Sistema distribuido de tiempo real.

Emilio Arenas Bosch, Bartolomé Martínez Pérez

Tercer curso de Ingeniería Técnica en Informática de Sistemas
arenas.emilio@gmail.com
martinez.tolo@gmail.com

***Resumen*— En este artículo se presenta la implementación de un sistema de básculas distribuido de tiempo real. El sistema se compone de 2 básculas, una base de datos de cotizaciones que se encarga de devolver el precio de los productos seleccionados y un supervisor de ventas. Cada uno de estos componentes es instalado sobre un microcontrolador dsPIC4011 con el soporte de un sistema operativo de tiempo real (SOTR) llamado SALVO OS. A su vez, se utiliza el protocolo de comunicaciones CAN para el envío y recepción de los mensajes entre los diferentes nodos del sistema.**

I. Introducción

Un Sistema Distribuido (SD) es un conjunto de computadores que se presenta al usuario como un único sistema coherente. Cada máquina posee sus componentes hardware y software que el usuario percibe como un solo sistema; Las características principales de un SD son:

- Para los usuarios, la percepción del trabajo debe ser similar al de un Sistema Centralizado (SC)
- Se ejecuta en múltiples computadoras
- Dependiente de redes

Los sistemas de tiempo real (STR) interactúan con un entorno de sucesos externos (estímulos de entrada) cuya dinámica es conocida. La corrección de su salida depende, no sólo de su valor, sino también del instante en que se produce. Tienen la facultad de ejecutar actividades o tareas en intervalos de tiempo bien definidos para garantizar la corrección de la salida. Los intervalos en que se ejecutan las tareas se definen por un esquema de activación y por un plazo de ejecución.

Este proyecto se basa un sistema de computadores distribuidos conectados por una red de comunicaciones CAN que cumple con unas restricciones temporales estrictas.

Los SO que garantizan las restricciones temporales son los conocidos como Sistemas Operativos de Tiempo Real (SOTR). En nuestro caso se ha usado el SOTR Salvos de Pumkin S.L., explicado más adelante con detalle.

Por otro lado hay que garantizar unos tiempos de latencia acotados y alta tolerancia a fallos en la red de comunicaciones. Por ello se usa el protocolo CAN. Este protocolo garantiza tiempos de latencia máximos y alto control de errores. Los mensajes enviados se identifican con un identificador de 11 bits, el cual indica el tipo de información que contiene la trama.

El resto del artículo se estructura como sigue. En la siguiente sección se detallan los componentes. En la sección III se encuentra el software utilizado, dividido en herramientas software y librerías. En la sección IV se dan las nociones básicas del sistema operativo Salvo para entender el desarrollo de la práctica expuesto en la sección V.

II. Hardware

La red está formada por un solo bus de comunicaciones serie con cuatro conectores DB9. Las placas utilizadas en el proyecto se dividen en tres componentes, cada uno integrado en el siguiente. Su composición se explica en los subapartados siguientes, desde el de más alto nivel al más bajo.

A. *Microcontrolador dsPIC30F4011*

Este microcontrolador forma parte de la familia dsPIC30F de Microchip Technology Inc. Entre los dispositivos de E/S usados están: el módulo CAN, para la comunicación con los demás MCU, el módulo UART, usado para comunicaciones serie (bit a bit) con un ordenador, dos entradas analógicas, además de puertos de uso genérico.

B. *Módulo iCM4011 (ingenia Communications Module 4011)*

Básicamente este módulo está formado por el procesador anterior junto con tranceivers para las comunicaciones que ofrece el dsPIC30F4011. También incorpora una interfaz USB con la que reprogramar la memoria principal del sistema.

C. *UIB-PC104 (User Interface Board)*

Es una placa que permite al módulo iCM4011 actuar sobre entradas y salidas, además de otros componentes electrónicos como alimentación y un conector DB9 para comunicaciones serie. Las entradas y salidas utilizadas son un LCD alfanumérico (pantalla delgada de pixeles monocromo), doce pulsadores (teclado pequeño) y dos potenciómetros (resistencia de valor variable manualmente).

III. Software

El software se puede dividir en: herramientas software y programa. El segundo, a su vez, está formado por librerías y el programa propiamente desarrollado, explicado en la sección IV.

A. Herramientas software

El programa esta escrito en lenguaje C con el programa de desarrollo MPLAB de Microchip. Este software proporciona herramientas para el desarrollo de proyectos en ensamblador o lenguaje C para gran cantidad de PIC's. El código generado por el MPLAB se introduce en los dsPIC con el programa dsPIC bootloader de ingenia-cat, S.L.

B. Librerías

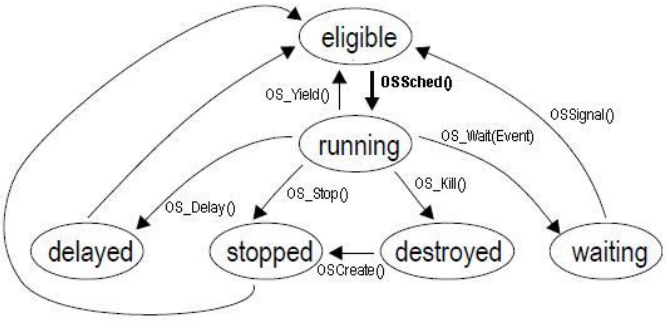
La librería de la placa se denomina UIBlib. Esta librería esta desarrollada por los creadores de la placa, ingenia-cat S.L., e incorpora gran cantidad de funciones para el uso completo de las mismas con el modulo iCM4011 instalado.

Las librerías Salvo implementan el sistema operativo de tiempo real. A continuación se explica con más detalle este software, el más tratado en el desarrollo del proyecto.

IV. SALVO OS

Es un Sistema Operativo de Tiempo Real de un tamaño diminuto y está diseñado para funcionar en microprocesadores y microcontroladores, independientemente de la plataforma. Está modulado en librerías de tal manera que sólo se cargan las funcionalidades usadas y se compila conjuntamente con la aplicación en un solo programa, ahorrando espacio.

Es un sistema multitarea. Las tareas, funciones en C, son como los programas de otros SO que compiten por ejecutarse en el procesador. Tienen siempre una prioridad asignada y están en alguno de los estados de la Figura 1.



La función OSSched() es la encargada de elegir la tarea que va a ser ejecutada. Se selecciona la tarea lista para ejecutarse con mayor prioridad.

Cabe decir que Salvo no es un sistema apropiativo, es decir, el SO no puede retirar una tarea que se esté ejecutando actualmente, sino que es esta quien abandona la ejecución mediante una de las llamadas al sistema que se observa en la figura 1.

El SO proporciona herramientas de concurrencia y comunicación entre tareas como son semáforos, mensajes y colas de mensajes, entre otros.

Cuando se usan estos mecanismos siempre existe una tarea que espera a que se señalice el semáforo o a que haya algún mensaje; y otra que escribe los mensajes o señaliza el semáforo. La tarea que espera por uno de estos mecanismos está en el estado bloqueado waiting. Para salir de este estado se debe señalizar el semáforo o llegarle un mensaje por los que esté esperando.

Otra utilidad del SO son los delays o retrasos. Con estos se puede enviar una tarea al estado delayed durante el numero de clocks (impulsos de reloj) del SO especificado en la llamada al sistema OS_Delay().

El SO necesita una interrupción hardware que sirva de reloj. Se debe crear una interrupción hardware donde se hace una llamada a la función OSTimer(), la cual avanza en una unidad el reloj del SO.

V. OBJETIVOS

La práctica se compone de un Sistema Distribuido de Tiempo Real (SDTR) formado por cuatro placas UIB-PC104, con módulo iCM4011, conectados mediante una red CAN sobre un bus serie.

A grandes rasgos consiste en realizar un sistema de ventas por básculas, con una base de datos (BD) de la cotización de cada producto y un supervisor para mantener un registro de las ventas. El número de productos diferentes que pueden ser seleccionados por el usuario es cuatro. Los objetivos para cada dispositivo se explican con más detalle a continuación.

A. Básculas

El usuario interactúa con las básculas a través del teclado para seleccionar el producto y aceptar la venta, a través del potenciómetro para simular la medición del peso y a través de la pantalla LCD para visualizar la información.

El peso tiene un rango de valores entre 0,000Kg y 9,999Kg. Una vez que el usuario ha seleccionado el producto y ha simulado la medición del peso, se hace una petición de la cotización del producto a la BD. Posteriormente la venta realizada se envía al supervisor a través de la red CAN para llevar un registro de las ventas.

B. Base de Datos (BD)

La base de datos permite al usuario modificar las cotizaciones de los productos desde 0,00$ hasta 9,99$. La interacción se realiza a través del teclado para modificar el valor que se visualiza por la pantalla LCD. Además, la base de datos responde automáticamente a las solicitudes CAN enviadas por las diferentes básculas del sistema.

C. Supervisor

Este dispositivo no interactúa con ningún usuario, únicamente con un ordenador a través del módulo UART. Se encarga de recibir las ventas por la red CAN desde las básculas y transmitir la información a un registro o log en el ordenador, donde cada línea corresponde con una venta.

VI. DESARROLLO

Los diferentes campos a tener en cuenta son: la comunicación CAN, la programación con Salvo de cada dispositivo y aspectos generales o comunes.

A. Red CAN

Los mensajes en una red CAN se identifican por el tipo de información que llevan. Hay dos tipos de mensajes: información de cotizaciones e información de ventas. Estos a su vez se subdividen en los ocho pares bascula-producto.

Así pues, para enviar información de cotizaciones tenemos ocho identificadores, por cada par báscula-producto. El contenido del mensaje es el valor de la cotización de ese producto. En caso de pedir la información desde las básculas a la BD se envía lo que se conoce como una trama remota.

Para el caso de la información de ventas ocurre lo mismo, teniendo ocho identificadores de mensajes más. Los datos del mensaje son el peso y el precio del producto.

El módulo CAN de las básculas se configura de tal manera que sólo reciba los mensajes con la información de interés para éste. La recepción se realiza por una interrupción Hardware.

Los valores decimales son transformados a enteros con potencias de 10. De esta forma, en vez de enviar 4 bytes de datos por decimal, enviamos 2 bytes de datos por entero . Al recibirlos se dividen por la potencia de 10 usada para el envío.

B. Aspectos generales

Aunque los datos son enviados como enteros por la red, internamente todos los nodos trabajan con decimales para tratar las variables de peso, precio y cotización.

Los demás aspectos generales están directamente relacionados con el SO Salvo.

La versión gratuita usada de Salvo, Salvo Lite, sólo permite la creación de un máximo de 3 tareas, 3 semáforos y 1 variable de mensajes para cada dispositivo. Esto impone restricciones determinantes en la implementación de los programas.

En los programas de las básculas y de la BD existen dos aspectos comunes: el uso de Timers y el polling de teclado (detección de teclas pulsadas).

Ambas aplicaciones usan retrasos en alguna de sus tareas. Para el uso de funciones de retraso de Salvos se ha implementado una interrupción Hardware periódica de 0,1s. En ella se hace la llamada al SO OSTimer() (explicada en la sección IV), lo que permite realizar retrasos en las tareas múltiplos de este periodo.

El polling de teclado utiliza la única variable de mensaje permitida por Salvo Lite. Esta función comprueba que se haya pulsado alguna tecla, y si es el caso pone su valor en el mensaje. Se haya pulsado o no una tecla, al final de la comprobación se retrasa durante 0,1s, es decir, es una tarea que periódicamente (cada 0,1s) comprueba las teclas pulsadas.

C. Básculas

Ésta es la aplicación más compleja. Tiene dos interrupciones hardware: interrupción periódica para la llamada al SO OSTimer(), explicado anteriormente, y otra para la recepción de los mensajes CAN.

Se han implementado 3 tareas, 1 semáforo y 1 variable de mensajes:

1) *P_SEM_CAN*. Éste es un semáforo de Salvo. Se inicializa a cero (o bajado) y se señaliza por la interrupción de CAN para indicar que se ha recibido un mensaje CAN. La tarea TaskGui() espera por este semáforo cuando tiene que recibir un mensaje CAN con la cotización de un producto.

2) *P_MSG*. Ésta es una variable tipo mensaje de Salvo. Es usada para almacenar el valor de la tecla pulsada. Es escrita por la tarea TaskPolling() cuando una tecla es pulsada y espera por ella la tarea TaskGui(), en los momentos en que espera a que el usuario pulse alguna tecla.

3) *TaskPolling()*.Esta tarea, ya mencionada, se ejecuta periódicamente cada 0,1s por llamadas a la función OSDelay(). Comprueba si se ha pulsado alguna tecla y si es así escribe su valor en P_MSG para 'despertar' a TaskGui() si está esperando por algún valor de teclado. Esta tarea tiene la más alta prioridad para que se pueda leer correctamente una pulsación de teclado y no es bloqueada por ninguna otra, sólo por ella misma cada 0,1s para dejar paso a las demás.

4) *TaskGui()*.Es la tarea más extensa y que incorpora la mayor parte de la aplicación. Se descompone en 4 fases: obtención de producto, petición de cotización, cálculo y visualización de precio según peso y cotización, y envío de datos de venta.

En la primera fase, únicamente espera a que se pulsen teclas del 1 al 4 que permiten la selección de un producto hasta que se pulse la tecla S, que indica la selección de ese producto.

En la segunda fase, se envía una petición con el identificador CAN correspondiente al número de báscula y producto seleccionado. Espera contestación por parte de la BD durante 1s; sino recibe contestación, repite el proceso una segunda vez. Sino recibe contestación, escribe un mensaje de error por pantalla y vuelve a la primera fase. En caso de éxito pasa a la siguiente fase.

En la tercera fase, se procede a la simulación del pesado del producto. En ella se visualiza el precio del producto en base al valor de su cotización y al valor del potenciómetro 1 de la placa, guardado en una variable global escrita por la tarea TaskPesada(). La tarea se bloquea durante un periodo de tiempo de 0,2s a la espera de la pulsación de la tecla S.

En la cuarta fase se envía el valor del peso y precio al supervisor con el identificador correspondiente a la bascula y producto seleccionado. Los valores de estos se convierten de decimal a entero antes de su envío.

Posteriormente, se vuelve al comienzo de la primera fase.

5) *TaskPesada()*. Esta tarea se activa periódicamente cada 2s ya que se supone que el valor del peso del producto no varía en periodos de tiempo cortos. Su función es obtener el valor digital del potenciómetro a través de una función de la librería UIBlib y transformarlo en un valor entre 0,000Kg y 9,999Kg.

6) *Interrupción CAN*. Esta interrupción esta configurada para recibir únicamente tramas del tipo cotización pertenecientes al número de báscula. Cuando se recibe el mensaje, transforma el valor de la cotización de entero a decimal dividiéndolo entre 100. Para finalizar, activa el semáforo P_SEM_CAN para indicar a la tarea TaskGui() que se a recibido la cotización.

D. Base de datos.

Este programa tiene dos interrupciones hardware, una para el Timer y otra para la recepción CAN, 3 tareas, 1 semáforo y 1 variable de mensajes:

1) *P_SEM_CAN*. Un semáforo de SALVO que se inicializa bajado y se señaliza por la interrupción CAN para

indicar que se ha recibido un mensaje. La tarea TaskCotizacion() espera por este semáforo como se describe mas adelante.

2) TaskPolling(). Esta tarea es también idéntica a la descrita en las básculas, con la diferencia que posee la prioridad intermedia entre las tres tareas.

3) TaskGui(). Tarea que comprende la interacción con el usuario del sistema. Se mantiene bloqueada en estado waiting a que se reciba la pulsación de una tecla. Cuando TaskPolling() escribe en P_MSG, la tarea es desbloqueada. La prioridad de esta tarea es la menor debido a que es la que necesita menor tiempo real.

El sistema de teclas implementado se basa en el sistema de cruceta. Es decir, las teclas 4 y 6 sirven para desplazarse por los diferentes productos que dispone la base de datos, y las teclas 2 y 8 se encargan de incrementar y decrementar el precio, respectivamente. Todas las modificaciones realizadas sobre el precio de los productos se ven reflejadas en la pantalla LCD.

*4) TaskCotizacion().*Esta tarea se encarga de responder a las basculas cuando estas hacen una petición de cotización. Espera la activación del semáforo P_SEM_CAN que indica que se ha recibido una petición de cotización por parte de alguna báscula.

Se envía una trama CAN con el mismo identificador adjuntado los datos solicitados. Los datos se convierten de decimal a entero multiplicándolos por 100, ya que las cotizaciones tienen 2 decimales.

Esta tarea tiene la prioridad más alta debido a la necesidad de responder lo más pronto posible a las solicitudes de las básculas.

5) Interrupción CAN. Esta interrupción se configura solo para recibir tramas del tipo cotización por parte de las básculas. Las tramas a las que el dispositivo debe responder son tramas remotas (sin datos) que sirven a modo de petición. Posteriormente guarda el identificador de la trama en una variable global y activa P_SEM_CAN para dar paso a TaskCotizacion().

E. Supervisor

Este programa es el más sencillo de todos. Se compone únicamente de 1 interrupción de recepción CAN, 1 tarea y 1 variable de mensajes.

1) *P_MSG.* Ésta es una variable tipo mensaje de Salvo. Esta variable es escrita por la interrupción CAN con los datos de la trama CAN recibida. Posteriormente es leída por la tarea TaskEscribirLog() que se describe a continuación. El mensaje que se envía es una variable compuesta (estructura) que contiene el numero de bascula, el identificador del producto, su precio y peso; enviados anteriormente desde las basculas

2) *TaskEscribirLog().* La tarea se mantiene bloqueada a la espera de que se escriba un mensaje en P_MSG. Una vez recibido el mensaje se escribe el contenido en una variable de tipo String para ser enviada por el modulo UART hacia un ordenador.

La conexión se realiza a través de la conexión USB. Para el envío y la configuración del módulo UART se usan las funciones de la librería UIBlib destinadas para ello.

En el ordenador se pueden recibir los datos a través de HyperTerminal en Windows o en Linux leyendo el fichero del dispositivo en la carpeta /dev.

3) *Interrupción CAN.* Esta interrupción se configura sólo para recibir tramas del tipo venta enviadas por las básculas. Realiza la conversión de los enteros de peso y precio a decimales. Escribe el número de báscula y el tipo de producto, parte del identificador CAN, en una variable compuesta; además de los valores decimales de peso y precio. Esta variable se envía por mensaje a través de P_MSG a TaskEscribirLog().

VII. CONCLUSIONES

Los sistemas de tiempo real son de gran importancia. Hoy en día están presentes en una gran cantidad de componentes electrónicos que usamos habitualmente, como son sistemas ABS, control de motores de coches, también muchos electrodomésticos de hoy en día, como son dispositivos multimedia.

Está clara la importancia de estos sistemas hoy en día. El número de estos dispositivos supera el del número de los de 'propósito general'.

Con esta práctica se puede ver el potencial y las características de estos sistemas; pero sobretodo su filosofía de programación de eventos y tareas.

REFERENCIAS

[1] Pumpkin, Inc. Home of Salvo ™. The RTOS that runs in tiny places. [Online]. Available: http://www.pumpkininc.com/

[2] DsPIC30F4011/4012 Data Sheet, High Perfomance, Digital Signal Controllers, Microship Tecnology In 2005.

[3] Ingenia Motion Control Solutions, digital servo drives and software for high precision brushless and DC. [Online]. Available: http://www.ingenia-cat.com/

Práctica realizada en la asignatura *Sistemas Empotrados.*
Profesores: Julián Proenza y Guillermo Rodríguez-Navas.

Bartolomé Martínez Pérez.
Estudiante de la titulación de Ingeniería Técnica en Informática de Sistemas en la Universitat de les Illes Balears (UIB).

Emilio Arenas Bosch.
Estudiante de la titulación de Ingeniería Técnica en Informática de Sistemas en la Universitat de les Illes Balears (UIB).