XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0906

Control de la Ejecución en Sistemas de Criticidad Mixta

A. Crespo, P.Balbastre, J. Simó
Universitat Politècnica de València, (acrespo,patricia,jsimo)@ai2.upv.es

J. Coronel
FentISS, (jcoronel@fentiss.com)

Resumen

En sistemas de tiempo real y, en general, en sistemas críticos, hay una tendencia en alza en utilizar aplicaciones con diferentes niveles de criticidad. Las soluciones basadas en hipervisores son una forma de implementar sistemas de criticidad mixta ya que proporcionan aislamiento temporal y espacial. Sin embargo, la ejecución de una partición puede ser afectada por la ejecución en otros núcleos, lo que se conoce comúnmente como interferencias, poniendo en peligro la ejecución en el tiempo especificado. En este artículo se intenta contribuir dando soluciones realistas a este problema. Se propone una solución de control de la planificación con dos controladores al nivel del hipervisor. Uno de los controladores está orientado a limitar el uso de recursos compartidos a través de la limitación en el uso de los buses en los núcleos no críticos. El otro controlador mide la actividad del núcleo crítico y tomas decisiones sobre la ejecución de los núcleos no críticos.

Palabras clave: Hipervisor, sistemas de criticidad mixta, control retroalimentado.

1. Introducción

En sistemas de tiempo real y, en general, en sistemas críticos, hay una tendencia en alza en utilizar aplicaciones con diferentes niveles de criticidad donde varios componentes con diferentes restricciones temporales se integran conjuntamente en una misma plataforma [7]. Las razones detrás de esta tendencia son principalmente no funcionales: reducir costes, volumen, peso y potencia consumida para diferentes sectores como el del control industrial, aviónica, espacio y automóvil por citar sólo algunos. Las capacidades de procesamiento que los sistemas mutiprocesador empotrados pueden alcanzar permite unificar en la misma plataforma hardware cada vez más funcionalidades y aplicaciones. En ambos casos, hay una necesidad de integración de aplicaciones críticas y no críticas. Esta integración es conocida actualmente en la comunidad de tiempo real como sistemas de criticidad mixta [5].

Desde el punto de vista de la arquitectura software, hay una tendencia en utilizar técnicas de virtualización para proporcionar aislamiento temporal y espacial a las arquitecturas particionadas. Esta técnica se utilizó por primera vez en el sector de la aviónica[17] y se extendió posteriormente al sector espacial[18]. El soporte de virtualización de las particiones es proporcionada por el hipervisor. Los hipervisores son capas de software que aprovechan las funcionalidades del hardware para establecer entornos de ejecución independientes. Esta tendencia se ha ido extendiendo y actualmente la mayoría de los procesadores ofrecen soporte de virtualización añadiendo un nivel más al modo de operación del procesador.

Las soluciones basadas en hipervisores son una forma de implementar sistemas de criticidad mixta, especialmente en sistemas multi núcleo. Las principales propiedades del hipervisor tiene que ver con el aislamiento temporal y espacial de las particiones software que se ejecutan por encima del hipervisor. Una partición se define como un entorno de ejecución que contiene una aplicación y su propio sistema operativo ejecutándose en su espacio de memoria propio. Los mecanismos del hipervisor proporcionan servicios para virtualizar los recursos hardware a las particiones. El hipervisor XtratuM [9] es un hipervisor bare-metal para sistemas empotrados desarrollado en varios proyectos europeos [16] [10] y se está utilizando actualmente en varios misiones en el sector espacial.

Unos de los puntos cruciales en los sistemas críticos multi-núcleo es el aislamiento temporal. A este respecto, el hipervisor puede garantizar recursos temporales exactos y predecibles a las particiones. Sin embargo, la ejecución de una partición puede ser afectada por la ejecución en otros núcleos, lo que se conoce comúnmente como interferencias. Este problema, debido al uso de recursos compartidos, puede afectar a la ejecución de una partición crítica poniendo en peligro la ejecución en el tiempo especificado. Por tanto, estas interferencias introducen un factor de impredecibilidad en la ejecución de una tarea crítica y no permite estimar un límite superior en el tiempo de ejecución

de peor caso (WCET), introduciendo anomalías temporales [14].

En este artículo, nos centraremos en el control de la ejecución de sistemas particionados de criticidad mixta sobre un hipervisor. Se propone una solución de control de la planificación con dos controladores al nivel del hipervisor. el resto del artículo se organiza de la siguiente forma: el apartado 2 cuenta el estado del arte en este área. El apartado 3 describe los contadores de prestaciones utilizados por el hipervisor para definir el control. En el apartado 5 se presentan los objetivos del controlador y los mecanismos utilizados. El apartado 6 describe el ámbito de ejecución de los controladores. El apartado 7 presenta los resultados de los experimentos realizados sobre una plataforma multi núcleo. Finalmente, en el apartado 8 se comentan las conclusiones y el trabajo futuro.

2. Estado del arte

Hay muchos trabajos que han abordado el problema de los recursos compartidos en sistemas multi núcleo. En [1], se analiza el impacto de los buses compartidos, caches y otros recursos y sobre predicción de prestaciones.

En [6], se propone un protocolo de bus basado en TDMA junto con un segundo protocolo dinámico que facilita la integración de sistemas de criticidad mixta libre de interferencias.

En [12] se presenta un controlador de acceso a recursos compartidos por las tareas en ejecución. En [20], se define un mecanismo de protección de memoria que regula los accesos a la misma. En [13], se propone un controlador distribuido que para las tareas de baja criticidad cada vez que detecta que su ejecución puede hacer perder plazos a tareas de alta criticidad. En [19] se propone una infraestructura de planificación que utiliza contadores de prestaciones para calcular la latencia media de acceso a memoria.

Por otro lado, las técnicas de control y planificación controlan el uso de los recursos de una plataforma computacional por medio de controladores que actúan sobre las tareas en ejecución. Se han definido muchas estrategias en este sentido, entre otras: optimización del periodo [2], memoria como recurso compartido [15], ajuste de los periodos de las tareas para minimizar el hiperperiodo [3].

3. Contadores de monitorización de prestaciones

Los procesadores actuales proporcionan mecanismos para monitorizar el funcionamiento del sistema, ofreciendo contadores de eventos que pueden ser leídos por las aplicaciones o los sistemas operativos. El monitor de prestaciones (Performance Monitor, PM) es un dispositivo que proporciona este mecanismo en el procesador multi núcleo PowerPC T2080 en la placa NXP QorIQ T2080RDB [11]. El procesador T2080 incluye cuatro núcleos de 64 bits e6500. El núcleo e6500 incluye un PM que proporciona un conjunto de PMCs (contadores de monitorización de prestaciones) para definir, habilitar y contar ciertas condiciones que pueden disparar la interrupción del monitor de prestaciones. Cada core puede configurar hasta seis contadores de 32 bits para almacenar eventos específicos.

Adicionalmente, el T2080 proporciona extensiones de hardware para virtualización y define un modo hipervisor que permite la ejecución del hipervisor. XatratuM ha sido portado a este procesador y proporciona virtualización completa, manejando los PMCs para almacenar eventos concretos durante la ejecución. Utilizando el PM, XtratuM proporciona la habilidad de contar eventos predefinidos por núcleo asociados con operaciones específicos, como ciclos de procesador, instrucciones ejecutadas, pérdidas de chace L1 y L2, accesos a bus de datos e instrucciones, etc.

Para cada contador se puede definir un umbral para disparar los eventos cuando un cierto valor es alcanzado. Los contadores pueden ser habilitados o deshabilitados según la necesidad del hipervisor o de las aplicaciones.

4. Planificación de hipervisor

XtratuM es un hipervisor bare-metal específicamente diseñado para sistemas empotrados de tiempo real, ofreciendo para-virtualización o virtualización completa dependiendo del soporte del hardware.

Xtratum define el concepto de CPU (núcleo) virtual (vCPU), las cuales son abstracciones que modelan el comportamiento de la CPU y pueden ser asignadas a las CPUs reales. XtratuM abstrae tantas vCPUS como núcleos físicos existan. Las particiones pueden ser mono núcleo o multi núcleo (utiiizan una o varias vCPUs). La asignación de vCPUS a CPUs se especifica en el fichero de configuración donde se define el sistema completo: hardware, dispositivos, planificación temporal, canales de comunicación, etc.

Una partición puede ser una aplicación compilada para ejecutarse sobre máquina desnuda, una aplicación de tiempo real con su propio sistema operativo de tiempo real o una aplicación ejecutándose sobre un sistema operativo de propósito

general. Las aplicaciones multi núcleo requieren un sistema operativo SMP y asignar varias CPUs a la partición.

La arquitectura software de un sistema particionado multi núcleo se muestra en la Figura 1. En la figura se muestra un sistema integrado por 5 particiones mono núcleo y una partición multi núcleo. También se muestra la asignación de núcleos virtuales a núcleos reales. P0 y P1 se asignan a la CPU0, P3 y P4 a la CPU1, P5 a la CPU2 y las dos vCPUs de P6 a las CPU2 y CPU3 respectivamente. Todas las tareas de una partición se ejecutan en su CPU asignada. La partición SMP planifica internamente qué tareas se asignan a cada vCPU y, en consecuencia, en qué CPU real se ejecutan.

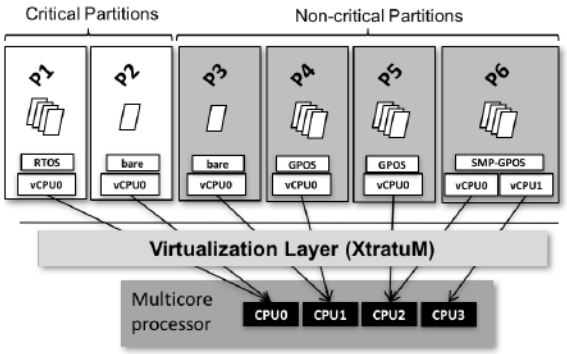

Figura 1: Arquitectura particionada en plataforma multi núcleo

La generación del plan cíclico debe considerar el impacto de los recursos compartidos (accesos a bus, caches L2, memoria, etc) en la ejecución de las tareas. En la herramienta Xoncrete [4] se implementan estas técnicas de planificación. En [8], se define una metodología para generar planes cíclicos en sistemas multi núcleo de criticidad mixta.

El número de núcleos se determina calculando la utilización de cada partición y asignando las particiones a los núcleos mediante técnicas de bin-packing.

El plan generado se carateriza por:

- Todas las particiones críticas se asignan a un subconjunto de núcleos llamados CC.
- Las particiones no críticas se asignan a otro subconjunto llamado NCC.
- Cada tarea en una partición tiene asignado su propio slot en un núcleo.
- Se considera el tiempo de ejecución de peor caso incrementado por un factor que tiene en cuenta la interferencia entre núcleos.

En este artículo se asume que las particiones críticas están alojadas en un solo núcleo mientras que las no críticas pueden estar en varios núcleos. Sin embargo, el esquema de control propuesto es compatible con la ejecución de particiones no críticas en el núcleo crítico CC. Como trabajo futuro se considerarán más de un CC.

5. Controladores de ejecución

El objetivo de los controladores es limitar la interferencia producida por las aplicaciones de los núcleos no críticos (NCP) sobre el crítico (CP).

Las aplicaciones críticas ejecutándose en los núcleos críticos tienen que asegurar el cumplimiento de las restricciones temporales y, en consecuencia, los controladores no pueden actuar directamente sobre su ejecución. Sin embargo, los núcleos no críticos pueden suspender su ejecución durante los intervalos en que su ejecución conprometa las aplicaciones críticas.

La figura 2 muestra el esquema de control propuesto.

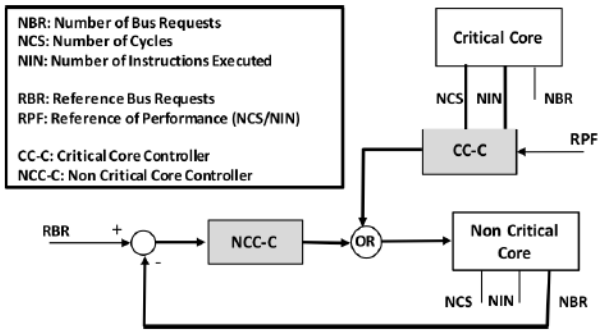

Figura 2: Esquema de control

Inicialmente, se consideraran 2 núcleos. El C_0 ejecuta partitiones críticas y el C_1 las aplicaciones no críticas. El núcleo no crítico (NCC) lee las peticiones de bus y las compara con una referencia determinada. Cuando la lectura excede la referencia, se toma la acción sobre el NCC. En cambio, el núcleo crítico lee los ciclos e instrucciones de la unidad de prestaciones y calcula la relación de ciclos por instruccion (CPI) y si excede la referencia, se toma una acción sobre el núcleo no crítico.

Ambos controladores son basados en eventos con el fin de limitar las ejecuciones del hipervisor y por lo tanto reducir la sobrecarga del sistema. Esto se puede conseguir debido a que los contadores de prestaciones se pueden configurar para generar una interrupción sobre el núcleo correspondiente cuando se alcanza un valor limite que es la referencia.

6. Ámbito de actuación de los controladores

Se asume, por simplicidad, que se dispone de 2 núcleos y 4 particiones. P0 y P1 se consideran críticas y P2 y P3 no críticas.

La implementación de los controladores se realiza a nivel de hipervisor. Para definir su comportamiento se definen un conjunto de requisitos.

Req1 : Si un núcleo crítico está inactivo, su controlador esta deshabilitado.

Req2 : Al empezar la ejecución de una ranura de tiempo de una partición crítica, se activan ambos controladores

Req3 : Cuando el controlador del núcleo crítico toma la acción de suspender la actividad del no crítico, la ejecución de la partición actual y futuras se suspenden.

Req4 : Cuando el controlador del núcleo no crítico toma la acción de suspenderse, la ejecución de la partición actual se suspende hasta que se reanude externamente.

Req5 : Cuando una partición crítica finaliza su ejecución, los controladores se deshabilitan y se reanuda la actividad suspendida en los núcleos no críticos.

Req6 : Las comnicaciones entre núcleos se realizan mediante interrrupciones entre núcleos (IPIs).

Req7 : Las decisiones y acciones a tomar han de ser simples con el fin de reducir la sobrecarga del hipervisor.

Req8 : Los controladores estarán basados en eventos con el fin de acotar el número de interrupciones.

En conclusión, el ámbito de los controladores de núcleo crítico será la ranura de tiempo de su ejecución. Los controladores de los núcleo no críticos será el mismo que el del anterior.

La figura 3 muestra un posible escenario de ejecución y activación de los controladores. La evolución de los contadores de prestaciones se muestra como ejemplo en la figura.

Durante la inicialización del hipervisor, se identifican los núcleos disponibles y se crea un hilo de ejecución interno para cada núcleo (HT0 and HT1). En t_0, HTO detecta el inicio de P0 e identifica (definido en la configuración) que es crítica y habilita los controladores definiendo los valores de referencia de estos. Al mismo tiempo, en t_0, HT1

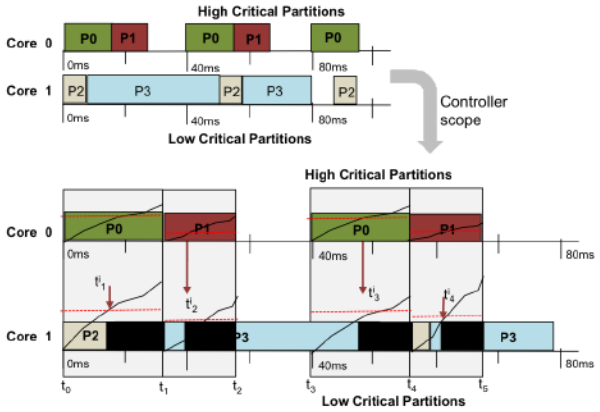

Figura 3: Controller scope

empieza la ejecución de P2 recibiendo una IPI de C_0 que indica que lo habilita. En el intervalo $[t_0, t_1^i]$, HT0 recibe varias interrupciones del contador de instrucciones y calcula el valor del CPI. Cuando este valor sea superior a la referencia suspenderá la actividad del C_1 (t_1^i).

En t_1, P0 termina y envía una IPI a C_1 para que reanude su ejecución. En t_2^i, como consecuencia de la interrupción y del cálculo del PCI, se suspende el C_1.

En t_2, HT0 termina y envía la IPI para que C_1 reanude la ejecución. HT0 lee la siguiente ranura de tiempo y detecta que en el instante t_3 se iniciará. De la misma manera, la acción en t_3^i es suspender, mientras que en t_4^i, C_1 se autosuspende.

El escenario anterior se puede extender a multiples núcleos no críticos. Cada uno de éstos tiene su propio controlador local. La acción tomada por el crítico de suspender la actividad afecta a todos los no críticos. Asimismo, la acción de reanudar la ejecución es para todos ellos.

La figura 4 muestra el esquema de controladores para múltiples núcleos no críticos.

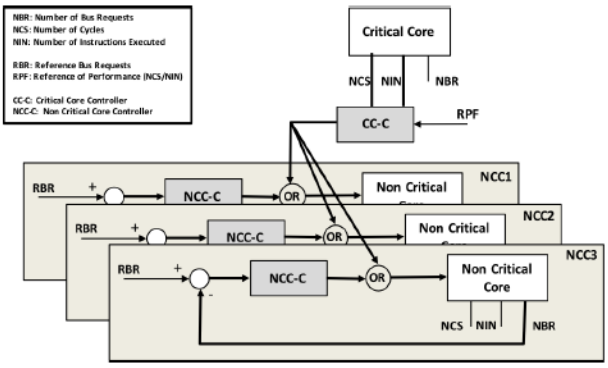

Figura 4: Esquema controladores para múltiples núcleos no críticos

7. Experimentación

En esta sección se detallan los escenarios evaluados y la influencia de los controladores. Después de un detallado análisis de los contadores de prestaciones, se han seleccionado 3 eventos para la implementación. Estos eventos son: PME_PROCESSOR_CYCLES que cuenta el número de ciclos, PME_INSTR_COMPLETED que cuenta el número de instrucciones ejecutadas y BIU_MASTER_REQUESTS que indica el número de peticiones de acceso a bus. Los eventos pueden contar cuando el procesador está en modo usuario, núcleo o hipervisor.

Estos escenarios se han ejecutado en la plataforma T2080 con una partición crítica ($CPart$) y de 0 a 3 particiones no críticas en los otros núcleos. Las particiones no críticas se consideran que van a interferir en la crítica y se identifican como dummys. El solapamiento entre la crítica y las no críticas varía en los experimentos con el fin de mostrar distintas situaciones y las acciones tomadas. El objetivo es medir el tiempo de respuesta de la partición crítica comparando su ejecución de forma aislada con su ejecución cuando hay otros núcleos en ejecución. Se han definido cuatro escenarios: $SC1$ los controladores no están activos, $SC2$ utiliza controladores locales, $SC3$ usa el controlador del núcleo crítico y $SC4$ utiliza ambos tipos. En las gráficas se identifican las gráficas mediante el número de particiones (dummy) no criticas en ejeución (n-D) y el intervalo de tiempo en el que se ejecutan.

7.1. Evaluación de escenarios

La figura 5 muestra la ejecución de $CPart$ en todas las situaciones. El eje X representa el timepo en milisegundos (aunque se leen los ciclos del procesador) mientras que el eje Y presenta las instrucciones ejecutadas.

En esta situación, $CPart$ termina su ejecución despues de 278ms si no hay interferencia. Cuando se ejecuta con 1,2 y 3 particioens en otros núcleos, su tiempo de ejecución es 409, 591 y 802 milisegundos, respectivamente.

La figura 6 muestra lo mismo que lo anterior pero con los controladores locales de los núcleos no críticos activados. La referencia puesta al C_1 cuando se ejecuta 1 partición no crítica no es alcanzada por lo que se ejecuta sin suspensión. Cuando se ejecutan 2 particiones, C1 y C2 se suspenden en los instantes 426 y 477 ms, respectivamente. Cuando son 3 no críticas, las suspensiones se producen en los instantes 413, 426 and 478 ms, respectivamente.

La figura 7 muestra la ejecución del escenario SC3

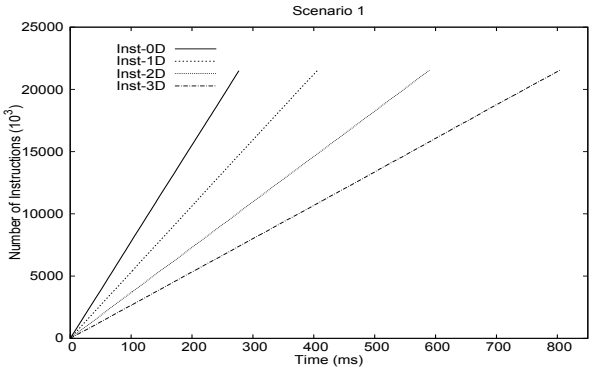

Figura 5: Todos los controladores están desactivados.

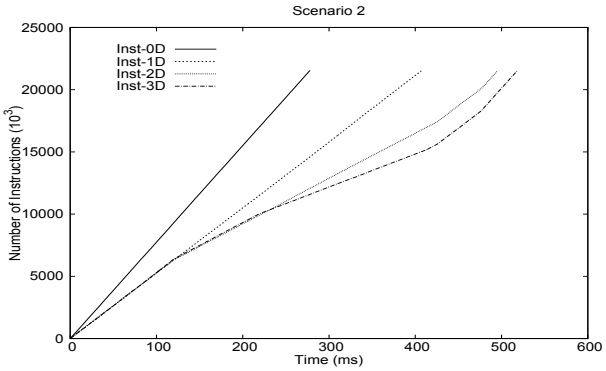

Figura 6: Los controladores locales están activados

con el controlador del núcleo crítico activado. En este caso, en el instante 205 ms suspende la actividad de C1 cuando sólo hay otro núcleo activo, en 245 ms cuando hay dos y en 252 ms cuando hay tres. En este caso, la suspensión es de todos los núcleos no críticos.

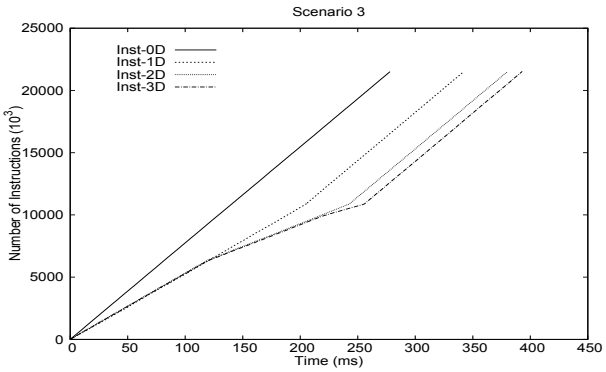

Figura 7: El Controlador crítico está activado

La figura 8 corresponde con el escenario SC4. En este caso, también es el controlador del núcleo crítico el que toma la decisión antes de los locales. En 206, 130 y 122 ms, se suspenden cuando hay 1, 2 o 3 núcleos no activos.

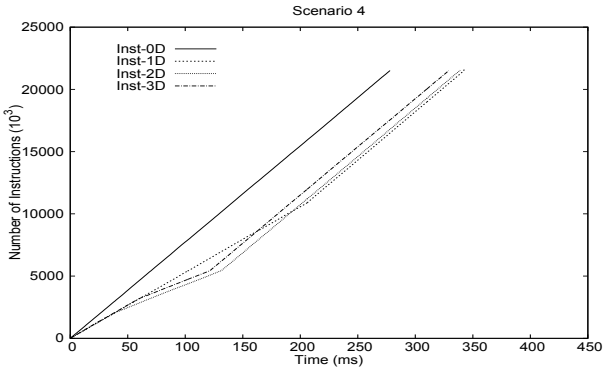

Figura 8: El controlador crítico y los locales activados

La conclusión de esta experimentación es que es posible controlar la ejecución de las actividades críticas en un sistema multi-núcleo. No obstante, es necesaria una identificación de estas actividades que permita determinar los parámetros apropiados de referencia para los controladores.

8. Conclusiones

En este artículo se ha propuesto un controlador implementado a nivel del hipervisor para controlar la ejecución de aplicaciones críticas en una plataforma multi núcleo. Se ha definido también el ámbito y el esquema del controlador, siendo este efectivo sólo cuando se están ejecutando las aplicaciones críticas. Las acciones de control sobre las aplicaciones críticas son simples: suspender la ejecución de los núcleos dedicados a las aplicaciones no críticas. Esto es debido a que las acciones a tomar por el hipervisor deben ser extremadamente sencillas de cara a facilitar una futura certificación y a evitar decisiones complejas que puedan incrementar la sobrecarga de tiempo de ejecución.

El trabajo futuro se centra en el ajuste del controlador y en cómo los parámetros de las particiones pueden tenerse en cuenta en el fichero de configuración del sistema para permitir al hipervisor definir las referencias más adecuadas para los controladores.

Agradecimientos

Este trabajo ha sido desarrollado en el marco de los proyectos de investigación M2C2 (TIN2014-56158-C4-01/02) y PROMETEOII/2014/031 (Generalitat Valenciana).

Referencias

[1] A. Abel and et al. Impact of resource sharing on performance and performance prediction: A survey. In Proc. 24th Int. Conf CONCUR 2013, Buenos Aires, Argentina, Aug. 27-30., pages 25–43, 2013.

[2] E. Bini and M. D. Natale. Optimal task rate selection in fixed priority systems. In Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS 2005), 6-8 Dec. 2005, Miami, FL, USA, pages 399–409, 2005.

[3] V. Brocal, P. Balbastre, R. Ballester, and I. Ripoll. Task period selection to minimize hyperperiod. In IEEE 16th Conference on Emerging Technologies & Factory Automation, ETFA 2011, Toulouse, France, Sept. 5-9, 2011, pages 1–4, 2011.

[4] V. Brocal, R. Masmano, I. Ripoll, A. Crespo, and P. Balbastre. Xoncrete: a scheduling tool for partitioned real-time systems. In Embedded Real-Time Software and Systems, 2010.

[5] A. Burns and R. I. Davis. Mixed criticality systems - a review, Ed. 2017. Univ. York. Internal Report.

[6] B. Cilku, A. Crespo, P. Puschner, J. Coronel, and S. Peiro. A tdma-based arbitration scheme for mixed-criticality multicore platforms. In Int. Conf. on Event-based Control, Communication and Signal Processing (EBCCSP), pp: 17-19, 2015. Krakow. Poland, 2015.

[7] E. Commision. Workshop on Mixed Criticality Systems, 2012. Brussels. cordis.europa.eu/fp7/ict/computing/home en.html.

[8] A. Crespo, P. Balbastre, J. Simo, and P. Albertos. Static scheduling generation for multicore partitioned systems. In Lecture Notes in Electrical Engineering. Vol. 376., pages 511–522, 2016. Int. Conf. on Information Science and Applications, ICISA 2016, Ho Chi Min, Vietnam, Feb 2016.

[9] A. Crespo, I. Ripoll, S. Peiró, and R. Masmano. Partitioned embedded architecture based on hypervisor: Then XtratuM approach. In EDCC, pages 67–72, 2010.

[10] DREAMS. Distributed real-time architecture for mixed criticality systems, 2013. EU FP7-ICT-610640 2013-17.

[11] I. Freescale Semiconductors. E6500rm, e6500 core reference manual - reference manual, Mon Jun 9 19:06:06 2014.

[12] S. Girbal, X. Jean, J. L. Rhun, D. Gracia Pérez, and M. Gatti. Deterministic platform

software for hard real-time systems using multi-core cots. In 34th Digital Avionics Systems Conference (DASC'2015), Prague, Czech Republic, 2015. Thales Research & Technology.

[13] A. Kritikakou, C. Rochange, M. Faugère, C. Pagetti, M. Roy, S. Girbal, and D. Gracia Pérez. Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In 22nd International Conference on Real-Time Networks and Systems, RTNS 2014, Versaille, France, October 8-10, 2014, page 139, 2014.

[14] T. Lundqvist and P. Stenström. Timing anomalies in dynamically scheduled microprocessors. In IEEE Real-Time Systems Symposium, pages 12–21, 1999.

[15] A. Marchand, P. Balbastre, I. Ripoll, R. Masmano, and A. Crespo. Memory resource management for real-time systems. In 19th Euromicro Conf. on Real-Time Systems, ECRTS'07, 4-6 July 2007, Pisa, Italy, pages 201–210, 2007.

[16] MultiPARTES. Multi-cores partitioning for trusted embedded systems, 2011. EU FP7-ICT-287702 2011-14.

[17] J. Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance, 1999.

[18] J. Windsor and K. Hjortnaes. Time and space partitioning in spacecraft avionics. Space Mission Challenges for Information Technology, 0:13–20, 2009.

[19] Y. Ye, R. West, J. Zhang, and Z. Cheng. MARACAS: A real-time multicore VCPU scheduling framework. In 2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016, pages 179–190, 2016.

[20] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In 19th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2013, Philadelphia, PA, USA, April 9-11, 2013, pages 55–64, 2013.