatura en centígrados se envía por comunicación serial.

Diseño de la solución

Se implementaran tres tareas.

También se creara un tipo de dato struct "message" que contiene un puntero a char y un entero que indicara el tipo de dato que se enviá.

El tipo de dato se utiliza para que la tarea que recibe los datos, pueda decidir como procesarlos.

El puntero apuntara al dato a enviar, este puede ser una cadena de texto o un byte.

La primera tarea generara datos aleatorios que simularan datos procedentes de un sensor de temperatura digital. Segun la temperatura medida se recibiran datos de un byte del sensor. Esta tarea almacena el byte en el puntero char de la estructura, ademas aloca un byte en memoria. Cabe destacar que los punteros char en C apuntan a un byte de memoria. Tambien almacenara el tipo de dato para que este pueda luego ser procesado. Por ultimo, pondrá el struct en una cola bloqueadora.

Esta tarea debe ejecutarse cada 4 segundos.

La segunda tarea leerá el estado del pulsador SW3. Cuando se presiona el pulsador se generara una cadena aleatoria de tamaño variable. Si se mantiene el pulsador no pasara nada (no se generaran nuevas cadenas). Recién una vez que es soltado el botón, y se vuelve a presionar, se volverá a generar una cadena distinta. Se aloca memoria suficiente para almacenar la cadena. La cantidad de memoria alocada variara con el tamaño de la cadena generada. La cadena se encapsulara en la estructura con el tamaño de la misma y tambien se guardara el tipo de dato que le corresponde. Luego, se pondrá en la cola bloqueadora.

Esta tarea debe ejecutarse cada 50 o 100 ms.

Para la tarea que genera cadenas el entero sera de valor 0, y para la tarea que toma datos del sensor el tipo de dato tendra el valor de 1.

La tercer tarea recibe los datos de la cola bloqueadora. Depende el tipo de dato que se reciba lo procesara de distintos modos.

Si es una cadena de texto lo enviara directamente.

Si es un byte del sensor de temperatura, se transformara el dato a temperatura en centigrados. Luego se enviara este dato.

Esta tarea no tendra retardo. Pero se bloqueara cuando no haya datos disponibles.

Al enviar un dato a la cola bloqueadora, si la cola esta llena, la tarea se bloqueara hasta que haya un espacio libre.

Al recibir datos de la cola bloqueadora, si la cola esta vaciá, la tarea se bloqueara hasta que haya un dato disponible.

En la primera y segunda tarea se debe alocar la menor cantidad de memoria necesaria para guardar las cadenas de texto o el byte del sensor. En la tercera tarea, luego de enviar los datos por comunicación serial, esta memoria debe ser liberada.

Solo habrá una cola bloqueadora, y esta sera la misma para las tres tareas. No se debe crear una variable global para la cola, esta debe ser instanciada en el main. Y se enviara como parámetro a las tareas creadas.

Implementación

El trabajo se desarrollo en el ambiente de desarrollo Kinetis Studio Design. Se programo para una placa K64F usando FreeRTOS.

Tanto para el desarrollo del software como para la lectura serial de los datos y para el análisis con Tracealyzer se utilizo una computadora portátil con Windows 10 instalado. Para la lectura serial de los datos se utilizo el programa Putty.

Algunas consideraciones del programa desarrollado:

- Para alocar y liberar memoria se utilizo la librería heap4.c.
 - Esta librería permite el alojamiento de memoria de modo seguro.
 - Es mas eficiente que la librería estándar de C.
 - Tiene un buen manejo sobre la memoria. Evitando, hasta cierto punto, la fragmentación.
 - Al alocar memoria: combina bloques pequeños adyacentes en uno mas grande. Así hace un uso eficiente de la memoria.
- La tarea que comprueba el botón pulsado.
 - El estado del botón se obtiene por polling. No se utilizaron interrupciones.
 - La solución para evitar generar varias cadenas al mantener el botón pulsado se implemento por software.
 - Se utilizo la función rand() para generar cadenas de tamaño variable y caracteres pseudoaleatorios.
 - Aloca la minima memoria necesaria para almacenar la cadena.
- En la tarea que genera valores aleatorios.
 - Se utilizo la función rand() para generar valores pseudoaleatorios.
 - Se utiliza uint8_t para almacenar los valores pseudoaleatorios.
 - El uint8_t se castea a char para poder almacenarlo en la estructura.
 - Se aloca un byte de memoria.
- Respecto de la tarea que enviá los datos por UART.
 - Utiliza una sola cola bloqueadora para leer los datos de las otras dos tareas.
 - Cuando no hay datos en la cola, se bloquea la tarea.
 - No comprueba la integridad de los datos recibidos.
 - Comprueba el tipo de dato que se recibe con la variable "tipo".
 - El sensor de temperatura mide de 0C a 255C, y de modo lineal. Por lo tanto la transformacion es uno a uno.
 - Se le agrega la etiqueta "Receptor:" a los datos enviados para mostrar que son cadenas de texto enviadas por la tarea que recibe los datos del buffer.
 - Libera la memoria.

Resultados

En Putty se puede observar lo siguiente.

COM3 - PuTTY

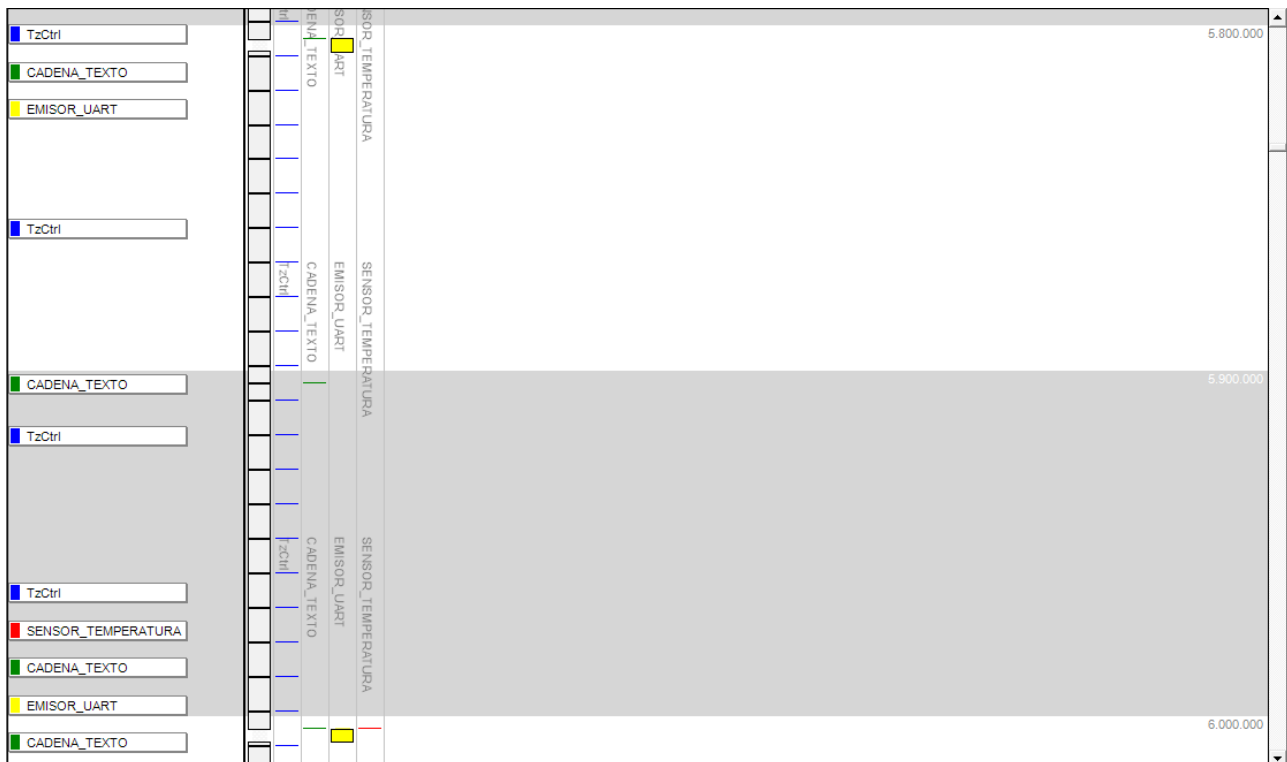
```
Receptor: La temperatura sensada es de: 91
Receptor: DUZSITVADWJAYLIEZMPATSRAJUKOTQIO
Receptor: UAHOAUI
Receptor: MXNRTGPGWAUKURFVTOIIZBJETJNXRQLHOIOLXQ
Receptor: G
Receptor: La temperatura sensada es de: 52
Receptor: KMWICIXNDHETY
Receptor: WWSLCHFRNKRNEAUUEWQKM
Receptor: VXGVVKRRXUWOOB
Receptor: LAWWSITFCECYNBIYWBRAKDMKBJTEIQRNSJBLPA
Receptor: YCERDOGQIJSABHFMW
Receptor: La temperatura sensada es de: 61
Receptor: NBXCZMCNRWJFEAEUPROHQY
Receptor: GUMMQIJGIFMIBTGCYPNJBMTIHPFNKLFKFA
Receptor: IXENFXZX
Receptor: BRUVAOTGEOMSPWQR
Receptor: La temperatura sensada es de: 36
Receptor: DBZXCUKNVKKLMGYLFFOSBZTHLLFGBBZTFTRN
Receptor: La temperatura sensada es de: 69
Receptor: La temperatura sensada es de: 70
Receptor: La temperatura sensada es de: 68
```

Allí se pueden observar las cadenas de texto enviadas por comunicación serial.

Aquellas que pertenecen a la tarea que genera números pseudoaleatorios son las que empiezan con “La temperatura sensada es de:”.

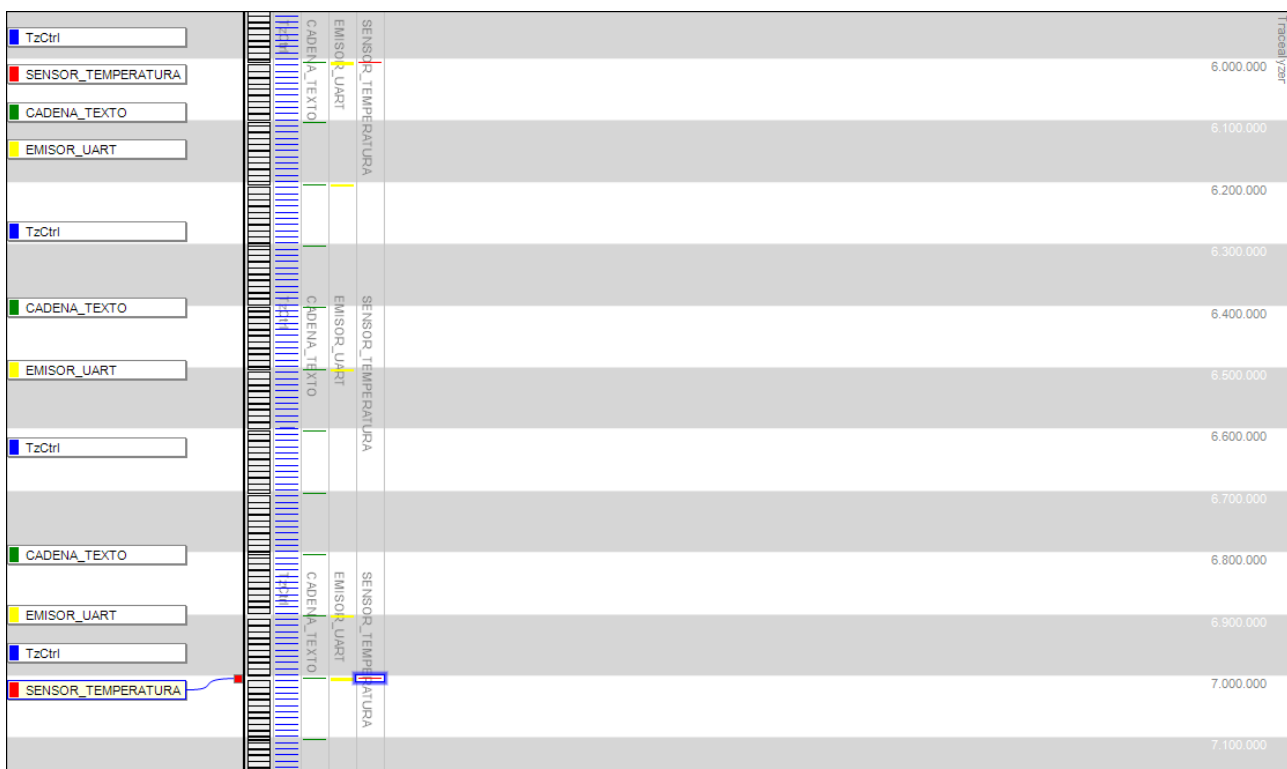
También se puede observar que las cadenas generadas por la tarea del pulsador varían en tamaño y en los caracteres que las componen.

A continuación se mostraran las imágenes obtenidas del Tracealyzer y se hará un análisis sobre lo observado en cada imagen.



En la primera imagen se observa varias ejecuciones de la tarea CADENA_TEXTO. Se puede notar que no siempre que se ejecuta esta tarea, se ejecuta la tarea EMISOR_UART. Esto es así ya que CADENA_TEXTO hace polling, por lo tanto no siempre pone un dato en la cola. EMISOR_UART solo se ejecutara cuando hay un dato en la cola para ser enviado.

Podemos notar dos ejecuciones de la tarea CADENA_TEXTO en la mitad superior de la imagen. En la primera el botón se presiono por primera vez y el dato fue enviado. En la segunda, el botón estando presionado o no, no iba a enviar ningún dato ya que este fue enviado antes.



En esta imagen se pueden ver dos cosas.

Por un lado podemos notar que cada vez que se ejecuta la tarea

SENSOR_TEMPERATURA también se ejecuta la tarea que envía los datos a la terminal.

Esto es así porque SENSOR_TEMPERATURA cada vez que se ejecuta genera un dato y lo pone en la cola.

Por otro lado, CADENA_TEXTO genera y envía datos de modo asíncrono, como podemos ver en la primera y segunda imagen. No cada vez que se ejecuta, enviara datos.

Conclusiones

FreeRTOS es un sistema operativo de tiempo real. Este sistema demuestra ser una herramienta poderosa (y de uso gratuito) a la hora de desarrollar un trabajo sobre un sistema embebido.

FreeRTOS permite crear varias tareas que se ejecutaran concurrentemente. Es una herramienta que facilita las operaciones sobre componentes de bajo nivel, como lo es el hardware. Ya que implementa muchas funciones que resuelven cosas como habilitar o deshabilitar interrupciones o módulos. Nos permite trabajar sobre distintas placas, usando un mismo sistema operativo, sin tener que estudiar tanto los detalles de cada placa. Facilita mucho la tarea del programador al ya tener varias cosas implementadas, este no debe empezar desde cero. Sino que tiene una base importante que le resuelve varias dificultades como el manejo de la memoria. Además que trae implementados ciertos tipos de datos especiales como los semáforos y las colas.

Fuentes

<http://www.freertos.org/a00111.html>

<http://www.freertos.org/a00018.html>

https://percepio.com/docs/FreeRTOS/manual/Recorder.html#Trace_Recorder_Library_Streaming_Mode

https://www.element14.com/community/community/designcenter/kinetis_kl2_freedom_board/blog/2016/04/15/getting-started-with-freertos-and-ksdk-20-creating-a-new-freertos-with-ksdk-v20-application-using-frdm-k64f-freedom-board