UNC | Universidad Nacional

Cátedra de Sistemas Operativos II

Trabajo Práctico Nº IV

Martín Illesca

23 de julio del 2020

Índice

Introducción

Propósito

El proposito de este trabajo practico es el uso y aplicación de un sistema de tiempo real que opera sobre un RTOS (Sistema operativo de tiempo real).

Los sistemas de tiempo real son aquellos en los que las tareas o procesos tratan de responder a sucesos del mundo exterior esto debe ser en tiempo real lo que implica ciertas restricciones temporales, estos sistemas deben contar con determinismo,sensibilidad, control de usuario y fiabilidad. Lo que representa un comportamiento predecible y un manejo importante del usuario en cuanto a prioridades y a comportamiento del scheduler.

Los sistemas operativos de tiempo real son aquellos que han sido diseñados para correr aplicaciones de tiempo real pudiendo cumplir todos los requisitos de estas.

Ámbito del Sistema

El sistema esta diseñado para ser corrido en un microcontrolador lm3s811 con un procesador Cortex M3 o bien por un similador con estas caracteristicas configuradas, usa FreeRtos como sistema operativo de tiempo real este viene incluido en el proyecto

Descripción General

Perspectiva del Producto

El producto es una aplicación de tiempo real sencilla que utiliza FreeRTOS, la misma cuenta con, en principio, seis tareas mas una septima propia de este sistema operativo que es la idle task, esta es una tarea que ocupa el procesador cuando no hay ninguna otra corriendo.

Ademas cuenta con un sistema productor-consumidor los cuales van tomando de y otorgando cadenas a una cola que se usa a modo de buffer, hay cuatro productores y un consumidor.

Se cuenta tambien con una tarea de tipo TOP que se encarga de mostrar estadisticas de todas las tareas del sistema operativo.

Funciones del Producto

Como se comento en el punto anterior el producto cuenta con tres tipos de tareas implementadas, cabe aclarar que la cola de FreeRTOS maneja automaticamente la sincronizacion :

- Productor: Escribe en la cola imprime su numero de productor , la cantidad de espacios ocupados/libres en el momento de la escritura y su watermark para ver las palabras sin utilizar del stack de esta tarea.

- Consumidor: Lee de la cola, imprime de que productor era el recurso leido, imprime los espacios ocupados/libres y su watermark para ver las palabras sin utilizar del stack de esta tarea.
- Top: Imprime estadisticas sobre las tareas presentes en el sistema(nombre,estado,watermark,prioridad y numero) y el watermark de esta tarea, esta tarea esta activada por defecto siendo la que tiene el periodo mas largo de las 3, se puede desactivar usando el flag TOP_TASK.

Se aclara tambien que para facilitar la lectura de la salida se agrego un delay en todas las tareas para que no corran constantemente al no depender estas de estimulos externos.

Características de los Usuarios

Se asume que el sistema estaria funcionando automaticamente los usuarios solo deberian leer la salida del mismo.

Suposiciones y Dependencias

Se asume que el sistema se correra en alguna plataforma que cumpla los requisitos mencionados anteriormente en “Ambito del sistema” , ademas se debe tener el comilador para arm arm-none-eabi-gcc, qemu-system-arm y gdb-multiarch o similar en caso de querer hacer debbuging.

Las librerias necesarias de FreeRTOS vienen incluidas en el proyecto por lo cual no hay necesidad de que se encuentren en el sistema destino.

Requisitos Futuros

En un futuro podria agregarse algun tipo de interfaz externa que por ejemplo genere el funcionamiento de las tareas productores para simular mejor el funcionamiento de un sistema real.

Implementación y Resultados

- La implementacion de vTaskProductor y vTaskConsumidor se llevo a cabo de una manera similar ya que sus funciones son parecidas con la salvedad de que uno lee y otro escribe de la cola. Se uso la cola provista por el ejemplo de FreeRTOS para este chip en particular. Como se menciono anteriormente se aprovecho el hecho de que las colas de FreeRTOS manejan automaticamente la sincronizacion evitando asi el uso de mutex para solucionar problemas de

concurrencia, las prioridades del productor y consumidor son 1 y 2 respectivamente.

- Para VtaskTop se hizo uso de la utilidad de FreeRTOS VtaskList la cual genera estadisticas del sistema y lo pasa a formato legible por humanos, esta tarea hace un llamado interno a uxTaskGetSystemState() y lo formatea, para hacer su uso hubo que configurar dentro del proyecto el uso de heap4 para poder liberar memoria alocada y ademas se levantaron las banderas correspondientes en el archivo de configuracion. A esta salida se le agrego ademas encabezado y separadores para mas facil lectura por parte del usuario,la prioridad del top es de 3 para evitar que otra tarea la expulse mientras se imprimen las estadisticas.

- Para la salida por puerto serie de todas las tareas se uso la funcion uartPrint la cual recibe una cadena, el largo de esta y la transmite carácter por carácter.

- Para la impresion del watermark se uso la funcion uxTaskGetStackHighWaterMark que devuelve la cantidad de palabras disponibles del stack de la tarea dada.

Delay utilizado entre ejecuciones de cada tarea

Cada tarea tiene un cierto delay entre cada ejecucion que se puso basicamente para facilitar la lectura del usuario siendo en cada tarea:

- Productores:Cada productor corre aproximadamente cada 2500 ms
- Consumidor:el consumidor corre cada 1000ms
- TOP: corre cada 10000ms

Calculo del stack necesario para las tareas

Para la tarea Productor se estima al rededor de una ocupacion de el watermark arroja en promedio 6 palabras por lo tanto se han usado 64 en total lo que se traduce

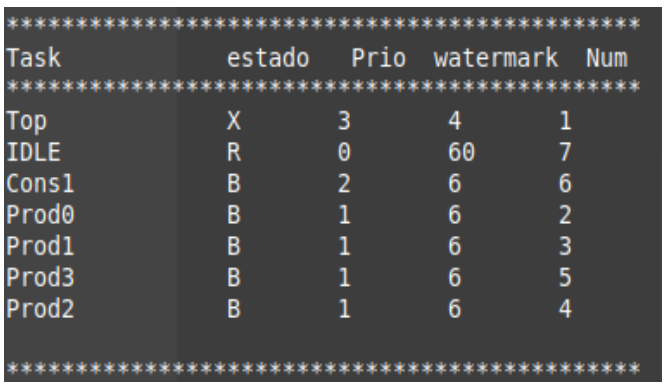
en 252 bytes usados, en un principio arroja un numero mucho mayor pero se asume que se debe a la disponibilidad de espacios disponibles en la cola, el caso del consumidor es analogo por lo tanto al ser el minimo asignable 70 se dejo ese para tener un sobrante en el stack para evitar casos de overflow.

Para el caso de Top se uso un stack de un poco mas que el doble del ultimo ya que según la documentacion vTaskList necesita un buffer de 40 bytes por tarea y eso ya representa casi la totalidad del stack minimo por lo tanto se agrego el necesario para eso con un sobrante en promedio de 4 palabras nuevamente para evitar el overflow.

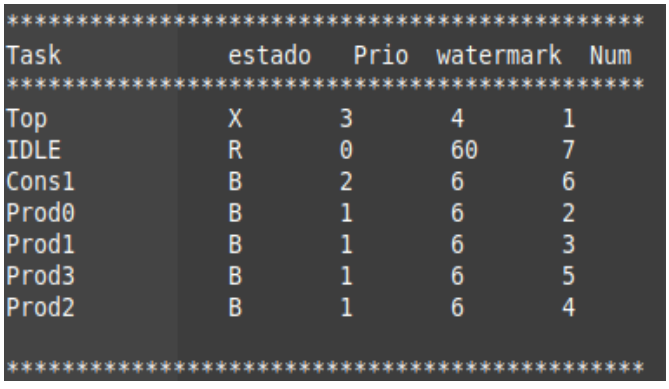
Se comprobo que con estos tamaños de stack las tareas desarrollaron su funcion normalmente en todos los casos de prueba ademas se comprobo que yendo debajo de estos stacks minimos necesarios se producia efectivamente un overflow que generaba un colapso del sistema.

Funcionamiento

El funcionamiento con las condiciones planteadas anterior mente demostro que en general al correr el top todas las tareas estan bloqueadas menos la idle que esta en estado ready y a la top que esta en estado executing, esto se debe en a los delays de cada tarea que la bloquean hasta su proximo periodo de ejecucion, si sacamos esos delays se puede ver que en generalestan todas las tareas listas a menos que las mismas se bloqueen por esperar a poder hacer una operación sobre las cola. La primer tarea a ejecutarse es la del consumidor ya que es la de mayor prioridad despues del top, y asi van bajando en orden de prioridad siendo la ultima la idle.



Task	estado	Prio	watermark	Num
Top	X	3	4	1
IDLE	R	0	60	7
Cons1	B	2	6	6
Prod0	B	1	6	2
Prod1	B	1	6	3
Prod3	B	1	6	5
Prod2	B	1	6	4



Task	estado	Prio	watermark	Num
Top	X	3	4	1
IDLE	R	0	60	7
Cons1	B	2	6	6
Prod0	B	1	6	2
Prod1	B	1	6	3
Prod3	B	1	6	5
Prod2	B	1	6	4

Conclusiones

A travez de la realizacion del trabajo practico se notaron complicaciones a las que no se esta acostumbrado dado que muchas veces eran propias de limitaciones de hardware propio del dispositivo fisico donde funcionaria este programa, por ejemplo la falta de algunas librerias estandar a las que se esta acostumbrado ya que el uso en general esta mas orientado a aplicaciones para sistemas mucho mas grandes y el reemplazo de estas por otras mas livianas provistas por esta librería a la que se noto extremadamente completa y bien documenta, mucho mas de lo previsto inicialmente.

Se comprobo que el manejo de tareas por parte del Scheduler se dio de la manera esperada como asi tambien el manejo de prioridades ya que se comprobo que siempre en todos los casos se ejecuto primero la de mayor prioridad en estado listo y se pudo tambien ver como afecta el hecho de no contar con un stack suficiente a la ejecucion y la importancia de la correcta utilizacion de recursos en un sistema de estas caracteristicas.