Práctica 7

Entorno de programación Visual C#.

Sistemas en Tiempo Real

Material: PC y Visual Studio 2013

Duración: 2 horas

Lugar: Laboratorios de prácticas (Laboratorio de Redes-Hardware)

La herramienta de desarrollo que utilizaremos para el desarrollo de las prácticas de la asignatura será el entorno de programación de Visual C, Microsoft Visual Studio 2013, el cual permite la creación, compilación y ejecución de programas escritos en lenguaje Visual C#; tanto en modo consola como en modo gráfico. Se recuerda al alumnado que, para la mejor comprensión y asimilación de los conocimientos y habilidades desarrollados durante la práctica, deberá haberse estudiado previamente el material docente disponible.

Introducción:

Como hemos visto durante el trascurso de las clases de teoría, un sistema en tiempo real es aquel que, no solamente responde a los eventos de su entorno, sino que además dicha respuesta se produce en el tiempo adecuado. Existen sistemas en los que el tiempo de respuesta resulta crítico y no pueden utilizarse acciones programadas temporizadas, debido a que éstas podrían ejecutarse tarde y tener consecuencias catastróficas. Por ejemplo, la centralita de control de un vehículo debe responder a los diferentes eventos que se produzcan en un tiempo adecuado. Por ello, en los sistemas de control en tiempo real, el uso de acciones programadas (mediante el empleo de *timers*) puede resultar insuficiente, por un lado, o bloquear el sistema, por otro; siendo necesaria una respuesta al evento de manera inmediata y transparente. Estableciendo una diferencia con lo visto hasta ahora en la asignatura, los *timers* nos permiten control y supervisión de forma síncrona dentro de la ejecución del programa principal: cada cierto tiempo se ejecutan las acciones programadas; mientras que, la

ejecución mediante los denominados *workers* o *hilos* nos permitirá controlar y supervisar la ejecución de forma asíncrona, es decir, de manera continua y en *background*. Dependiendo del sistema hardware en el que se ejecute nuestro programa, la ejecución se realizará sobre un solo procesador o sobre más de uno. **Nota**. Repasar los conceptos vistos en teoría.

Desarrollo de la práctica:

1. Desarrollo de aplicación de control y supervisión con timers.

En esta primera parte de la práctica, procederemos a resolver el ejercicio que a continuación se indica. Dicho ejercicio será utilizado en la segunda parte de esta práctica, para profundizar en el funcionamiento de los hilos o *workers* de ejecución asíncrona y en background, completando lo visto en teoría.

1) Descarga el proyecto práctica 7 del espacio virtual de la asignatura. La interfaz ya se encuentra desarrollada, el objetivo de esta práctica es centrarnos en la programación en tiempo real. Solamente, completaremos los conocimientos de programación visual vistos hasta ahora con tres aportes nuevos:

1. La aplicación emplea un componente de usuario denominado *temperatura_led.xaml*, dicho componente ya se encuentra agregado al proyecto, por lo que no es necesario realizar ninguna acción sobre él. Además, se trata de un componente dinámico programado; pues cuenta con un archivo de C# en el que se define su comportamiento (*temperature_led.xaml.cs*). Puede consultarse el contenido de dicho fichero, aunque no se implementará ningún componente de este tipo en la asignatura. Lo que sí es obligatorio conocer es cómo se emplean (ver siguiente punto).

2. El acceso a los componentes de usuario dinámicos, programados mediante C#, se realiza de forma idéntica al resto de componentes de usuario:

```
//modificación de los sensores al pulsar sobre el botón de test
//función private void test_Click(object sender, RoutedEventArgs e)
sensor_1.value.Content = "100";
sensor_2.value.Content = "70";
sensor_3.value.Content = "50";
```

3. Cuando se quiere acceder a un componente de usuario en concreto mediante C#, por ejemplo el sensor_12, podemos emplear la forma vista en el punto 2. Pero si no sabemos a priori qué sensor se debe modificar, porque no se conoce el sensor que se debe actualizar hasta el momento en el que el programa se ejecuta, tendremos que utilizar el siguiente código:

```
//modificación de los sensores al pulsar sobre el botón de test
private void actualizar_sensor(int sensor, int dato)
{
   //actualizamos el sensor correspondiente
   temperature_led led = (temperature_led)this.FindName("sensor_" + sensor.ToString());
   led.value.Content = dato.ToString();
}

//invocación de la función actualizar_sensor()
actualizar_sensor(12,100);
```

Como puede verse, se emplea la instrucción *this.FindName()* para buscar un componente en concreto de entre los componentes disponibles, realmente se busca el identificador del componente que se le especifica como parámetro, en este caso, un componente que tenga como identificador *"sensor_"+sensor.ToString()*, asignándose dicho valor a la variable *led*. Posteriormente, la variable *led*, es la que se utiliza para acceder al componente deseado. Puedes depurar el programa y, con el inspector de variables, comprobar el funcionamiento de la instrucción. El que se haya incluido la instrucción comentada en una función es un ejemplo de uso.

Para realizar la práctica 7, se empleará una función denominada *actualizar_posicion()*, cuyo código se encuentra implementado en el proyecto suministrado. Su ejecución responde a lo que se detalla en este punto, además de lo que se especificará, en relación a la práctica que se desarrollará.

2) Abre el proyecto descargado con el entorno de programación Visual Studio (Figura 1).

3) Ejecuta el proyecto, deberás ver la interfaz que se muestra en la Figura 2, aunque no realiza ninguna operación, pues debemos completar el código para realizar la práctica. Revisa el código de la misma y, muy especialmente, revisa el código de la función *actualizar_posicion()*.

4) En el proyecto descargado, puedes encontrar el archivo ejecutable, que puedes emplear para comprobar el funcionamiento del programa.

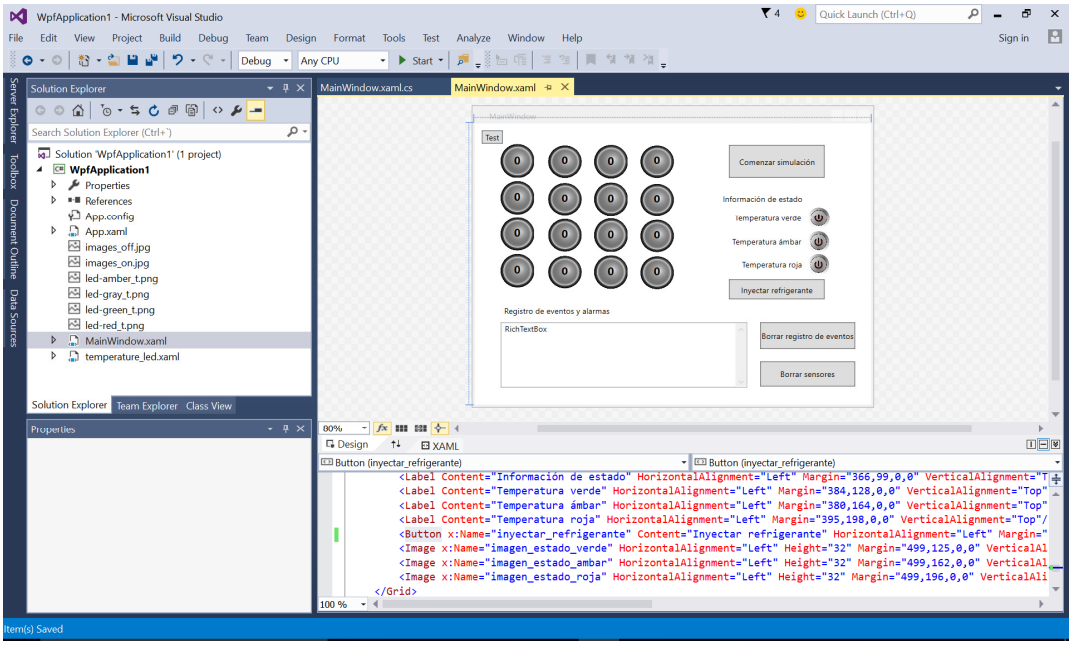


Figura 1. Visión general del proyecto

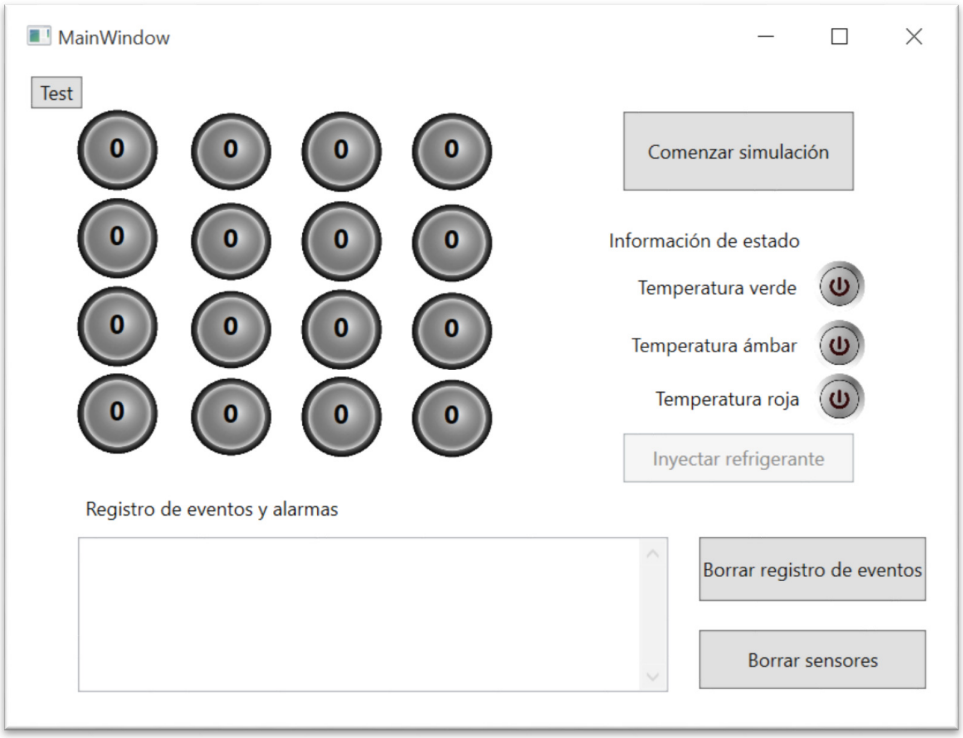


Figura 2. Ejecución del proyecto

5) A continuación, se detalla el enunciado del ejercicio a realizar en esta primera parte de la práctica.

Ejercicio:

Realizar un programa en Visual C# (WPF) que, dada una matriz de enteros 4x4 declarada como una variable global, simulará una matriz de sensores de temperatura de un sistema de control, monitorización y refrigeración de motores, operando sobre la misma una serie de *timers*, atendiendo a las directrices que se indicarán a continuación. Los *timers* deberán ejecutarse según lo que se especificará más adelante para cada uno de ellos. El proyecto a desarrollar simulará un sistema remoto de control y supervisión para la refrigeración de 16 motores, así como las acciones de control que se realizarán sobre éste.

El interfaz contará con cuatro botones, los cuales serán los encargados de:

- Botón "Borrar registro". Borra toda la información contenida en el *richTextBox*, componente que muestra la información de eventos y alarmas.
- Botón "Sensores a cero". Es la puesta a cero o reinicio de la matriz de temperaturas (variable global), así como la puesta a cero de la representación visual de dicha información (imágenes de los sensores de temperatura), y de la puesta a off (off.jpg) de las imágenes que muestran la información de estado (ver *timer 2* para más información). Este botón no detiene la simulación.
- Botón "Activar simulación": activa y desactiva la simulación, no se borra nada, pero sí que se bloqueará el botón de "Inyectar refrigerante" (ver botón de "Inyectar refrigerante". Se mostrarán mensajes en el registro de eventos y alarmas al efecto, indicando la puesta en marcha y la parada del sistema.
- Botón "Inyectar refrigerante": se detalla en los siguientes apartados del ejercicio.

El programa contará con dos *timers* temporizados, que deberán ser implementados atendiendo a:

- Timer 1. Este *timer* se encargará de actualizar, cada tres segundos, una posición aleatoria de la matriz de temperaturas, teniendo en cuenta las siguientes reglas:
    - Si la posición contiene el valor cero se generará un valor de 0 a 100 (ambos incluidos), representado un valor de temperatura expresada en grados.
    - Si la posición contiene un valor menor que 50, se incrementará el valor almacenado en 20 grados.

- Si la posición contiene un valor mayor o igual que 50 y menor que 70, se incrementará el valor almacenado en 10 grados.
- Si la posición contiene un valor mayor o igual que 70 y menor que 90, se incrementará el valor almacenado en 5 grados.
- Si la posición contiene un valor mayor que 90, se incrementará el valor almacenado en 1 grado.
- Se mostrarán mensajes en el registro de eventos y alarmas, para indicar el sensor que ha sido actualizado y con qué valor. Además, el mensaje será negro (*Brushes.Black*), verde (*Brushes.Green*), ámbar (*Brushes.Orange*) o rojo (*Brushes.Red*), según el rango de temperatura del sensor: negro para temperaturas menores de 50 grados, verde, para temperaturas comprendidas entre los 50 y los 69 grados, ámbar, para temperaturas comprendidas entre 70 y 89 grados y, rojo, si la temperatura es igual o superior a 90 grados.

Además, será necesario ejecutar la instrucción de la función de visualización, función *actualizar_posicion()*, para representar en pantalla la evolución que se producirá en la matriz de temperaturas donde corresponda.

Nota importante. Para el correcto funcionamiento de la función *actualizar_posicion()*, los componentes de tipo *temperatura_led* tienen un nombre de identificador de 1 a 16, siendo *sensor_1* el identificador del primer elemento de la matriz de sensores (esquina superior izquierda), mientras que *sensor_16*, será el identificador del último elemento (esquina inferior derecha).

- Timer 2. Este *timer* se encargará de visualizar la "Información de estado". Su función es recorrer la matriz de temperaturas, para lo cual ejecutará su código cada medio segundo, buscando valores en la matriz de temperaturas que se encuentren comprendidos en los rangos de 50 a 69, 70 a 89 y 90. En caso afirmativo, es decir, se han encontrado temperaturas en alguno de dichos rangos, deberá:
  - Indicar visualmente que se han encontrado temperaturas dentro del rango correspondiente, cambiando la imagen que corresponda en la zona de "Información de estado" de la interfaz (imagen *on.jpg*): temperaturas en verde, en ámbar o en rojo, dependiendo de las temperaturas que se hayan encontrado. Nótese que puede indicarse la activación de más de una a la vez (de una, de dos o de las tres).

- Si no se han encontrado temperaturas en un determinado rango, deberá indicarlo gráficamente cambiando la imagen de las temperaturas correspondientes con la imagen de apagado (*off.jpg*).
- Habilita el botón ""Inyectar refrigerante".

Nota. Las imágenes *on.jpg* y *off.jpg* ya se encuentran agregadas al proyecto, y pueden ser usadas por éste, tal y como se ha visto en la asignatura.

En caso negativo:

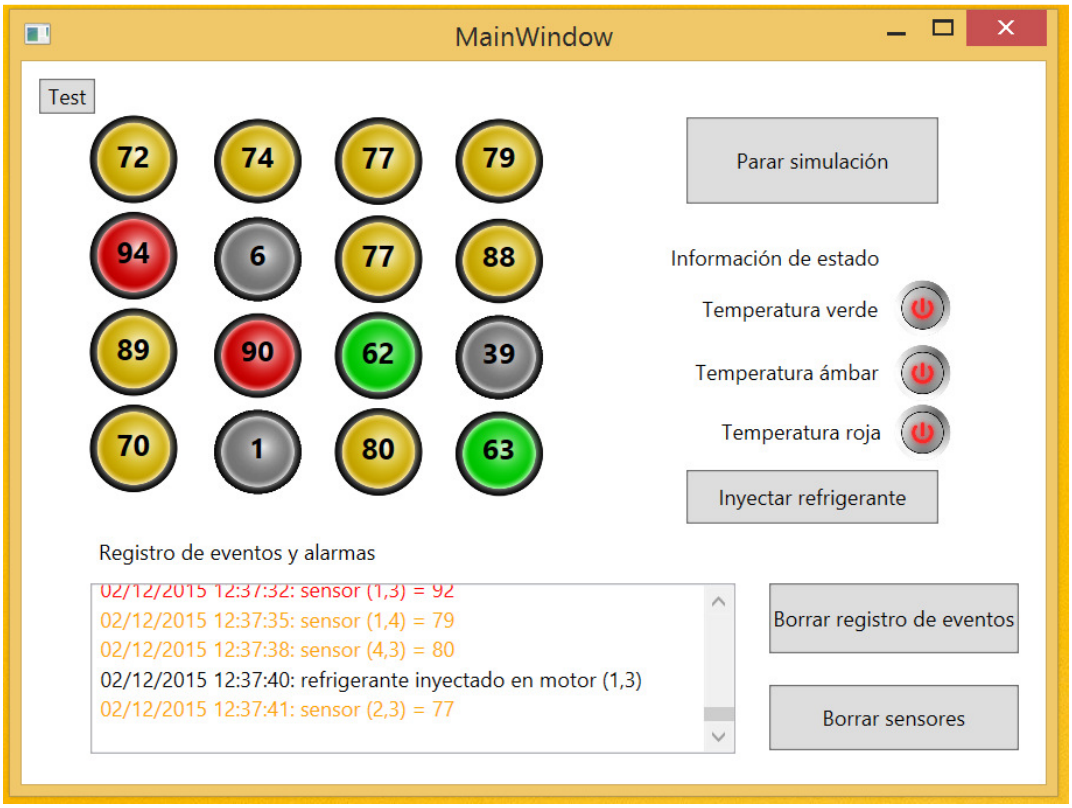
- Simplemente, si no hay ninguna temperatura dentro de los rangos establecidos para las alarmas, todas las imágenes de estado aparecerán apagadas (*off.jpg*).
- Deshabilita el botón "Inyectar refrigerante".

- Botón "Inyectar refrigerante". Este botón será el encargado de inyectar cierta cantidad refrigerante en el sistema (unidades de refrigerante), pero solamente podrá inyectar un número determinado de unidades cada vez que el usuario pulsa dicho botón. El refrigerante produce siempre una refrigeración de un grado centígrado por unidad, lo que se deberá reflejar en la matriz de temperaturas con una bajada sobre el valor almacenado proporcional al número de unidades inyectadas. El algoritmo que deberá programarse, cada vez que se pulsa el botón para realizar la inyección del refrigerante, será el siguiente:
  - Solamente se refrigera una temperatura por pulsación del botón, atendiendo a las siguientes prioridades y unidades de refrigerante inyectado.
  - Si hay una temperatura en rojo ésta será la primera que recibirá inyección de refrigerante cuando se pulse el botón y en cualquier orden, si hay más de una temperatura de este tipo. Las temperaturas en rojo son refrigeradas obligatoriamente con 15 unidades de refrigerante por pulsación.
  - Si no hay temperaturas en rojo, las temperaturas ámbar serán entonces las refrigeradas en cualquier orden, si hay más de una temperatura de este tipo. Las temperaturas en ámbar son refrigeradas obligatoriamente con 10 unidades de refrigerante por pulsación.
  - Si no hay temperaturas en rojo ni en ámbar, las temperaturas en verde serán las refrigeradas en cualquier orden, si hay más de una temperatura de este tipo. Las temperaturas en verde son refrigeradas obligatoriamente con 5 unidades de refrigerante por pulsación.

- Se mostrarán mensajes en el registro de eventos y alarmas, indicando qué motor ha sido refrigerado; en la aplicación se mostrará una bajada de temperatura en el sensor correspondiente.
- Será necesario ejecutar la instrucción de la función de visualización *actualizar_posicion()*, con el fin de actualizar la representación gráfica.

Nota. Hay que tener en cuenta que la generación de valores en el *timer 1* se producirá cada tres segundos, mientras que, el *timer 2* se ejecutará cada 0,5 segundos; por lo que, el estado indicado por la "Información de Estado" en el interfaz, dará la sensación de ejecución instantánea.

Ejemplo de la interfaz:



2. *Desarrollo de aplicación de control y supervisión con workers.*

En esta segunda parte de la práctica, procederemos a resolver el ejercicio desarrollado en la primera parte de la misma, profundizando en el funcionamiento de los hilos o *workers* de

ejecución asíncrona y en *background*, completando lo visto en teoría y justificando la importancia de los mismos.

1) Modifica el programa anterior para que, el *timer 2* sea implementado mediante un *worker* en *background*. Según lo visto en clase debes recordar que:

1. Los *workers* no tienen acceso a la interfaz, debes utilizar una de sus funciones específicas para ello.
2. El *worker* a programar se ejecutará de forma continua durante el programa y, deberás tener en cuenta lo visto en clase para que dicho *worker* se ejecute correctamente (pausa de un milisegundo), con el fin de que no interfiera con la interfaz.

2) Añade un proceso en *background* al programa que has desarrollado hasta el momento para esta segunda parte de la práctica. Dicho proceso deberá recorrer de forma indefinidamente la matriz de temperaturas y si encuentra una temperatura mayor o igual a noventa grados deberá fijarla a cero; no se modificará el interfaz gráfico. **Nota**. Debido a la velocidad de proceso, no verás temperaturas rojas, aunque sí que verás descuadres entre el color de los mensajes que se muestran en el registro de eventos y el valor que toma el led. No hay que corregir este problema.