Uso colaborativo del procesador en sistemas embebidos para múltiples interfaces

Waldo Valiente, Esteban Carnuccio, Mariano Volker, Graciela De Luca, Raúl Villca, Germán Lorenz
Departamento de Ingeniería e Investigaciones Tecnológicas
Universidad Nacional de La Matanza
Dirección: Florencio Varela 1703 – CP 1754 –
{wvaliente, ecarnuccio, mvolker, gdeluca }@unlam.edu.ar,
{raul.villcasd, germangelv}@gmail.com

RESUMEN

Las soluciones IoT requieren de sistemas embebidos cada vez más complejos, como es el caso de nuestra investigación para el monitoreo de personas, asistencia ante caídas, signos vitales anómalos y seguridad. Donde muchas interfaces entre los sensores y los canales de comunicación se deben atender junto con la lógica del programa principal. Estos a su vez se construyen sobre pequeñas arquitecturas con procesadores de rendimiento moderado impulsados por el bajo consumo energético que se busca. En el presente trabajo se analizan técnicas de programación alternativas y funcionalidades provistas por los Sistemas Operativos de Tiempo Real, que brindan la posibilidad de alternancia de tareas sobre un único procesador. También al mismo tiempo se evalúan las necesidades de recursos, que en estos tipos de procesadores son reducidas. Se proponen soluciones para compartir el procesador evitando en lo posible el uso de funciones provistas por los sistemas operativos de tiempo real, ya que estos consumen una cantidad de recursos considerables para estas arquitecturas. Se propone solucionar esta dificultad mediante la implementación de una máquina de estados a través del uso de interrupciones, como única solución o una combinación de ambas.

Palabras clave: *RTOS, interrupciones, monitoreo de personas, máquina de estado, funciones bloqueantes.*

CONTEXTO

Nuestra Línea de Investigación es parte del proyecto "*Dispositivo de asistencia de personas mediante monitoreo y análisis de datos en la nube"*, dependiente de la Unidad Académica del *Departamento de Ingeniería e Investigaciones Tecnológicas*, perteneciente al programa de Investigaciones CYTMA2 de la Universidad Nacional de La Matanza, el cual está formado por docentes investigadores y alumnos de las carreras de ingeniería en informática e ingeniería en electrónica. Este proyecto es continuación de los trabajos que viene realizando el grupo de investigación, en sistemas operativos, computación de alto rendimiento e Internet de las cosas (IoT), entre otros.

1. INTRODUCCIÓN

Los sistemas embebidos que forman parte de un sistema complejo de IoT [1], se construyen sobre arquitecturas simples y colaborativas como el procesador ARM de la familia Cortex-M3 [2] [3]. Estos sistemas tienen la complejidad de atender y gestionar los recursos para múltiples interfaces sobre un único hilo de ejecución. La dificultad radica en cómo repartir el recurso escaso, en este caso el procesador, brindando atención a cada interfaz, siempre manteniendo los tiempos de respuesta requeridos en todos los dispositivos que se utilizan. En nuestra investigación [4] se evidencia que cuando se utilizan interfaces complejas se requiere realizar una serie de pasos o procedimientos compuestos para lograr dar respuesta a una solicitud. Como puede ser el caso de conexión a Internet por GSM, conexión local por Bluetooth y obtener datos desde el GPS, junto a otros sensores que requieren un muestreo periódico. Tal es el caso del acelerómetro, donde se evalúan las mediciones para verificar si la persona que lleva el dispositivo se encuentra en una caída o no.

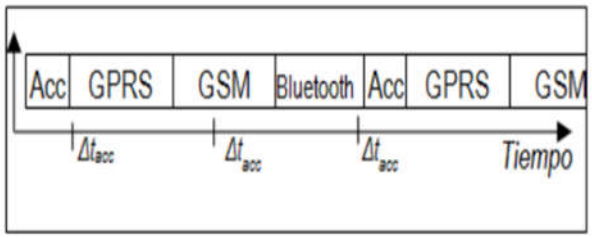

Figura 1 – Múltiples funciones sobre un único hilo de ejecución.

Se puede observar en la Figura 1 que sobre un único procesador ejecutan varias funciones. La función que requiere ejecutar regularmente y la del acelerómetro (Acc), no siempre logra cumplir con su periodicidad (Δt_{acc}) requerida para el correcto funcionamiento del algoritmo de caídas.

2. LINEAS DE INVESTIGACIÓN Y DESARROLLO

A continuación se plantean distintos métodos para lograr repartir el uso del único procesador entre varias funciones que se necesitan ejecutar del sistema embebido y reconociendo aquellas que son críticas.

Las implementaciones en sistemas embebidos podrían clasificarse en non-os, bare-bones con RTOS [1] y Full RTOS. En el caso de una implementación sin S.O se programa un binario con todo el programa en sí, que realiza la inicialización-ejecución principal y finalización. En el caso de bare-bones con RTOS es un mix entre ambos métodos. Por último en el caso del uso de Full RTOS, la funcionalidad necesaria se implementa como una aplicación convencional en un sistema operativo de tiempo real, Figura 2.

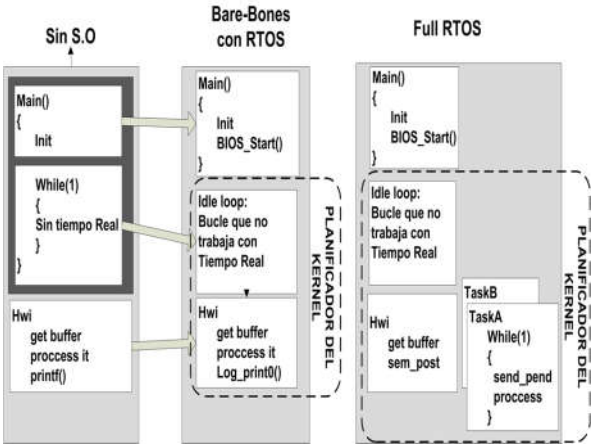

Figura 2 –Clasificación de implementación en sistemas embebidos.

Cambios de contexto por hardware

La arquitectura ARM Cortex-M3 [5] facilita dos mecanismos para interrumpir el programa en ejecución. El primero *SysTick* es un reloj de 24 bits que puede interrumpir al procesador cada vez que su contador llegue a cero. Este mecanismo es utilizado para realizar cambios de

[1] RTOS Real Time Operating System

contexto al estilo *round-robin*, Figura 3.

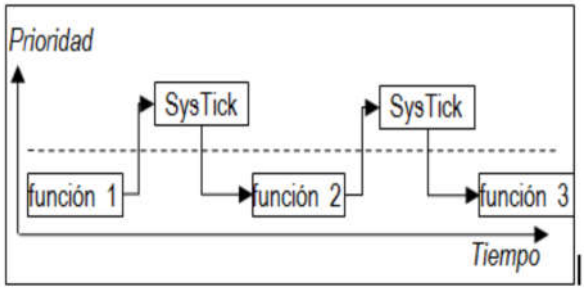

Figura 3 - Interrupción SysTick.

En el segundo mecanismo *PendSV* el programa permite ceder el control del CPU al manejador, cuando este se encuentre inactivo. Se puede utilizar con cambios de contexto al estilo de FIFO Figura 4.

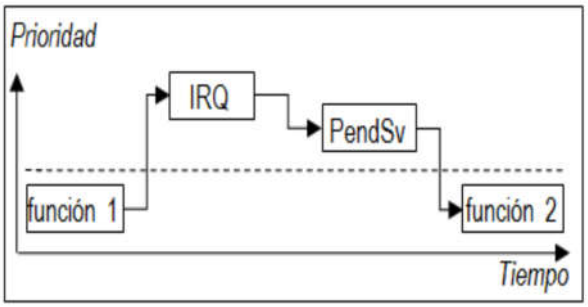

Figura 4 - Interrupción PendSV.

La ventaja de este método es que el programador tiene el control y puede definir el comportamiento que necesite para su implementación. La desventaja se tiene, cuando debe desarrollarse la lógica del planificador en forma más robusta, esta tarea suele ser engorrosa y difícil de probar la integridad del mismo.

Cambios de contexto por software

Los cambios de contexto se realizan agregando bibliotecas de terceros en el programa. Estas poseen tal nivel de integración brindando servicios directamente a nivel de Sistema Operativo de Tiempo Real (RTOS) [6] en reducido tamaño, que facilitan su programación e implementación. En el mercado existen abundante cantidad de desarrollos para la arquitectura en cuestión, a continuación se enumeran los más destacadas:

La versión de *mbedOS* [7], entre sus principales características se destaca la seguridad y la conexión de dispositivos, mientras que *FreeRTOS* [8], facilita la programación, implementación e integración con servicios en la nube de Amazon™. El caso de *NuttX* [9], pensado para dispositivos desde 8 a 32bits basado en el estándar de APIs Unix. El pequeño kernel de *µ-velOSity* [10], es diseñado para ser eficiente y ocupar poca memoria. *RTEMS* [11], es soportado en gran cantidad de arquitecturas. *ChibiOS* [12], es un pequeño kernel que permite la integración con otros componentes *OpenSource* para brindar más servicios.

Los RTOS facilitan la implementación ya que los cambios de contexto, junto con otras funciones de comunicación de tareas, ya se encuentran integradas en el kernel, el desarrollador solo se limita a desarrollar la lógica de la aplicación para las tareas que ejecutan ver Figura 5.

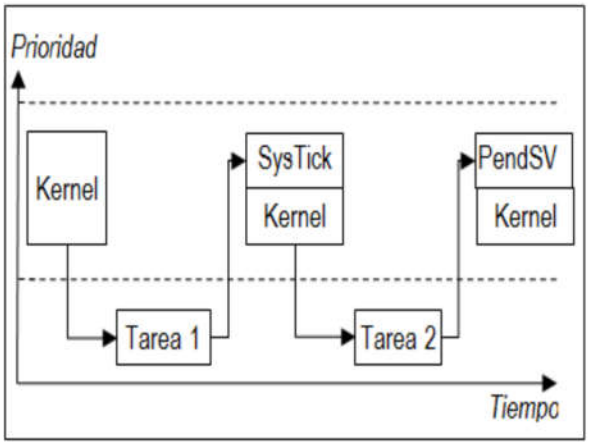

Figura 5 – Cambio de contexto utilizando RTOS

La ventaja de este mecanismo es que soluciona

los problemas planteados con bibliotecas probadas por la comunidad y acelera el tiempo de desarrollo. La desventaja que es dificultosa la coordinación entre tareas fuertemente acopladas y la utilización de software de terceros amplía el espacio que ocupara el programa en el reducido espacio de memoria que posee la arquitectura embebida.

Modelo máquina de estado finito

La alternativa que plantea nuestro grupo de investigación, permite distribuir el uso del único procesador entre varias funciones. La misma radica en descomponer la lógica de cada función en sub-partes no bloqueantes. Así el programa principal, compagina su ejecución entre las distintas sub-partes de cada función.

El método encontrado consiste en organizar esta división utilizando el modelo de máquina de estado finito. Así las funciones de las interfaces tienen un comportamiento distinto, según el estado que se encuentre. En cada interacción del programa se evalúa el estado en el que se encuentra la aplicación, ejecutando la pequeña sub-parte que le corresponde. A continuación, se evalúa la siguiente porción de código de la función siguiente.

En la Figura 6, se observa que la función del acelerómetro (Acc) logra cumplir con la periodicidad requerida. Mientras se altera la ejecución de las sub-partes del GPRS (G_{e1},G_{e2},G_{e3},G_{e4}), para el GSM (S_{e1},S_{e2},S_{e3}) y Bluetooth (B_{e1},B_{e2}).

Figura 6 – Alternancia por estado de funciones.

La ventaja de esta solución es que no ocupa mucho lugar en memoria y no requiere de grandes cambios de implementación, comparándolo con la migración del programa al utilizar RTOS. Como desventaja, sin un buen diseño de la transición de estados, puede haber partes de funciones que nunca se ejecuten. Otra desventaja es que no se diferencian prioridades entre las funciones de interfaces.

3. RESULTADOS OBTENIDOS/ESPERADOS

Luego del análisis de las alternativas se confirma, que se puede obviar la utilización de un RTOS siempre que el programa no utilice funciones bloqueantes. Los RTOS son una solución compleja que brinda beneficios en casos que el código sea bloqueante, sin poder desarrollarlo de otra forma. La propuesta de máquina de estado es equitativa en el uso del procesador, pero no satisface estrictamente el cumplimiento de la periodicidad necesaria en la función crítica. En la investigación en curso se realiza la prueba de una solución hibrida. La lógica de las interfaces de GSM, GPRS y Bluetooth, menos prioritarias, se desarrollarán siguiendo el modelo de máquina de estado finito, que permite el uso del procesador colaborativamente.

Para la interfaz de acelerómetro, que es pieza clave en la detección de caídas. Se realizara un pequeño módulo que permita cambios de contexto utilizando interrupciones *Systick*. Luego de tener las mediciones periódicamente, se evaluará el algoritmo de caídas junto con las demás interfaces antes mencionadas.

4. FORMACIÓN DE RECURSOS HUMANOS

La presente línea de investigación dentro del departamento de Ingeniería e Investigaciones

Tecnológicas forma parte del trabajo que dos investigadores se encuentran realizando su tesis de maestrías. Completan el grupo de investigación cuatro alumnos de Ingeniería en Informática que se encuentran finalizando su formación de grado y realizan su iniciación a la investigación.

5. BIBLIOGRAFÍA

[1] Bonilla-Fabela, Morales-Escobar y Guajardo-Muñoz, «IOT, El Internet de las Cosas y la Innovación de sus Aplciaciones,» 2 ed., 2016, pp. 2313-2340.

[2] STMicroelectronics, «Referece Manual - STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs,» STMicroelectronics, 2018.

[3] STMicroelectronics, «Datasheet - production data Ultra-low-power 32-bit MCU ARM®-based Cortex®-M3,» de *STM32L151x6/8/B STM32L152x6/8/B*, STMicroelectronics, 2016.

[4] E. C. M. V. G. D. L. G. G. D. G. R. V. M. V. Waldo Valiente, «Dispositivo de asistencia de personas mediante monitoreo IoT,» SEDICI, San Luis, 2018.

[5] T. Martin, «The Designer's Guide to the Cortex-M Processor Family,» India, elseiver, 2016, pp. 71-130.

[6] L. P. L. P. M. T. Fei Guan, «Open source FreeRTOS as a case study in real-time operating system evolution,» de *Journal of Systems and Software*, elsiever, 2016, pp. 19-35.

[7] A. Mbed, «Mbed OS,» 2018. [En línea]. Available: https://www.mbed.com/en/platform/mbed-os/.

[8] Amazon Web Services, «Amazon FreeRTOS Sistema operativo compatible con IoT para microcontroladores,» 2019. [En línea]. Available: https://aws.amazon.com/es/freertos/.

[9] G. Nutt, «NuttX Real-Time Operating System,» 2019. [En línea]. Available: https://nuttx.org/.

[10] Green Hills Software, «µ-velOSity real-time operating system (RTOS),» 2019. [En línea]. Available: https://www.ghs.com/products/micro_velosity.html.

[11] On-Line Applications Research (OAR) Corporation, «RTEMS - An Open Real-Time Operating System,» 2019. [En línea]. Available: http://rtems.com/.

[12] G. D. Sirio, «ChibiOS free embebbed RTOs - ChibiOS Homepage,» 2017. [En línea]. Available: http://www.chibios.org/dokuwiki/doku.php.

[13] STMicroelectronics, «Datasheet - production data- Medium-density performance line ARM®,» de *STM32F103x8*, STMicroelectronics, 2015.