PRÁCTICA 3: DISEÑO DE UNA PLATAFORMA INDEPENDIENTE

INTRODUCCIÓN

Con esta práctica pretendemos que completéis el trabajo realizado en las dos anteriores, de modo que al final os acerquéis al máximo a un juego completo autónomo y funcionando correctamente en el procesador LPC2105.

OBJETIVOS:

El objetivo principal de esta práctica es finalizar el proyecto en el que habéis estado trabajando para conseguir un sistema empotrado. Particularmente, nos fijaremos los siguientes objetivos de aprendizaje:

- Diseñar e implementar una gestión completa y robusta de periféricos
- Componer una arquitectura de sistema que aproveche los modos del procesador
- Implementar y usar llamadas al sistema para interactuar con un dispositivo, un temporizador, y para activar y desactivar las interrupciones.
- Concebir e implementar de un protocolo de comunicación robusto para el puerto serie
- Identificar condiciones de carrera y su causa, así como solucionarlas

REQUISITOS FUNCIONALES

El juego deberá iniciar y terminar de manera ordenada.

Se minimizará el consumo de la placa entrando a power-down al terminar una partida. No habrá en el juego ninguna espera activa salvo las condiciones de overflow.

Se gestionarán como llamadas al sistema la activación y desactivación de las rutinas de servicio de periféricos.

La entrada de la jugada se realizará a través de línea serie. El programa enviará al usuario el tablero para visualizar los movimientos por pantalla.

Se dará opción de confirmar y cancelar mediante los botones.

Al final de la partida, se enviará el tiempo de juego y el tiempo total de cálculo a través de la línea serie.

Se puede depurar con nivel de optimización O0, pero la versión final debe funcionar correctamente en modo *release* con O3.

PASOS A SEGUIR

La práctica consta de los siguientes apartados obligatorios:

A. UTILIZACIÓN DEL REAL TIME CLOCK Y WATCHDOG

Nuestro procesador dispone de dos contadores con funcionalidades específicas: el Real-time clock (RTC)y el watchdog (WD). En el diseño final vamos a utilizarlos de la siguiente forma:

- Configuraremos el RTC para que mida el tiempo transcurrido durante la partida. Para ello debéis leer el manual y ajustar convenientemente su reloj interno. No necesitamos que genere ninguna interrupción, tan sólo que podamos leer el tiempo transcurrido. Por tanto, bastará con diseñar las siguientes funciones:
    - RTC_init(): inicializa el RTC, reseteando la cuenta, ajustando el reloj y activando el enable. Para ello trabaja con los registros CCR (ver sección 5.5 del manual), PREINT y PREFRAC (ver 5.7).
    - RTC_leer_minutos(): devuelve los minutos de juego (entre 0 y 59).
    - RTC_leer_segundos(): devuelve los segundos transcurridos en el minuto actual (entre 0 y 59).
- El watchdog es un contador que permite resetear el procesador. Es muy útil para resetear un programa que se haya quedado colgado. Se puede utilizar también como un contador convencional, pero lo que nos interesa es aprender a utilizarlo para que genere un reset. Funciona de la siguiente forma. En primer lugar, se le asigna un tiempo de cuenta con el registro WDTC y se activa el bit de reset y el bit de enable en el registro WDMOD. A continuación, hay que "alimentarlo" realizando dos escrituras **consecutivas** en WDFEED: WDFEED = 0xAA; WDFEED = 0x55 (cuidado, si las escrituras no son consecutivas, por ejemplo, si hay una interrupción entre ambas, se genera una excepción). Cuando el WD se alimenta por primera vez comienza su cuenta. Si la termina, y el bit de enable y de reset del WDMOD están activos, el WD resetea la ejecución. Si no queremos que el sistema se resetee se debe volver a alimentar antes de que termine. En tal caso, el WD volverá a empezar la cuenta desde el principio. Diseñad las siguientes funciones:
    - WD_init(int sec): inicializa el watchdog timer para que resetee el procesador dentro de *sec* segundos si no se le "alimenta".
    - WD_feed(): alimenta al watchdog timer.

B. IMPLEMENTACIÓN DE LLAMADAS AL SISTEMA

Tal y como habéis aprendido en la asignatura de sistemas operativos, el núcleo de un sistema operativo (SO), se ejecuta en un modo de ejecución diferente al de las aplicaciones. Al primer modo se le suele denominar supervisor y al segundo usuario. Una de las ventajas de estos modos es que permiten restringir ciertas operaciones para que sean realizadas únicamente por el núcleo del SO y hacer los sistemas más seguros.

En la arquitectura ARM, cuando las aplicaciones requieren de servicios del SO, emplean la instrucción `SWI` #número_servicio para requerirlos. Por ejemplo, leer un fichero puede corresponderse con el número 4 tal y como sucede en algunos sistemas Linux, (https://chromium.googlesource.com/native_client/nacl-newlib/+/master/libgloss/arm/linux-syscall.h). Al ejecutarse, SWI causa una interrupción software. Es decir, el modo de ejecución cambia a supervisor, el registro CPSR se almacena en el registro SPSR del modo supervisor y se ejecuta el código de la entrada correspondiente en el vector de SWI.

En nuestro diseño actual el procesador siempre está en modo supervisor y a partir de este momento, sólo tendrá que trabajar en dicho modo al realizar tareas del sistema como la interacción con periféricos. En concreto, se implementarán 5 llamadas al sistema: clock_gettime para leer marcas temporales, enable_irq/disable_irq para activar/desactivar las interrupciones y enable_irq_fiq/disable_irq_fiq para activar/desactivar interrupciones y fiq.

Para implementar estas 3 las llamadas al sistema realizaremos los siguientes pasos:

- En el fichero de arranque, `Startup.s`, **debemos comprobar que cambia a modo usuario** antes de invocar a la función `__main`
- Para acceder al timer1 vía una interrupción, definiremos su número de interrupción, por ejemplo, el 0, y añadiremos el siguiente código en el módulo del timer:

```
uint32_t __swi(0) clock_gettime(void);
uint32_t __SWI_0 (void) {
   …
}
```

  Es importante revisar en el binario generado el código de esta función para entender cómo se produce el cambio de contexto.

- A partir de este momento, las marcas temporales se leerán llamando a la función clock_gettime(), la función timer1_leer() no podrá ser invocada desde espacio de usuario.
- Estudiaremos el fichero SWI.s suministrado para ver cómo se accede al vector de interrupciones.
- Buscaremos en Startup.s el vector de excepciones y cambiaremos el código para que invoque a nuestro manejador de interrupciones
- Para activar/desactivar excepciones y fiq, asumiremos que no nos caben en el vector y saltaremos a ellas desde la propia rutina de servicio de SWI. Además, su bloque de código retornará a modo usuario. Activar/Desactivar las IRQs e IRQs y FIQs corresponderán a las llamadas al sistema 0xFF, 0xFE, 0xFD y 0xFC, respectivamente.

```
void __swi(0xFF) enable_isr (void);
void __swi(0xFE) disable_isr (void);

void __swi(0xFD) enable_isr_fiq (void);
void __swi(0xFC) disable_isr_fiq (void);
```

- Dentro del código para activar/desactivar las interrupciones y fiq, que deberá estar dentro del fichero SWI.s, hay que cargar el registro SPSR, modificar el bit de IRQ, actualizar SPSR, y finalmente volver a modo usuario. Para cargar el registro SPSR, deberéis de revisar la documentación de la instrucción SWI y la copia en memoria del registro CPSR al cambiar de modo.

C. ELIMINACIÓN DE CONDICIONES DE CARRERA

Una **condición de carrera** es un problema que ocurre cuando se accede de manera concurrente a un recurso compartido y el resultado final de las operaciones depende de la temporización. Es decir, el resultado de un programa puede depender de cómo ocurran las operaciones desde por ejemplo una rutina de servicio y el hilo del programa principal.

En nuestro caso concreto la cola de eventos pude sufrir condiciones de carrera si mientras la estamos leyendo desde el programa principal, nos interrumpen desde una rutina de servicio a interrupción para añadir un nuevo elemento, y finalmente volvemos a actualizar los índices al completar la operación de lectura.

Para solucionar el problema deberemos asegurar que la ventana de error es lo más pequeña posible mediante la separación de la lectura de la posición a leer en una variable temporal y su posterior actualización. Además, deberán emplearse estructuras para no tener que manejar varios índices por posición. Una vez minimizada la ventana de error, deberemos emplear las nuevas llamadas al sistema de desactivar/activar interrupciones para asegurarnos la actualización atómica de los índices de la cola al leerla y así solucionar la condición de carrera.

Además, tal y como describe el manual del procesador en la sección 4.3, el watchdog requiere no ser interrumpido. Por tanto, **antes de alimentar el watchdog hay que deshabilitar todas las interrupciones** y habilitarlas inmediatamente después de la alimentación.

D. INTRODUCCIÓN DE JUGADA POR LÍNEA SERIE

Vamos a permitir jugar remotamente. Los comandos recibidos nos llegaran por la línea serie (UART0). Así mismo el tablero y los resultados de tiempo se enviarán por la línea serie.

La línea serie envía carácter tras carácter en un protocolo asíncrono, nos interesa poder encapsular los distintos mensajes en comandos. Para ello se formarán tramas con delimitador de inicio (#), carga y delimitador de final (!) para poder determinar dónde empieza y dónde acaba cada mensaje.

Los comandos aceptables codificados en ASCII son:

- Acabar la partida: `#RST!`
- Nueva partida: `#NEW!`
- Jugada: `#FCV`***`S`***`!`

En este último caso, para poder detectar errores, se añade a la información de Fila, Columna y Valor información redundante mediante una suma de chequeo (***S***). Este checksum se calcula como la suma de la fila, la columna y valor módulo 8. Por ejemplo, la jugada fila 5 – columna 6 – valor 4, llevará como checksum el carácter '7' (5 + 6 +4 = 15, 15 % 8 = 7). El comando quedaría `#5647!`

Al igual que el resto de los periféricos, a la línea serie accederemos por interrupciones y al llegar un comando completo se generará un nuevo evento.

Cuando la RSI detecte un carácter generará el evento correspondiente. El gestor de entrada salida lo procesará y lo almacenará en un buffer. El tamaño máximo del buffer es de 10 caracteres y se debe controlar su tamaño para no sufrir un desbordamiento de buffer, (*buffer overrun* o *buffer overflow*), que corrompería la pila o la zona de datos y supondría un fallo de seguridad importante.

Cuando el gestor detecte un comando valido generará el evento correspondiente. En el caso del evento *Jugada,* la información de fila, columna y valor se deberá almacenar en el campo auxiliar del evento (podéis elegir el formato).

También se enviará por la línea serie el tablero cada vez que se actualice para poder ser visualizada en pantalla.

Debéis ser capaces de enviar el tablero codificando las celdas con caracteres ASCII fácilmente entendibles (podéis elegir los que queráis para mostrar celdas, fronteras, pistas y candidatos) y mostrarlo por filas. Por ejemplo:

```
+-+-+-+-+-+-+-+-+-+
| .1. |2. .4| .2. |
+-+~+-+^+-+-+-+^+-+
|3. . |7.6. | .5. |
+~+-+-+-+-+-+-+-+-+
…
```

Siendo por ejemplo el “1” una pista y los “2” errores. Un tablero al lateral podría contener los candidatos (revisar la tabla ASCII válida por UART).

Otro diseño alternativo podría ser tener una cuadrícula donde cada cuadrado corresponda a dos caracteres, de modo que aparece el número introducido y al lado una letra que indica si es pista o hay error y un listado que muestre los candidatos del tablero con Fila Columna y todos los candidatos posibles (de aquellas casillas que no son pista, podéis poner los dígitos de candidatos o codificarlos de algún modo.). Se adjunta la siguiente figura ejemplo

```
+ - - + - - + - - +
| 3P  |     |     |
+ - - + - - + - - +
|  3E |     |  1P |
+ - - + - - + - - +
|     |     |     |
+ - - + - - + - - +
```

FC candidatos

22 25

El envío por línea serie se puede hacer carácter a carácter, pero eso la hace muy ineficiente. Realizar una función que permita enviar un array de char (tipo C) mediante interrupciones sin esperas activas.

E. TRATAMIENTO DE TIMER0_RSI COMO FAST INTERRUPT

Empezamos a tener muchas fuentes distintas de interrupción. Además, el tratamiento de la línea serie puede alargar un poco más de lo que estamos acostumbrados el tiempo con las IRQs desactivadas.

Si un evento de un periférico solicita una interrupción y antes de poder atenderlo llegase otro evento del mismo, nos encontraríamos con que al atender al periférico habríamos perdido uno de ellos al no poder distinguirlos. En nuestro sistema, el Timer0 es probablemente el más crítico. Por ello nos solicitan pasar la rutina de servicio del timer0 a interrupción rápida (FIQ).

Deberéis entender el fichero Startup.s, en especial la creación de la tabla de vectores de excepción global del sistema. Modificad la entrada de la tabla de vectores correspondiente al manejador de la FIQ para que al enlazar el linker encuentre la RSi del timer0 de vuestro código en C. **Importante: ajustad la definición de la pila de FIQ**, que ahora mismo tiene 0B de tamaño.

Acordaros de que para forzar el alineamiento de la pila es muy posible que tengáis que añadir la directiva `PRESERVE8 {TRUE}`.

F. MAQUINA DE ESTADOS DEL JUEGO COMPLETO EN SU IMPLEMENTACIÓN FINAL

Es preciso que diseñéis la máquina de estados completa que cumpla todos los requisitos anteriores.

- Al inicio de la partida, el programa mostrará por línea serie información sobre el juego y de cómo jugar mientras se espera la pulsación de cualquier botón o llega un nuevo comando de nueva partida.
- A continuación, aparecerá el **tablero** por línea serie en la consola, el formato lo dejamos libre, pero deberéis intentar que se vea de la mejor forma posible para facilitar el jugar. Se debería de alguna forma saber que son pistas y candidatos.
- Vamos a incluir la llegada de comandos por línea serie junto al aceptor o cancelar por botones. La fila, columna y valor se recibirán por línea serie. Se mostrará el tablero con el indicador especial de la jugada y se dispondrá de 3 segundos para aceptar o cancelar con los botones. Durante estos tres segundos parpadeará el pin GPIO de la práctica 2.
- Al procesar el movimiento, se enviará de nuevo el tablero por la línea serie.
- El usuario irá introduciendo valores (fila, columna, dato). Tras cada inserción el sistema recalculará las listas de candidatos y comprobará la existencia de errores. Si hay algún error se mostrarán claramente todas las celdas involucradas en ese error. Se deben resaltar todas las celdas involucradas, incluidas las pistas, para facilitar que el usuario descubra la celda errónea. En todo caso se debe poder seguir distinguiendo pistas de valores de usuario.
- Al terminar la partida, se mostrará la siguiente información; razón por la que se termina, tiempo total de la partida con el RTC y tiempo total de computo de la función **`candidatos_actualizar`** con Timer1.

- Además, se mostrará la leyenda de volver a jugar. Si el usuario decide volver a jugar se empezará una nueva partida.
- En el modo reléase, si se produce un overflow, transcurridos más de 20 segundos en la espera activa se resetea el juego, para ello se utilizará el watchdog.
- Se debe mantener el esquema de la cola de eventos de la práctica 2.

APARTADOS OPCIONALES:

- Toolboxes conectadas al GPIO que emulen botones
- Reevaluar el rendimiento mirando ahora las funciones más costosas.

MATERIAL DISPONIBLE:

En la página de la asignatura en Moodle podéis encontrar códigos de ejemplo. Os recordamos, podrán aparecer indicaciones a lo largo de la práctica, se recomienda consultar Moodle con frecuencia y en particular la información que vaya apareciendo en la wiki.

EVALUACIÓN DE LA PRÁCTICA

Está práctica se presentará el miércoles 22 de diciembre. La memoria el viernes 14 de enero.

La evaluación de esta práctica y la presentación podrá ser individual (la memoria es única y debe incluir la práctica 2 y la práctica3).

PRESENTACIÓN 22 DE DICIEMBRE

Debéis entregar previa a la presentación el proyecto completo comprimido en .zip a través de Moodle. En un fichero comprimido Correccion_P3_NIP-Apellidos_Alumno1_NIP-Apellidos_Alumno2.zip. Los detalles de los horarios de corrección se publicarán en Moodle.

ENTREGA 14 DE ENERO

Debéis entregar dos ficheros, la memora y el proyecto.

La memoria en formato pdf El archivo que contiene la memoria debe llamarse

p2-3_memoria_NIP-Apellidos_Alumno1_NIP-Apellidos_Alumno2.pdf

El proyecto keil incluiría todos los ficheros y se entregará comprimido en un fichero zip. El fichero debe tener el siguiente nombre

p2-3_NIP-Apellidos_Alumno1_NIP-Apellidos_Alumno2.zip

Por ejemplo: p2-3_memoria_345456-Gracia_Esteban_45632-Arribas_Murillo.pdf y p2-3_345456-Gracia_Esteban_45632-Arribas_Murillo.zip

ANEXO 1: REALIZACIÓN DE LA MEMORIA FINAL

Esta memoria incluirá el trabajo realizado en las prácticas 2 y 3. La memoria de las prácticas tiene diseño libre, pero debe incluir los apartados propios de una memoria técnica (recomendable revisar el documento Redacción de una Memoria Técnica, disponible en la web de la asignatura). Es obligatorio que incluya al menos los siguientes contenidos (no es la estructura que debe seguir la memoria):

1. Resumen ejecutivo (una cara como máximo). El resumen ejecutivo es una página independiente al inicio de la memoria que describe brevemente qué habéis hecho, por qué lo habéis hecho, qué resultados obtenéis y cuáles son vuestras conclusiones.
2. Código fuente comentado (sólo el que habéis desarrollado nuevo y sea **relevante** de las prácticas 2 y 3). Como siempre, cada función debe incluir una cabecera en la que se explique qué hace, qué parámetros recibe..., ese apartado, valorad su inclusión como anexo.
3. Descripción de los problemas encontrados en la realización de la práctica y sus soluciones.
4. Resultado final.
5. Conclusiones valorando vuestro proyecto final.
6. Las correcciones solicitadas, en su caso, en las entregas previas.

Se valorará que el texto sea **claro y conciso, se recomienda una extensión menor de 15 páginas,** sin contar los anexos, con el correspondiente rigor técnico. Incluir en la memoria las partes relevantes de vuestro código que habéis desarrollado para cada sección. Cuánto más fácil sea entender el funcionamiento del código y vuestro trabajo, mejor.