Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática industrial 16 (2019) 264-272

www.revista-riai.org

Aplicaciones Ada en Android con requisitos de tiempo real

Alejandro Pérez Ruiz *, Mario Aldea Rivas, Michael González Harbour

Grupo de Ingeniería Software y Tiempo Real, Departamento de Ingeniería Informática y Electrónica, Universidad de Cantabria, 39005, Santander, España

Resumen

Android es el sistema operativo más extendido en el ámbito de los dispositivos móviles. Su gran expansión y desarrollo ha provocado que exista un gran interés para utilizarlo en entornos con requisitos temporales. Este trabajo presenta un mecanismo para utilizar el lenguaje de programación Ada en el desarrollo de aplicaciones de tiempo real sobre Android. Ada es un lenguaje que ofrece soporte para aplicaciones con requerimientos temporales bajo la suposición de que la plataforma de ejecución proporciona las garantías necesarias en tiempos de respuesta. Para satisfacer estas garantías proponemos que las aplicaciones escritas en este lenguaje utilicen los mecanismos de aislamiento proporcionados por el sistema operativo Android/Linux, a través de los cuales es posible aislar uno o varios núcleos del procesador para ser usados exclusivamente por aplicaciones de tiempo real. Además, hemos estudiado los mecanismos que se encuentran disponibles en Android para compartir datos entre aplicaciones Ada con requisitos temporales y el resto de aplicaciones que se ejecutan en el mismo sistema.

Palabras Clave:
Sistemas operativos, Tiempo real, Sistemas operativos de tiempo real, Programas concurrentes Ada, Compiladores

Real-time Ada applications on Android

Abstract

Android is the most extended operating system in the field of smartphones. Its wide diffusion has caused a great interest in using it in real time environments. This paper presents a mechanism to use the Ada programming language for real-time applications on Android. Ada is a language that offers support for environments with real-time requirements under the assumption that the execution platform provides the necessary guarantees on response time. To accomplish these guarantees, we propose that applications written in this language use the isolation mechanisms provided by the Android/Linux operating system through which it is possible to isolate one or several processor cores to use them exclusively with real-time Ada applications. In addition, we have studied the available mechanisms in Android to share data between these isolated real-time Ada applications with other applications executing in the same system.

Keywords:
Operating systems, Real-time, Real-time operating systems, Ada tasking programs, Compilers

1. Introducción

Android ha estado en constante evolución desde la aparición del primer teléfono con este sistema operativo, habiendo experimentado un gran crecimiento respecto al número de características implementadas y tipos de dispositivos soportados. Su gran utilización en todo tipo de dispositivos móviles ha motivado su utilización en áreas para las que, en principio, no estaba destinado, como es el caso del control industrial, aplicaciones médicas o automoción entre otras. Las aplicaciones utilizadas en muchas de estas nuevas áreas tienen, en mayor o menor medida, requisitos de tiempo real.

Este sistema operativo ha sido desarrollado principalmente bajo la licencia Apache 2.0, aunque hay algunas partes que utilizan otro tipo de licencias, como por ejemplo los parches del kernel de Linux que poseen licencias GPLv2. Este tipo de software libre permite a desarrolladores y a la industria tener un profundo conocimiento sobre todas las características que ofrece Android. El kernel de Android está basado en el de Linux y por lo tanto es posible utilizar características avanzadas ofrecidas por este último.

*Autor para correspondencia: perezruiza@unican.es

To cite this article: Pérez A., Aldea M. and González M.. "Real-Time Ada applications on Android". 2019 Revista Iberoamericana de Automática e Informática Industrial 16, 264-272. https://doi.org/10.4995/riai.2019.10604

Android está formado por componentes software distribuidos en diferentes capas (ver Figura 1). La capa inferior corresponde al kernel de Linux, que proporciona las operaciones básicas del sistema, tales como gestión de la memoria, procesos o drivers. Sobre el kernel se sitúa otra capa que contiene una serie de librerías nativas escritas en C o C++ y compiladas para un hardware específico. Entre este conjunto de librerías, cabe destacar por su relevancia para el presente trabajo la librería Bionic, que es la versión modificada de la librería estándar C utilizada en Android. La librería Bionic está diseñada específicamente para este sistema operativo, pero tal y como veremos en este trabajo presenta importantes limitaciones cuando se utiliza con aplicaciones de tiempo real.

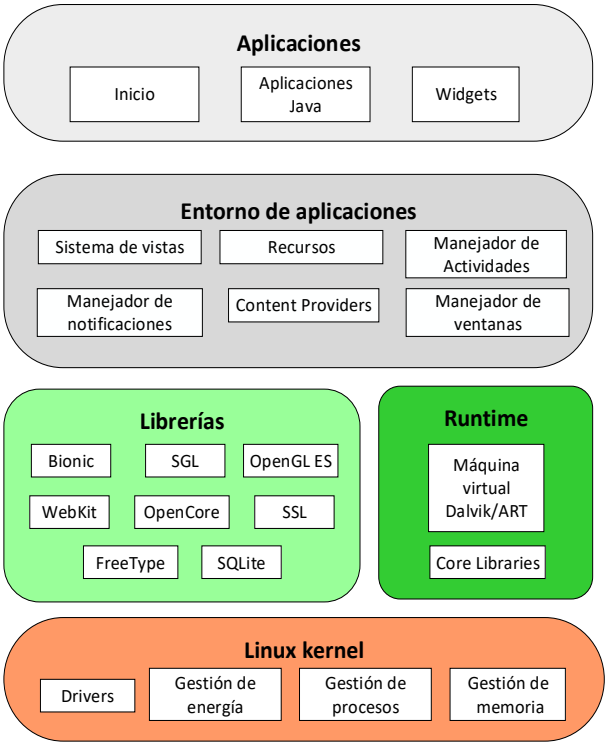

Figura 1: Arquitectura en capas software de Android.

Android permite a los desarrolladores codificar aplicaciones principalmente en Java. Sin embargo, es posible ejecutar aplicaciones escritas en cualquier otro lenguaje siempre y cuando se utilice el compilador adecuado para este sistema operativo. Además, el entorno de desarrollo de Android ofrece soporte para escribir código en C/C++ (Android Developers, Android NDK)

A pesar de que el lenguaje de programación C es uno de los más extendidos para el desarrollo de sistemas de tiempo real existen alternativas como Ada. El lenguaje Ada ofrece ventajas sobre C (Brosgol B.J., 2013) cuando desarrollamos programas concurrentes con requisitos de tiempo de real, ya que proporciona soporte nativo en el propio lenguaje para las características requeridas en los sistemas de tiempo real. No solo incluye características de tiempo real y concurrencia, sino que proporciona soporte para características adicionales que son útiles para sistemas embebidos, tales como programación por contrato, datos fuertemente tipados, cláusulas de representación, comprobaciones estáticas en la compilación y primitivas avanzadas para la sincronización. Todo ello contribuye a la capacidad de desarrollar software más fiable.

La ejecución en Android de aplicaciones nativas con requisitos temporales requiere limitar, en la mayor medida posible, las interferencias ejercidas sobre ellas por el sistema operativo y el resto de aplicaciones. Para lograr ese fin, en un trabajo previo (Perez A. et al, 2015), hemos desarrollado una técnica de aislamiento de uno o varios procesadores encargados de ejecutar las aplicaciones de tiempo real. El aislamiento se logra utilizando de forma apropiada un conjunto de servicios proporcionados por el kernel de Linux, sin necesidad de modificar ni la máquina virtual ni el propio kernel.

Las aplicaciones nativas con requisitos de tiempo real se ejecutan directamente sobre el kernel y las librerías nativas del sistema. Entre estas librerías se encuentra Bionic, la cual, como se ha comentado anteriormente, no es apropiada para ser utilizada por aplicaciones de tiempo real debido a sus limitaciones. En nuestra propuesta se reemplaza la librería Bionic por la implementación tradicional de la librería (glibc) que es utilizada en otros sistemas Linux.

Las principales aportaciones de este trabajo son: (a) desarrollo de un entorno para la compilación de aplicaciones Ada para Android; (b) identificación de las limitaciones para tiempo real de la librería Bionic y reemplazo con la librería glibc tradicional; (c) análisis de los mecanismos disponibles en Android para comunicar las aplicaciones nativas de tiempo real con el resto de las aplicaciones Java; y (d) aplicación de técnicas de aislamiento que permiten reducir drásticamente las interferencias del sistema sobre las aplicaciones de tiempo real.

En resumen, este trabajo describe un conjunto de herramientas y técnicas que permiten ejecutar en Android aplicaciones Ada de tiempo real laxo. Además, se proporcionan mecanismos para comunicarlas con otras aplicaciones Java ejecutadas en el sistema sin requisitos de tiempo real.

El presente trabajo se organiza del siguiente modo: La Sección 2 describe brevemente los trabajos relacionados. En la Sección 3 se describen los mecanismos de aislamiento, mientras que en la Sección 4 se exponen las limitaciones de la librería Bionic para su utilización con aplicaciones Ada y se explica el proceso para realizar su reemplazo por la librería tradicional glibc. En la Sección 5 se presenta un resumen de las distintas distribuciones del compilador GNAT y cuáles son los pasos para compilar un programa Ada en Android. También se exponen los tests que se han ejecutado para verificar la corrección del entorno. La Sección 6 describe brevemente los mecanismos para la comunicación de los programas Ada ejecutados en núcleos aislados con el resto de aplicaciones Java ejecutadas en el sistema. Finalmente, en la Sección 7 se presentan nuestras conclusiones y trabajos futuros.

2. Trabajos relacionados

En los últimos años, dentro de la comunidad científica y en algunos entornos empresariales ha existido un interés por conseguir tener compiladores Ada para Android (Ruiz J., 2013). Prueba de esto es que la compañía AdaCore ha implementado un entorno de desarrollo cruzado (AdaCore webpage) para procesadores ARM Cortex que ejecuten

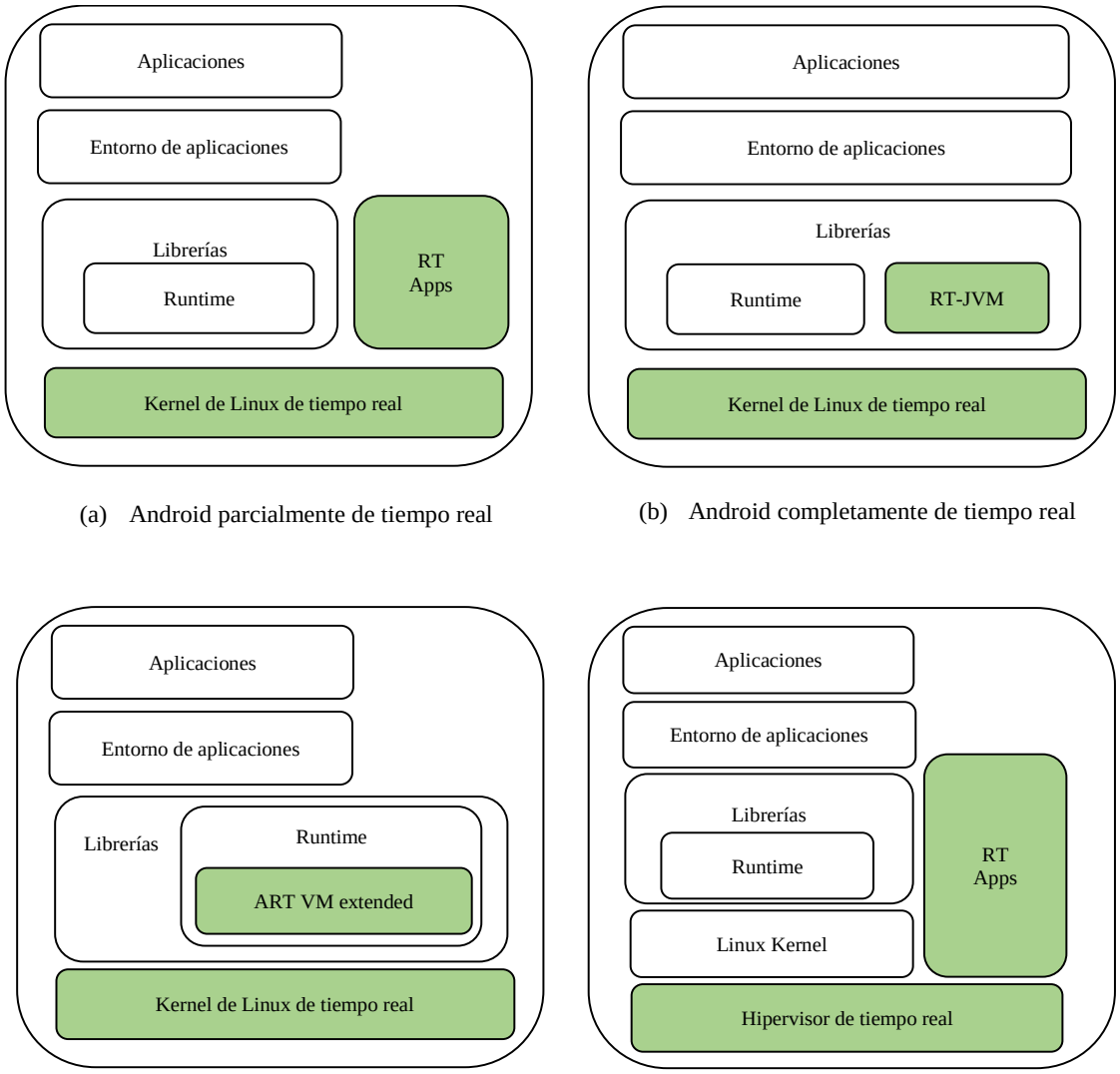

Figura 2: Posibles soluciones para adaptar Android a entornos de tiempo real.

Android. Este entorno de desarrollo se distribuye bajo una licencia de software comercial que permite crear programas que no estén sujetos a la licencia GPL, aunque esto implica que no es gratuito.

Como hemos mencionado, una de las razones de importancia para la utilización de Ada en Android es el hecho de que este lenguaje proporciona soporte para programar aplicaciones de tiempo real. Algunos estudios previos (Bhupinder S. and Madisetti V., 2010) (Perneel L et al, 2012) (Maia C. et al, 2010) (Perneel L et al, 2013) han analizado la viabilidad de ejecutar aplicaciones con requisitos temporales en Android. En todos ellos se llega a la conclusión de que la plataforma Android no es válida para entornos de tiempo real a menos que apliquemos algunos mecanismos adicionales o modificaciones en el sistema. Los problemas más importantes encontrados en los estudios previos son los siguientes:

- Android utiliza una modificación de la librería tradicional glibc que se denomina Bionic. Esta librería carece de aspectos claves para el desarrollo de aplicaciones de tiempo real, por ejemplo, no posee herencia de prioridad en los mutexes, algo que es esencial para evitar la inversión de prioridad en aplicaciones de tiempo real.

- La variabilidad en los tiempos de respuesta del kernel de Android y la máquina virtual de Java (ART, Android RunTime) no permite tener tiempos de respuesta acotados.

- El kernel de Linux utiliza por defecto un planificador llamado “Completely Fair Scheduler (CFS)”. Este planificador proporciona ventanas temporales de tiempo de ejecución en el procesador para cada uno de los procesos. Esto es incompatible con los requisitos temporales de las aplicaciones de tiempo real en las que se necesita planificar primero aquellos procesos más urgentes.

Debido a las limitaciones citadas anteriormente, un trabajo previo (Maia C. et al, 2010) muestra cuatro soluciones teóricas para adaptar el sistema operativo Android para requisitos de tiempo real. Estas cuatro soluciones se muestran en la Figura 2. La primera de las soluciones (Figura 2a) propone ejecutar

aplicaciones de tiempo real sobre un kernel de Linux con propiedades de tiempo real. La segunda solución (Figura 2b) consiste en añadir una máquina virtual Java (RT-JVM) con características de tiempo real que permitan ejecutar aplicaciones Java de tiempo real. La siguiente solución teórica (Figura 2c) modifica la máquina virtual Java de Android (ART VM) añadiendo características de tiempo real al mismo tiempo que se utiliza un kernel de Linux de tiempo real. Finalmente, la última solución (Figura 2d) utiliza un hipervisor de tiempo real para ejecutar aplicaciones propias de Android en paralelo a la ejecución de aplicaciones de tiempo real.

Algunos estudios han explorado la posibilidad de crear modificaciones y extensiones para Android siguiendo las soluciones teóricas previas para así obtener una respuesta temporal predecible del sistema. La solución ilustrada en la Figura 2a ha sido implementada en (Mauerer W. et al, 2102) aplicando una serie de parches (RT_PREEMPT) en el kernel de Android/Linux para así obtener características de tiempo real. Además, para permitir una comunicación entre las aplicaciones Java y las aplicaciones de tiempo real se ha desarrollado un canal de comunicación y sincronización entre los dos tipos de aplicaciones.

Otro trabajo (Kalkov I. et al, 2012) (Kalkov I. et al, 2014) (Kalkov I. et al, 2015) ha realizado una adaptación de Android basándose en la solución propuesta en la Figura 2c. En este caso se modifica el kernel de Android/Linux con la aplicación del parche RT_PREEMPT y además se añaden modificaciones a otros componentes de Android como el recolector de basura en la máquina virtual de Java. También se modifica la clase Service para permitir el cambio de prioridades en las aplicaciones Java y el módulo llamado Binder ha sido adaptado para añadir herencia de prioridad en las llamadas remotas a procedimientos.

Hay otro estudio (Yan Y. et al, 2013) (Yan Y. et al, 2014) (Yan Y. et al, 2017) que ha optado por una solución diferente a las 4 propuestas en la Figura 2. En este caso, se ha creado un prototipo llamado RTDroid para que sea compatible con aplicaciones escritas para la plataforma Android. En esta solución se utiliza una máquina virtual Java con características para tiempo real (Fiji-VM) sobre un sistema operativo de tiempo real (Linux-RT o RTEMS). En este caso la API original de Android ha sido recreada paso a paso para permitir la ejecución de aplicaciones Android. Podemos decir que de algún modo esta propuesta está inspirada por la solución expuesta en la Figura 2b, aunque han decidido construir todo el sistema desde cero.

Cualquier solución basada en las propuestas de la Figura 2 implica un profundo proceso de adaptación del sistema operativo; ya sea modificando el kernel de Android/Linux o adaptando la máquina virtual de Java para llegar a tener características de tiempo real en el sistema. Esta adaptación es compleja y requiere de frecuentes actualizaciones para mantener la compatibilidad con las nuevas versiones de Android que van apareciendo. Por esta razón, en un trabajo previo (Perez A. et al, 2015) hemos descrito una solución que aprovecha las ventajas que ofrecen las arquitecturas multinúcleo para aislar un núcleo del procesador y así utilizarlo exclusivamente para ejecutar aplicaciones de tiempo real laxo sobre él. Esta solución no requiere ninguna modificación en el sistema operativo y es muy portable a través de los distintos tipos de dispositivos que se distribuyen con Android. En la siguiente sección se describe brevemente esta solución.

3. Aislamiento de CPU para ejecutar aplicaciones de tiempo real

En nuestro trabajo previo (Perez A. et al, 2015) se ha presentado un mecanismo para ejecutar aplicaciones de tiempo real laxo en dispositivos Android. Las aplicaciones con requerimientos temporales se ejecutan directamente sobre el kernel de Linux en un núcleo aislado del procesador sin necesidad de modificar el código del kernel o la plataforma. Esta solución puede ser usada en cualquier dispositivo Android con un procesador multinúcleo y un kernel Linux versión 2.6 o superior. Las aplicaciones de tiempo real deben ejecutarse haciendo uso de las prioridades de tiempo real ofrecidas por el kernel de Android/Linux (política de planificación SCHED_FIFO).

Por defecto las versiones superiores o iguales a la 2.6 del kernel de Linux tienen activada una opción que hace que la mayoría del código del kernel sea expulsable, y por lo tanto en cualquier momento durante la ejecución de este código puede producirse una expulsión, excepto en los manejadores de interrupción y las regiones protegidas con spinlocks. Además, las políticas de planificación de tiempo real (SCHED_FIFO y SCHED_RR) están incluidas en estas versiones del kernel.

A pesar de todas las características ofrecidas por el kernel de Android/Linux hay algunos inconvenientes que debemos resolver si queremos tener una predictibilidad temporal razonable en las aplicaciones con requisitos temporales:

- Interferencias entre aplicaciones de tiempo real y otras aplicaciones que pueden estar ejecutándose en el sistema.
- Efectos de los manejadores de interrupciones sobre las aplicaciones de tiempo real.
- Cambios dinámicos en la frecuencia del procesador y apagado automático de algunos de los núcleos del procesador para el ahorro de energía.
- Limitaciones de la librería Bionic para aplicaciones con requisitos de tiempo real.

Linux proporciona algunos mecanismos para aislar CPUs de la actividad general del planificador. El más adecuado para nuestra propuesta es el denominado “cpuset”. Esta funcionalidad proporcionada por el kernel no está activada por defecto en los dispositivos Android, lo cual significa que debemos recompilar el kernel para proceder a su activación. El mecanismo “cpuset” permite restringir los procesadores y los recursos de memoria asignados a un conjunto de procesos. Cuando el sistema operativo se inicializa, todos los procesos pertenecen a un solo “cpuset”. Si tenemos los suficientes privilegios podemos crear un nuevo cpuset y mover procesos de uno a otro.

De tal modo, podemos asignar todos los procesos del sistema a un cpuset específico y al mismo tiempo crear otro cpuset donde solo se encuentren los procesos de tiempo real.

La implementación del mecanismo de cpuset establece que por defecto todos los nuevos procesos que se creen serán asignados al mismo cpuset al que pertenece su proceso padre, excepto para los procesos hijos de kthreadd. Este demonio se

ejecuta en el espacio del kernel y es utilizado por el sistema para crear nuevos hilos del kernel. Por lo tanto, no podemos garantizar que algunos hilos del kernel no se ejecuten en uno de nuestros núcleos aislados. A pesar de esto, en numerosos experimentos llevados a cabo hemos confirmado que en raras ocasiones ocurre y tampoco tiene un gran impacto en los tiempos de ejecución.

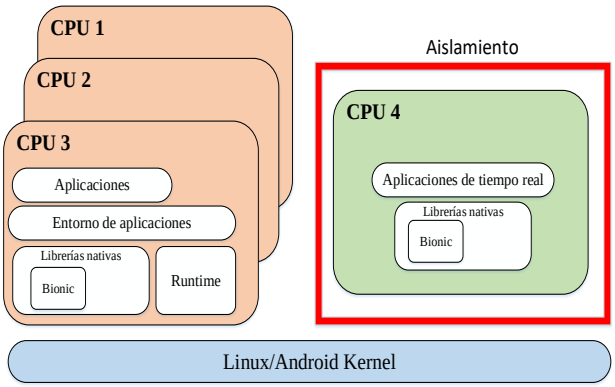

Figura 3: Solución para ejecutar aplicaciones de tiempo real en Android.

Con este mecanismo de aislamiento no podemos evitar la llegada de interrupciones en alguno de los núcleos aislados, pero desde la versión 2.4 del kernel de Linux existe la posibilidad de asignar ciertas interrupciones a un núcleo del procesador (o a varios núcleos). Esta funcionalidad se denomina en inglés *SMP IRQ affinity* y permite especificar qué núcleos manejarán las distintas interrupciones que ocurren en el sistema. Por cada interrupción hay un directorio en el sistema donde hay un fichero que permite establecer la afinidad modificando una máscara. No todas las interrupciones son enmascarables, por ejemplo, aquellas producidas entre los núcleos del procesador (IPI-interprocessor interrupts). Como en el caso de los hilos del kernel hemos realizado numerosos tests que muestran que el impacto de este tipo de interrupciones es limitado.

Para alcanzar un mayor grado de predictibilidad en los tiempos de ejecución es necesario fijar la frecuencia de los núcleos del procesador destinados a la ejecución de aplicaciones con requisitos de tiempo real. Además, algunos dispositivos Android utilizan demonios para apagar los núcleos del procesador que no están siendo usados con el objetivo de ahorrar energía. Para prevenir que esto ocurra sobre los núcleos aislados, se deben desactivar este tipo de demonios.

Aplicando todos los mecanismos descritos en nuestro trabajo previo (Perez A. et al, 2015) hemos determinado una mejora sustancial en las interferencias sobre una tarea que se ejecuta en un núcleo aislado, si comparamos dichas interferenciascon las que se producen en la ejecución en un núcleo no aislado. En nuestros experimentos hemos hallado que en las ejecuciones de una tarea de tiempo real sobre un núcleo no aislado se pueden llegar a observar interferencias que alcanzan los 1.5 segundos, mientras que en un núcleo aislado aplicando los mecanismos descritos previamente la mayor interferencia observada es menor de 100 microsegundos.

En la Figura 4 se ilustran los tiempos de respuesta de peor caso para la ejecución de cuatro tests. En dichos tests se mide el tiempo de respuesta de una instrucción de adición de un entero cuando se ejecuta sobre un núcleo no aislado frente a la ejecución sobre un núcleo aislado siguiendo la solución propuesta en nuestro trabajo previo (Perez A. et al, 2015). De este modo se puede observar como hay una mejora significativa en los tiempos de respuesta cuando estamos ejecutando sobre un núcleo aislado debido a que se reduce el número de interferencias.

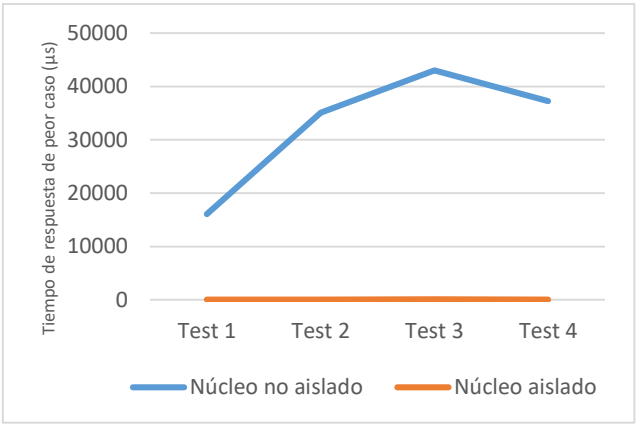

Figura 4: Tiempos de respuesta de peor caso para la ejecución de diferentes tests sobre un núcleo aislado frente a un núcleo no aislado.

4. ¿Es la librería Bionic adecuada para Ada y tiempo real?

Aunque las aplicaciones Android se escriben principalmente en el lenguaje de programación Java, es posible ejecutar código escrito en otros lenguajes para solventar algunas limitaciones de Java, como por ejemplo la penalización del rendimiento de la máquina virtual. Android proporciona un kit de desarrollo nativo (Native Development Kit, NDK) que soporta C/C++.

El kit de desarrollo nativo de Android (NDK) utiliza la librería Bionic. Esta librería ha sido desarrollada por Google bajo la licencia BSD para así conseguir que las aplicaciones Android estén libres del efecto “copyleft” y de este modo permitir la creación de código propietario en el espacio de usuario. Además, la librería Bionic es mucho más pequeña que la librería tradicional glibc y está diseñada para ser eficiente y así operar correctamente en CPUs con frecuencias bajas.

Tanto la librería tradicional glibc como la librería Bionic proporcionan y definen las llamadas al sistema y otras funciones básicas que utilizan casi todos los programas ejecutados en el sistema. Entre dichas funciones se encuentra la librería Pthread (Bradford N. et al, 1996) que proporciona soporte para la gestión de hilos y está incluida en la implementación tradicional de glibc. Está basada en el estándar POSIX definido por IEEE para especificar una API portable entre los diferentes sistemas operativos. Sin embargo, como ya hemos mencionado, la librería Bionic ha realizado importantes cambios en su implementación.

Para determinar si las aplicaciones Ada de tiempo real pueden ser implementadas sobre la librería Bionic hemos procedido a comprobar si esta librería proporciona las funciones requeridas por la librería de tiempo de ejecución (Run-Time System) del compilador libre gnat que utilizaremos en este trabajo. Se ha detectado que algunas de las funciones esenciales relativas a los protocolos de los mutexes no están

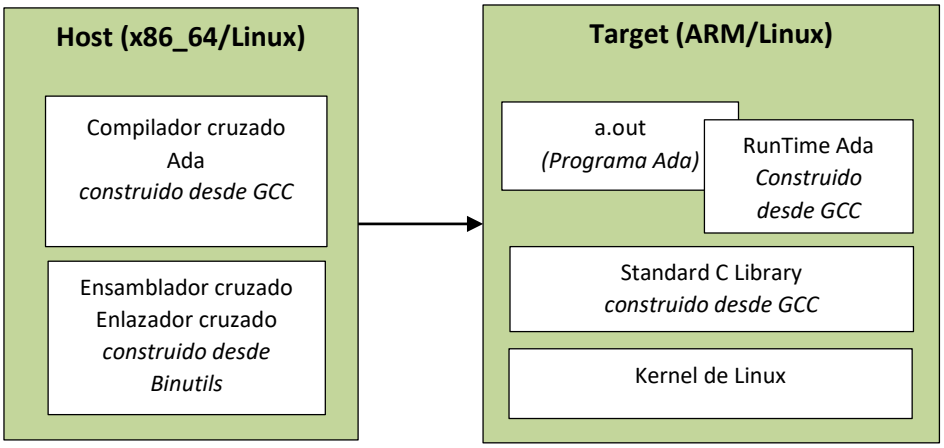

Figura 5: Proceso de compilación cruzada y ejecución de un programa Ada con host x86_64 Linux en un target ARM/Linux

implementadas. Las siguientes funciones y símbolos no están disponibles en la librería Bionic:

- Mutexes:
  - pthread_mutexattr_setprotocol
  - pthread_mutexattr_setrioceiling
  - pthread_mutexattr_getprioceiling
  - pthread_mutexattr_setprioceiling
- Prioridades:
  - pthread_setschedprio
  - Símbolo PTHREAD_EXPLICIT_SCHED
- Señales:
  - Sigwaitinfo
  - Sigqueue
  - Sigtimedwait

Estas limitaciones hacen que el uso de la librería Bionic en la compilación y el proceso de ejecución de la librería de ejecución (Run-Time System) de Ada no sea adecuado.

4.1. Glibc en Android

La primera solución para solventar las limitaciones descritas previamente podría ser la modificación del código de la librería Bionic para añadir soporte para las funciones no implementadas. Esto implicaría un gran esfuerzo y una constante adaptación para las futuras nuevas versiones de la librería. Por lo tanto, hemos decidido optar por una solución más portable y fácil que consiste en utilizar la librería tradicional glibc en Android.

Para utilizar la librería glibc en Android es necesario usar una librería compilada y adaptada para arquitecturas ARM en sistemas operativos Linux, debido a que nuestros experimentos se llevan a cabo en un smartphone Nexus 5 que posee esa arquitectura hardware. El modo más directo e inmediato sería utilizar un compilador cruzado ARM/Linux con la opción de compilación estática seleccionada, ya que de este modo se obtendría un código ejecutable que incorporaría todas las funciones de las librerías usadas por el programa compilado. Sin embargo, esto nos limitaría exclusivamente a la ejecución de programas enlazados estáticamente. Para aprovechar las ventajas de ahorro de memoria del enlazado dinámico hemos decidido usar la librería glibc en Android con enlazado dinámico. Para ello, usando un compilador cruzado, debemos seguir los siguientes pasos:

1. Copiar todas las librerías dinámicas al dispositivo Android donde queremos ejecutar de manera nativa las aplicaciones Ada. Las librerías dinámicas pueden ser obtenidas de las proporcionadas por el compilador cruzado. Normalmente en distribuciones de Linux se ubican en la ruta /usr/compilador-cruzado/lib.

2. Durante la compilación es necesario indicar el enlazador dinámico que se utilizará y la ruta de las librerías dinámicas en el dispositivo Android. Esto se hace mediante las opciones de compilación `--dynamic-linker` y `--rpath`.

Teniendo en cuenta que el kernel de Android no sigue la rama principal de desarrollo del kernel de Linux sería arriesgado decir que la librería glibc puede ser usada en este sistema operativo sin previamente haber ejecutado alguna serie de tests. Hemos adaptado los tests funcionales que están disponibles en el conjunto llamado “Open POSIX Test Suite” (Open POSIX Test Suite from A GPL Open Source Project), que incluye tests funcionales escritos en C para probar hilos, semáforos, temporizadores, variables condicionales, colas de mensajes y protocolos de herencia de prioridad con mutexes. Tal y como describimos en un trabajo previo (Perez A. et al, 2016), todos estos tests han obtenido resultados satisfactorios cuando se han ejecutado en Android usando la librería glibc tradicional. Por lo tanto, podemos afirmar que es posible utilizar la librería glibc en Android.

En la siguiente sección se describirá el proceso para construir y utilizar un compilador GNAT en un dispositivo Android.

5. GNAT para Android

GNAT es el compilador libre GNU Ada y es el único que soporta todos los anexos opcionales del estándar del lenguaje. Algunos autores del compilador GNAT crearon la compañía AdaCore para proporcionar soporte continuo en el desarrollo del compilador y también ofrecer un soporte profesional.

Actualmente hay tres distribuciones principales de GNATque se describen brevemente a continuación:

- Edición GNAT GPL: Está disponible de manera libre, está mantenida por la compañía AdaCore y no se distribuye para desarrollo profesional. Si se quiere distribuir un programa en código binario enlazado con la librería de ejecución de Ada (run-time system) se debe licenciar dicho programa con una licencia compatible con GPL, lo que implica distribuir el código fuente.

- GNAT Pro: Es una versión profesional de GNAT desarrollada por AdaCore, de pago, que utiliza una licencia de software comercial que no obliga a los creadores de aplicaciones a distribuir su código fuente.

- GNAT FSF: Esta versión está incorporada en el popular sistema de compilación GCC, perteneciente a la "Free Software Foundation". Se distribuyen solamente las fuentes. Se distribuye bajo una licencia GPL modificada que permite el desarrollo de software propietario.

Tomando en cuenta las características de cada distribución, hemos decidido descargar las fuentes de GNAT FSF disponibles en GCC para generar un compilador cruzado GNAT específico para nuestro entorno de desarrollo. Hemos seleccionado la versión GNAT FSF debido a su licencia y porque permite construir diferentes versiones de GNAT adaptables a diferentes tipos de dispositivos.

Nuestro objetivo ha sido generar un compilador cruzado GNAT FSF para un entorno x86_64/Linux (host) que nos permita compilar programas para un dispositivo ARM/Linux (target). Para crear un compilador cruzado como el ilustrado en la Figura 5 es necesario generar previamente algunas herramientas y librerías. El proceso para crear todas las partes mostradas en la Figura 5 puede ser automatizado mediante la creación de un script que permita construir cada una de las partes necesarias.

El compilador cruzado se ha construido para la arquitectura ARM/Linux porque no podía ser generado directamente para Android. Sin embargo, como Android está basado en el kernel de Linux, tal y como hemos explicado en la subsección 4.1, es posible ejecutar binarios generados con el anterior compilador cruzado si usamos la librería glibc en vez de la librería Bionic. Para utilizar la librería glibc en Android con un compilador GNAT debemos copiarla en nuestro dispositivo Android (target) e indicar el enlazador dinámico correcto a utilizar y determinar la ruta de las librerías dinámicas dentro del dispositivo. A modo de ejemplo mostramos una instrucción de compilación siguiendo lo descrito anteriormente:

```
arm-unknown-linux-gnueabi-gnatmake
hello_world.adb -fPIE -pie -largs -Wl,--dynamic-
linker=/data/local/libs/ld-linux.so.3 -Wl,--
rpath=/data/local/libs
```

Para verificar que el proceso previo se puede llevar a cabo con todos los programas Ada sin que exista ningún problema, en la siguiente subsección vamos a describir los tests que se han ejecutado.

5.1. Probando Ada en Android

El conjunto de tests denominado "The Ada Conformity Assesment Test Suite (ACATS)" (Brukardt R.L., Ada Conformity Assessment Test Suite) se utiliza para verificar que los compiladores Ada son conformes con el estándar Ada. Tal y como ya se ha descrito en un trabajo previo (Eilers D., et al, 2011) los ACATS no están integrados de forma nativa en ningún entorno de test moderno y están diseñados para que los usuarios de estos tests creen scripts personalizados para su compilación, ejecución y análisis de resultados.

Existe un script que forma parte del conjunto de tests del código fuente del GCC (localizado en gcc/testsuite/ada/acats). Este script por defecto está diseñado para usarse con un compilador nativo. Sin embargo lo hemos adaptado para que se pueda utilizar con el compilador cruzado de GNAT para ARM/Linux en un dispositivo Android[*]. El script original se ha dividido en dos partes; la primera de las partes (ejecutada en el dispositivo anfitrión) realiza la compilación de todos los tests mediante el compilador cruzado GNAT y la segunda parte se lanza directamente en el dispositivo Android para llevar a cabo la ejecución de cada uno de los tests compilados previamente.

Este conjunto de tests está compuesto por más de 2300 tests individuales. Se han ejecutado en un Nexus 5 con Android 6.0 donde todos los tests han pasado satisfactoriamente. Después de verificar que el compilador cruzado GNAT FSF para ARM/Linux puede ser usado en Android, en la siguiente sección hablaremos sobre los mecanismos que se pueden utilizar en este sistema operativo para compartir datos entre programas escritos en diferentes lenguajes (por ejemplo Ada y Java) que se ejecutan respectivamente en un núcleo aislado y en núcleos no aislados.

6. Compartiendo datos entre aplicaciones

Hemos demostrado que es posible ejecutar aplicaciones Ada con requisitos de tiempo laxos en Android, pero sería deseable tener mecanismos para compartir datos entre las aplicaciones nativas que se ejecutan en un núcleo aislado (por ejemplo Ada o C) y las aplicaciones Java convencionales de Android. Por lo tanto, a continuación describiremos las alternativas para compartir memoria entre procesos en Android.

6.1. Anonymus Shared Memory (Ashmem)

Android por defecto no soporta las interfaces POSIX para alojar memoria compartida en el sistema. El mecanismoque se utiliza en Android y que ha sido implementado por Google exclusivamente para este sistema operativo se denomina Anonymus Shared Memory (Ashmem). Este mecanismo no funciona como la memoria compartida en un sistema Linux. Las regiones de memoria compartida se crean bajo una entrada en un fichero en /dev/ashmem. Posteriormentes ese fichero es

[*] El script adaptado se encuentra disponible en: http://www.istr.unican.es/androidrt

borrado pero el correspondiente i-nodo sigue existiendo debido que hay un descriptor de fichero abierto sobre él.

Se pueden crear diferentes regiones ashmem con el mismo nombre y todas ellas aparecerán como /dev/ashmem/<name>(deleted), pero cada una corresponderá a un i-nodo diferente, y por lo tanto a una región de memoria diferente. Por esta razón, el nombre de una región de memoria de este tipo es solo para propósitos de depuración. No es posible abrir una región utilizando su nombre. Una región de memoria ashmem se borra automáticamente cuando el último descriptor de fichero que apunta a ella se cierra.

Además de todo lo mencionado anteriormente, algunos autores (Damschen M., 2012) desaconsejan el uso de este mecanismo debido a la escasa documentación que existe sobre él, y consideran que podrían aparecer cambios en su diseño o implementación que afectasen en un futuro a su comportamiento.

6.2. Ficheros mapeados en memoria

Un mecanismo para compartir memoria entre procesos consiste en utilizar un fichero. Para mejorar el rendimiento, el fichero se puede mapear en la memoria virtual. El fichero mapeado se puede compartir entre diferentes procesos y será accesible a través de un puntero aritmético. Hemos realizado algunos tests para medir el rendimiento de este mecanismo y después de analizar los resultados hemos detectado que cada 4096 bytes escritos en el segmento de memoria mapeado se produce un volcado de los datos al disco. Esto provoca un aumento significativo en los tiempos de respuesta. En la Tabla 1 se pueden observar los tiempos de respuesta obtenidos cuando se comparte un valor entero utilizando un fichero mapeado en memoria entre dos procesos que se encontraban en ejecución en un núcleo aislado del procesador. El peor tiempo de respuesta es debido al volcado que se produce cada 1024 enteros.

Tabla 1: Tiempos de respuesta obtenidos en 10000 lecturas de un entero utilizando un fichero compartido mapeado en memoria desde un núcleo aislado del procesador.

Fichero mapeado en memoria	
Primera lectura	2879 ns
Tiempo medio	501 ns
Peor tiempo de respuesta	14115 ns

6.3. Memoria compartida de POSIX

La interfaz POSIX proporciona una serie de funciones que nos permiten tener segmentos de memoria compartida entre procesos. Los objetos compartidos se pueden mapear concurrentemente en un espacio de direcciones de diversos procesos. Ninguna de estas funciones se encuentra en la librería Bionic pero sí que se encuentran dentro de la librería tradicional glibc, la cual puede utilizarse en Android siguiendo las directrices descritas en la subsección 4.1. Para utilizar la función que abre un segmento de memoria compartida (shm_open) se debe deshabilitar la capa de seguridad llamada SELinux (Security-Enhanced Linux in Android) que Android tiene activada por defecto en sus últimas versiones. SELinux proporciona una protección en el acceso de las aplicaciones Java a los ficheros alojados en el directorio /dev. Desde Android 4.2, SELinux se usa para definir los límites de las aplicaciones de Android, y por lo tanto si lo desactivamos temporalmente podemos comprometer la seguridad del sistema en su conjunto.

6.4. tmpfs

El sistema de ficheros temporales (tmpfs) es un sistema de ficheros que permite almacenar ficheros en memoria virtual. Está creado para parecer un sistema de ficheros tradicional montado en disco, pero realmente se almacena en la memoria volátil del sistema. Debido a que los ficheros montados con tmpfs se mantienen en la cache del kernel no pueden ser retirados de la memoria volátil. Además el acceso a ellos es significativamente más rápido que el acceso a disco.

Este mecanismo es el más adecuado para utilizar entre aplicaciones nativas de tiempo real (por ejemplo, Ada o C) y las aplicaciones Java porque está soportado nativamente en Android. Hemos realizado diversos tests para medir los tiempos de respuesta para acceder a un número entero utilizando este mecanismo y hemos obtenido unos tiempos de respuesta que oscilan entre los 340 y 570 nanosegundos en nuestro entorno de pruebas (Nexus 5 con Android 6.0). En la Tabla 2 se ilustran los tiempos de respuesta que se obtienen al compartir un valor entero entre dos procesos que se ejecutan en un núcleo aislado del procesador. Por lo tanto, podemos concluir que tiene un rendimiento adecuado para nuestra propuesta. Además, también es posible utilizarlo junto a algunas librerías existentes (Liblfds, “a portable, license-free, lock-free data structure library” escrita en C) para conseguir una sincronización sin bloqueos (lock-free).

Tabla 2: Tiempos de respuesta obtenidos en 1000 lecturas de un entero utilizando el mecanismo tmpfs desde un núcleo aislado del procesador.

tmpfs	
Primera lectura	573 ns
Tiempo medio	364 ns
Peor tiempo de respuesta	573 ns

7. Conclusiones y trabajos futuros

En este trabajo se ha presentado un estudio sobre la viabilidad de la ejecución de aplicaciones Ada de tiempo real laxo en el sistema operativo Android. Utilizamos los mecanismos descritos en un trabajo previo (Perez A. et al, 2015) para aislar un núcleo del procesador donde ejecutar las aplicaciones Ada. Hemos mostrado como los programas Ada pueden compilarse con un compilador cruzado ARM/Linux utilizando la librería tradicional glibc en vez de la librería Bionic de Android. Además, hemos adaptado un script existente para ejecutar el conjunto de tests denominado Ada Conformity Assesment Test Suite (ACATS) en dispositivos Android, y hemos verificado que estos han pasado satisfactoriamente.

Las aplicaciones Ada pueden comunicarse con las aplicaciones convencionales (por ejemplo, Java) utilizando el mecanismo tmpfs y sincronizarse mediante algunas de las implementaciones existentes de estructuras de datos lock-free.

Nuestro siguiente paso será estudiar los mecanismos de sincronización más apropiados entre las aplicaciones de tiempo real Ada y el resto de aplicaciones Android para así

poder aplicar todo lo propuesto en este trabajo en un caso real de estudio que nos permita una evaluación más exhaustiva.

Agradecimientos

Este trabajo ha sido financiado en parte por el Gobierno de España en el proyecto TIN2014-56158-C4-2-P (M2C2).

Referencias

AdaCore webpage. Disponible en: https://www.adacore.com/press/gnat-pro-7-2-for-android [consultada el 10-agos-2018]

Android Developers., Android NDK Disponible en: http://developer.android.com/tools/sdk/ndk/index.html [consultada el 10-agos-2018]

Brosgol B.J., 2013. Ada and Java: Real-Time advantages. Embedded Systems Programming.

Bhupinder S. and Madisetti V., 2010. Realiable Real-Time Applications on Android OS. Whitepaper.

Bradford N., Buttlar D., and Farrell J., 1996. Pthreads programming: A POSIX standard for better multiprocessing. O'Reilly Media, Inc.

Brukardt R.L. Ada Conformity Assessment Test Suite (ACATS), Disponible en: http://www.ada-auth.org/acats.html [consultada el 10-agos-2018]

Damschen M., 2012. Concurrent shared memory access for Android applications and real-time processes: Bachelor Thesis, Universität Paderborn.

Eilers D., and Koskinen T., 2011. Adapting ACATS to the Ahven Testing Framework. Reliable Software Technologies – Ada-Europe 2011, Lecture Notes in Computer Science, Vol. 6652/2011, pp. 75-88.

Kalkov I., Franke D., Schommer J. F., and Kowalewski S., 2012. A real-time extension to the Android platform. Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems, JTRES, pp 105–114, New York.

Kalkov, I., Gurghian, A., and Kowalewski, S., 2014. "Predictable Broadcasting of Parallel Intents in Real-Time Android", in Proc. 12th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), pp. 57-66.

Kalkov I., Gurghian A, and Kowalewski S., 2015. Priority Inheritance during Remote Procedure Calls in Real-Time Android using Extended Binder Framework. Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES).

Liblfds, a portable, license-free, lock-free data structure library written in C. Disponible en: https://liblfds.org/ [consultada el 10-agos-2018]

Maia C., Nogueira L. and Pinho L. M.., 2010. Evaulating Android OS for Embedded Real-Time Systems. In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2010, pages 63- 70, Brussels, Belgium.

Mauerer W., Hillier G., Sawallisch J., Hönick S., and Oberthür S., 2012. Real-time android: deterministic ease of use. Proceedings of the Embedded Linux Conference Europe (ELCE '12).

Open POSIX Test Suite from A GPL Open Source Project. Disponible en: http://posixtest.sourceforge.net/ [consultada el 10-agos-2018]

Perez Ruiz A., Aldea M., Gonzalez Harbour M., 2015. CPU Isolation on the Android OS for running Real-Time Applications. Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES).

Perez Ruiz A., Aldea M., Gonzalez Harbour M., 2016. Servicios de tiempo real en Android. Servicios de tiempo real en Android. V Simposio de Sistemas de Tiempo Real in the V Congreso Español de Informática (CEDI).

Perneel L., Fayyad-Kazan H. and Timmerman M., 2012. Can Android be used for Real-Time purposes? International Conference on Computer Systems and Industrial Informatics, ICCSII ’12, pages 1–6.

Perneel L., Fayyad-Kazan H., and Timmerman M., 2013. Android and Real-Time Applications: Take Care!. Journal of Emerging Trends in Computing and Information Sciences, Volume 4, Special Issue ICSSII

Ruiz J., 2013. Ada on Android. FOSDEM, Brussels.

Security-Enhanced Linux in Android. Disponible en: https://source.android.com/security/selinux [consultada el 17-oct-2018].

Yan Y., Cosgrove S., Anand V., Kulkarni A., Konduri S. H. and Ko S. Y., Ziarek L., 2014. Real-Time Android with RTDroid. Proceedings of the 12th International Conference on Mobile Systems, Applications, and Services (MobiSys).

Yan Y., Dantu K., Ko S., Vitek J. and Ziarek L., 2017. Making Android Run on Time. Real-Time and Embedded Technology and Applications Symposium (RTAS'17).

Yan Y., Konduri S. H., Kulkarni A., Anand V. and Ko S. Y., Ziarek L., 2013. RTDroid: A Design for Real-Time Android. Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES).