Servicios de tiempo real en el sistema operativo Android

Alejandro Pérez Ruiz, Mario Aldea Rivas y Michael González Harbour

Grupo de Ingeniería Software y Tiempo Real, Universidad de Cantabria

{perezruiza,aldeam,mgh}@unican.es

Resumen. Debido a la gran expansión y crecimiento de Android el interés por utilizar este sistema operativo en entornos de tiempo real es cada vez mayor. En este trabajo se describen una serie de mecanismos proporcionados por el sistema operativo Android/Linux mediante los cuales es posible aislar uno o más núcleos de un multiprocesador simétrico para ser utilizados exclusivamente por tareas con requisitos temporales. Gracias a los mecanismos de aislamiento, la tasa de interferencias sufridas por las tareas con requisitos temporales respecto a otras tareas o aplicaciones que se ejecutan en el sistema operativo es muy baja. Un segundo aspecto en el que se mejora el comportamiento de tiempo real del sistema operativo Android está relacionado con las limitaciones para tiempo real de la librería *bionic* (modificación de *glibc* para Android). Para solventar estas limitaciones se ha utilizado la librería *glibc* incluida en la distribución estándar de Linux. Se han realizado una serie de tests que demuestran que la librería tradicional funciona correctamente en Android. Asimismo se ha llevado a cabo la caracterización temporal de Android/glibc para las funciones más relevantes de POSIX para tiempo real observándose que la respuesta temporal del sistema es apropiada para aplicaciones de tiempo real laxo.

Palabras clave: Android, tiempo real, sistemas operativos, multinúcleo

1 Introducción

Desde que apareció el primer móvil con Android este sistema operativo ha estado en constante evolución tanto en características como en soporte para distintas clases de dispositivos. Debido a ello existe un gran interés en utilizar este sistema operativo en entornos que poseen requerimientos temporales. Existen ventajas cuando utilizamos Android para propósitos de tiempo real, ya que es un sistema adaptado y optimizado para utilizar en dispositivos que sean eficientes energéticamente y de recursos limitados, y además es un sistema extendido por un gran número de dispositivos. Lo habitual es que los procesadores modernos usados con dispositivos Android tiendan a utilizar múltiples núcleos.

Este sistema operativo se desarrolla principalmente bajo licencia Apache 2.0, aunque hay determinadas partes que son distribuidas con otros tipos de licencias, por ejemplo, algunos parches del *kernel* tienen licencia GPLv2. Este tipo de licencias de software libre permite a los desarrolladores y a la industria aprovechar las caracterís-

Publicado en las Actas del V Simposio de Sistemas de Tiempo Real, CEDI 2016, Colección Aquilafuente 223, Ediciones Universidad de Salamanca, págs. 69-80, 2016
ISBN: 978-84-9012-631-8

ticas ofrecidas por Android, aplicar parches desarrollados por terceros y realizar modificaciones. Además, el *kernel* de Android está basado en el de Linux. Las últimas versiones de Android que se ejecutan en los dispositivos más modernos utilizan las versiones 3.4 o 3.10 del *kernel* de Linux. Este hecho permite a los desarrolladores utilizar características avanzadas ofrecidas por Linux.

Android está orientado a la ejecución de aplicaciones desarrolladas en Java, aunque es posible ejecutar aplicaciones escritas en cualquier lenguaje utilizando un compilador adecuado. Este sistema operativo proporciona un *framework* que facilita el desarrollo de aplicaciones Java para diferentes dominios. Para alcanzar este objetivo, Android está formado por diferentes capas de componentes software (ver figura 1). La capa más baja corresponde con el *kernel* de Linux, el cual proporciona el funcionamiento básico del sistema, como por ejemplo el manejo de la memoria, los procesos y los drivers. Encima de la capa compuesta por el *kernel*, hay otra capa que contiene un conjunto de librerías nativas que están escritas en C o C++ y son compiladas para una arquitectura hardware específica. Estas librerías incluyen una modificación de la librería C de gnu *glibc* llamada *bionic*. Esta librería está diseñada específicamente para Android pero como analizaremos más adelante presenta algunos inconvenientes cuando es utilizada con aplicaciones de tiempo real.

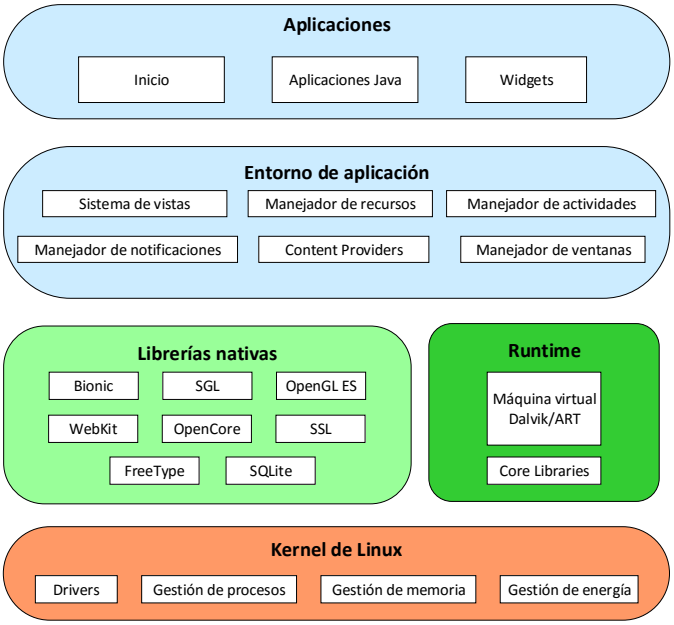

Fig. 1. Arquitectura software de Android

Un componente clave de Android es su máquina virtual para aplicaciones Java. Las últimas versiones de Android incorporan una nueva máquina virtual Java denominada ART (Android RunTime) [1], la cual es la sucesora de la ya obsoleta Dalvik. ART se encarga de ejecutar las aplicaciones escritas en Java compiladas en un formato especí-

fico (*Dalvik executable*) que permiten su portabilidad a través de distintas arquitecturas hardware.

Como ya hemos descrito en un trabajo previo [2], Android al basarse en el *kernel* de Linux ofrece mecanismos que nos permiten aislar un núcleo en un dispositivo con un procesador multinúcleo, de tal modo que se puede utilizar dicho núcleo como un entorno donde ejecutar aplicaciones nativas con requisitos temporales que se ejecutan directamente sobre el *kernel* y las librerías nativas. No obstante, este tipo de aplicaciones utilizan la ya citada librería *Bionic* desarrollada por Google, que no posee todas las características que tiene la implementación tradicional. En particular no soporta ningún protocolo de sincronización para los mutexes, siendo esto un requisito indispensable para cualquier aplicación de tiempo real.

El presente trabajo se estructura del siguiente modo: en la Sección 2 se describen algunos de los trabajos relacionados. En la Sección 3 se muestra la solución que hemos planteado para conseguir ejecutar aplicaciones de tiempo real en Android. La Sección 4 describe brevemente las limitaciones de la librería *Bionic* para aplicaciones de tiempo real. En la Sección 5 se explica cómo superar estas limitaciones utilizando la librería tradicional *glibc* en Android. En la Sección 6 se hace una caracterización de Android para su uso con requisitos de tiempo real. Por último, en la Sección 7 se presentan las conclusiones y los trabajos futuros.

2 Trabajos relacionados

Algunos trabajos [3] [4] [5] [6] han analizado la posibilidad de utilizar Android para ejecutar aplicaciones con requisitos temporales. En ellos se llega a la conclusión de que en primera instancia la plataforma Android es inapropiada para usarse en entornos de tiempo real, a menos que apliquemos mecanismos o modificaciones al sistema operativo. Los principales problemas encontrados en estos estudios son los siguientes:

- La librería *Bionic* tiene más restricciones que la *libc* tradicional y algunas de ellas dificultan su uso en sistemas de tiempo real.
- La variabilidad en los tiempos de respuesta del *kernel* de Android y de la máquina virtual Java impiden tener tiempos de respuesta acotados.
- El *kernel* de Linux utiliza por defecto el planificador llamado "*completamente justo*" (*Completely Fair Scheduler*, en inglés). Este planificador proporciona fracciones temporales dinámicas entre las distintas tareas del sistema para intentar ser lo más justo con todas las tareas que deben ser ejecutadas. Esto es claramente incompatible con los requisitos de predictibilidad de una aplicación de tiempo real.

Debido a las limitaciones citadas anteriormente, un trabajo anterior [5] ha propuesto cuatro posibles soluciones para que Android sea adecuado para ejecutar aplicaciones de tiempo real. En la figura 2a se ilustra la primera solución propuesta, que consiste en ejecutar todas las aplicaciones con requisitos temporales sobre un *kernel* de Linux con características de tiempo real. La siguiente solución que se plantea consiste en añadir una máquina virtual con características de tiempo real (RT- JVM); de este modo gracias a un *kernel* de Linux de tiempo real se podrían ejecutar programas Java

de tiempo real (ver figura 2b). En la figura 2c se ilustra la siguiente propuesta que propone utilizar los servicios de un *kernel* de tiempo real junto con una modificación de la máquina virtual de Android (ART VM extendida). La última solución planteada en la figura 2d utiliza un hipervisor de tiempo real capaz de ejecutar el sistema operativo Android en concurrencia con un sistema operativo de tiempo real.

A raíz de las propuestas anteriores se han llevado a cabo algunos desarrollos para crear extensiones de tiempo real para Android. La solución ilustrada en la figura 2a se ha realizado en un trabajo [7] en el cual se han aplicado una serie de parches (*RT_PREEMPT*) sobre el *kernel* Android/Linux para conseguir características de tiempo real en el sistema. Además para permitir una comunicación entre las aplicaciones Java propias de Android con las de tiempo real han desarrollado un canal de comunicaciones y sincronización entre los dos tipos de aplicaciones.

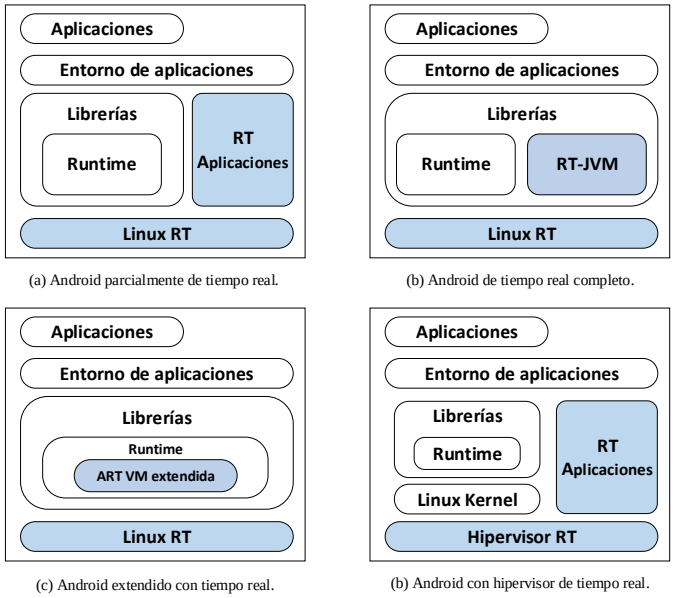

Fig. 2. Soluciones teóricas para la integración de tiempo real en Android [5]. Las partes de color azul indican cambios en la arquitectura de Android.

Otro trabajo [8] [9] ha realizado una adaptación de Android basándose en la solución propuesta en la figura 2c. En este caso también han modificado el *kernel* de Android/Linux con el parche *RT_PREEMPT* y además han añadido modificaciones en otros componentes de Android como en el recolector de basuras de la máquina virtual Java, en la clase *Service* para permitir el cambio de prioridades a nivel de aplicación Java y en el módulo *Binder* para añadir herencia de prioridades en las llamadas remotas a procedimientos dentro de Android.

Existe otro estudio [10] [11] que ha optado por una solución diferente a las cuatro propuestas en la figura 2. En este caso se crea un prototipo denominado RTDroid que pretende ser compatible con las aplicaciones escritas para la plataforma Android. Se

utiliza una máquina virtual de Java con características de tiempo real (Fiji-VM) sobre un sistema operativo de tiempo real (Linux-RT o RTEMS). La API original de Android ha sido recreada paso a paso para poder ejecutar aplicaciones Android. En cierto modo está solución se basa en la figura 2b, pero en este caso, han construido todo el sistema desde cero.

En las dos primeras implementaciones citadas anteriormente nos encontramos con la alta complejidad que reside en la adaptación del *kernel* de Android/Linux para poder aplicar parches de tiempo real sobre él, ya que las últimas versiones del *kernel* de Android no están integradas en la rama principal de desarrollo del *kernel* de Linux. Y la solución propuesta con RTDroid requiere un fuerte proceso de adaptación con cada nueva versión del sistema que aparezca. Debido a esto, en un estudio anterior [2] hemos buscado una solución más portable y fácil de desarrollar y de mantener. En la siguiente sección vamos a describir brevemente dicha solución.

3 Aislamiento de la CPU para ejecutar aplicaciones de tiempo real

Android es un sistema operativo que está en constante evolución y por ello en todas las soluciones citadas en la sección anterior existe una fuerte dependencia con la versión Android utilizada durante su desarrollo. Además, este sistema operativo es una plataforma muy fragmentada [12] lo que hace que todas las soluciones que requieran modificar el *kernel* o la plataforma en general sean difíciles de mantener y extender a distintas clases de dispositivos.

En nuestro estudio previo [2] hemos presentado un mecanismo para ejecutar aplicaciones de tiempo real laxo sobre un dispositivo Android. Para ello las aplicaciones con requisitos temporales (que estarán desarrolladas en lenguaje C) son ejecutadas directamente sobre el *kernel* de Linux en un núcleo aislado de la CPU, sin necesidad de modificar el código del *kernel* ni el de la plataforma. Esta solución puede ser usada en cualquier dispositivo Android con un procesador multinúcleo y *kernel* de Linux versión 2.6 o superior. Dichas aplicaciones con requisitos temporales deben ser ejecutadas haciendo uso de las prioridades de tiempo real que ofrece el *kernel* de Linux/Android (política de planificación *SCHED_FIFO*).

Las versiones del *kernel* de Linux superiores a la 2.6, por defecto tienen activada una opción que hace que la mayor parte de este sea expulsable, por ello en cualquier momento durante la ejecución de código perteneciente al *kernel* se puede producir una expulsión, excepto en los manejadores de interrupciones y regiones protegidas con *spinlocks*. Además en estas versiones del *kernel* también se incluyen políticas de planificación de tiempo real (*SCHED_FIFO* y *SCHED_RR*).

A pesar de todas las características ofrecidas por el *kernel* de Linux/Android existen algunos inconvenientes que debemos solventar si queremos tener un grado de predictibilidad razonable para aplicaciones con requisitos temporales:

- Interferencias entre aplicaciones de tiempo real de diferentes aplicaciones que se pueden estar ejecutando en el sistema.

- Efecto de los manejadores de interrupciones sobre las tareas de la aplicación.
- Cambios dinámicos de frecuencia y apagado automático de los núcleos del procesador.
- Limitaciones de la librería *bionic* para las aplicaciones de tiempo real.

Las tres primeras limitaciones se han resuelto en el estudio previo [2] mientras que en el presente trabajo se resuelve la cuarta limitación relacionada con la librería *bionic*. A continuación se procede a resumir los pasos realizados en el estudio previo [2] para alcanzar nuestra solución que evita interferencias de otras aplicaciones de tiempo real del sistema, elimina interrupciones, fija la frecuencia y evita el apagado de los núcleos del procesador.

Si nos aprovechamos de las ventajas ofrecidas por los procesadores multinúcleo podemos aislar un núcleo de la CPU utilizando mecanismos ofrecidos por Linux/Android para así conseguir un entorno para las aplicaciones de tiempo real. Estas aplicaciones se ejecutarán directamente sobre el *kernel* de Linux/Android y las librerías nativas ofrecidas por el sistema. En la figura 3 se ilustra la arquitectura llevada a cabo para nuestra solución aquí descrita.

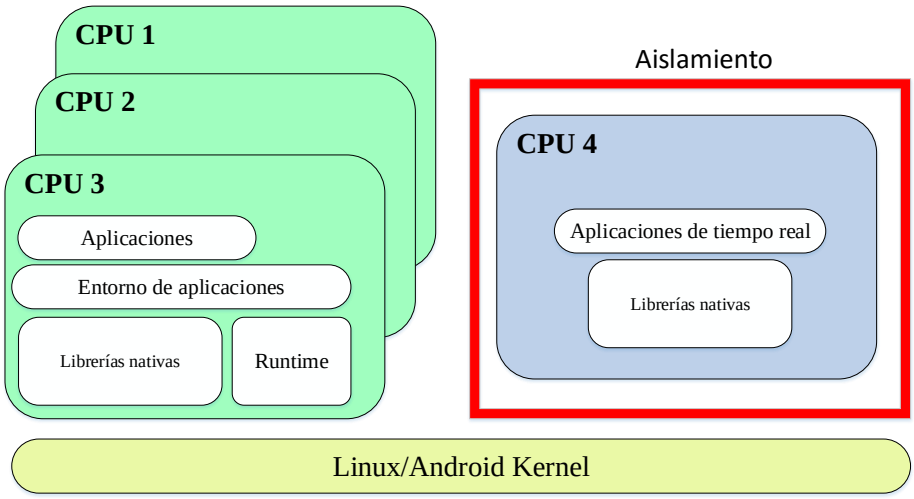

Fig. 3. Solución propuesta en un trabajo previo [2] para ejecutar aplicaciones de tiempo real en Android.

Linux proporciona algunos mecanismos para aislar CPUs de la actividad general del planificador del sistema. El más conveniente para nuestro propósito es el denominado *Cpuset*. Esta funcionalidad proporcionada por el *kernel* no viene activada por defecto en los dispositivos Android, lo que hace que debamos recompilar el *kernel*. Su objetivo consiste en restringir el número de procesadores y recursos de memoria que un

conjunto de procesos pueda utilizar. Cuando el sistema operativo arranca, todos los procesos pertenecen a un único *cpuset*. Si tenemos suficientes privilegios podemos crear nuevos *cpusets* y mover procesos de uno a otro. De este modo podemos mover todos los procesos del sistema a un *cpuset* y al mismo tiempo se puede crear otro *cpuset* donde únicamente se asignen los procesos de tiempo real.

La implementación del mecanismo de los *cpuset* hace que por definición todos los nuevos procesos que se creen se asignen al mismo *cpuset* donde está su proceso padre, a excepción de los procesos hijos de *kthreadd*. Este demonio se ejecuta en el espacio del *kernel* y es utilizado por el sistema para crear nuevos *threads* del *kernel*. Por consiguiente no podemos garantizar que algunos *threads* del *kernel* no se ejecuten en un núcleo aislado. A pesar de ello en los numerosos experimentos realizados esta situación apenas se repite y no tiene gran incidencia sobre los tiempos de ejecución.

Con el anterior mecanismo de aislamiento no conseguimos evitar que lleguen interrupciones a un núcleo aislado, pero a partir de la versión 2.4 del *kernel* de Linux se ha incluido la posibilidad de asignar ciertas interrupciones a un núcleo del procesador (o a un conjunto de núcleos). Esta funcionalidad se denomina *SMP IRQ affinitiy* y nos permite controlar qué núcleos manejan las diferentes interrupciones que se producen en el sistema. Para cada interrupción hay un directorio en el sistema donde existe un fichero que nos permite establecer la afinidad modificando una máscara. No todas las interrupciones son enmascarables, por ejemplo las producidas entre los núcleos del procesador (*IPI-interprocessor interrupts*, en inglés). Al igual que con los *threads* del *kernel* las numerosas pruebas realizadas muestran que el impacto de estas interrupciones es escaso.

Para alcanzar mayor grado de predictibilidad en los tiempos de ejecución es necesario fijar la frecuencia de los núcleos destinados a ejecutar aplicaciones con requisitos temporales. Además, algunos dispositivos Android utilizan demonios que apagan los núcleos de la CPU que no están siendo usados para así poder ahorrar energía. Para evitar que esto ocurra en el núcleo aislado debemos desactivar dichos demonios.

Aplicando todos los mecanismos descritos anteriormente, en el estudio previo que hemos realizado [2] se ha determinado que se consiguen mejoras suficientemente sustanciales en la ejecución de una tarea simple en un núcleo aislado respecto a ejecutar esa misma tarea de una manera no aislada. Esta mejora es suficiente para requisitos de tiempo real laxo.

4 Librería Bionic y sus limitaciones para tiempo real

La implementación de la librería estándar C habitualmente utilizada en Linux es la GNU C Library (comúnmente conocida como *glibc*). Esta librería proporciona soporte para los lenguajes C y C++. Como Android está basado en el *kernel* de Linux parecería lógico asumir que se utilizase esta librería, sin embargo debido a las especificaciones y requisitos de Android se decidió utilizar una nueva librería bautizada con el nombre de *Bionic*. Existen tres razones principales por las que se decidió no utilizar una librería estándar como la *glibc* e implementar una nueva:

- Los dispositivos móviles por norma general tiene un espacio de memoria bastante más limitado que otros dispositivos más "tradicionales", como por ejemplo un ordenador de escritorio.
- Los procesadores que utilizan los dispositivos con Android poseen velocidades inferiores si los comparamos con ordenadores de sobremesa o portátiles.
- Los desarrolladores de Android querían una librería que no estuviese sujeta a las limitaciones de la licencia GPL.

El *kernel* de Linux utiliza una licencia GPL, pero un requisito de Android es que en él se puedan utilizar aplicaciones de terceros que no sólo permitan ser monetizadas sino que en estas aplicaciones se pueda utilizar código propietario. De tal modo que se decidió utilizar una licencia BSD, que sigue siendo de código abierto pero no obliga a que todo el código que se crea a partir de ellas herede la condición de código abierto; por consiguiente los programadores de aplicaciones Android podrán desarrollar código propietario para este sistema operativo.

4.1 Ventajas de la librería *Bionic*

Esta librería tiene optimizado su tamaño debido a que todos los comentarios de los ficheros de cabecera han sido eliminados, y han suprimido todo el código que han considerado inútil para Android. Por otro lado como ya se ha comentado previamente se distribuye con licencia BSD lo que evita cualquier problema legal utilizando software propietario en el espacio de usuario. Por último, existe una ventaja que reside en su optimización para procesadores lentos, para lo que se han realizado cambios en la implementación de la librería *Pthreads* basada en el estándar POSIX. Aunque esto para el caso de querer utilizar esta librería para aplicaciones de tiempo real es un gran inconveniente como veremos a continuación.

4.2 Limitaciones para tiempo real

La librería *Pthreads* [13] que se encuentra en la implementación de la *glibc* tradicional se basa en el estándar POSIX definido por el IEEE para ser compatible a través de diferentes sistemas operativos. Sin embargo como se ha comentado en la subsección anterior la librería *Bionic* ha realizado importantes cambios en este aspecto.

Para determinar si podemos usar *Bionic* en aplicaciones de tiempo real escritas en C hemos ido probando si las funciones más relevantes que se utilizan de manera habitual con requisitos temporales están disponibles. Hemos detectado que algunas tan imprescindibles como las relacionadas con los protocolos de los mutexes no se encuentran implementadas. A continuación se listan las funciones y símbolos no disponibles:

- Mutexes:
    - `pthread_mutexattr_setprotocol`,
    - `pthread_mutexattr_setprioceiling`.
    - `pthread_mutexattr_getprioceiling`.

    - `pthread_mutexattr_setprioceiling`.
- Prioridades:
    - `pthread_setschedprio`.
    - El símbolo `PTHREAD_EXPLICIT_SCHED`.

También existen otras funciones relevantes para aplicaciones de tiempo real que no están implementadas:

- Señales:
    - `sigwaitinfo`.
    - `sigqueue`.
    - `sigtimedwait`.

Estas limitaciones serían más que suficientes para considerar Android un sistema operativo inadecuado para ejecutar aplicaciones de tiempo real, por ello en la siguiente sección se describe la solución adoptada para solventar dichas limitaciones.

5 Glibc en Android

Lo primero que se puede considerar para poder solucionar las limitaciones de tiempo real descritas en la sección anterior podría ser modificar el código de la librería *Bionic* para dar soporte a las funciones no implementadas. Esto supondría un gran esfuerzo y una constante adaptación a las nuevas versiones de la librería que van apareciendo. Por ello, hemos decidido optar por una solución más portable y fácil de aplicar. Esta consiste en utilizar la librería tradicional g*libc* en Android.

Para poder utilizar la librería *glibc* en Android, en nuestro caso[1] es necesario usar una librería compilada y adaptada para arquitecturas ARM que ejecuten un sistema operativo Linux. La forma más directa e inmediata es utilizar un compilador cruzado ARM/Linux con la opción *static* seleccionada, para así conseguir un código ejecutable que incorpore todas las funciones de las librerías de las que haga uso el programa compilado. Esto nos limita a la ejecución exclusiva de programas enlazados estáticamente, de tal modo que también hemos optado por conseguir ejecutar programas que hagan uso de la librería *glibc* en Android de manera dinámica. Para lograr esto se debe realizar lo siguiente:

- Llevar todas las librerías dinámicas al dispositivo Android donde queremos ejecutar las aplicaciones nativas.
- Durante la compilación debemos indicar cuál es el enlazador dinámico y cuál es la ruta donde se encuentran las librerías dinámicas. A modo de ejemplo se expone la orden de compilación para un programa sencillo:

[1] Todas las pruebas se realizan en un Nexus 5 con arquitectura ARM y la versión de Android 6.0.

```
arm-linux-gnueabi-gcc hello_world.c -o hello_world
-Wl,--dynamic-linker=/data/local/libs/ld-linux.so.3
-Wl,-rpath=/data/local/libs -fPIE -pie
```

Teniendo en cuenta que el *kernel* que incorporan los dispositivos con Android no sigue la rama principal de desarrollo del *kernel* de Linux, sería arriesgado afirmar que la librería *glibc* se puede utilizar sin inconveniente alguno en este sistema operativo sin antes realizar algunos tests. Hemos adaptado los test funcionales que están disponibles en el conjunto denominado "Open POSIX Test Suite" [14], donde se realizan pruebas funcionales para *threads*, semáforos, temporizadores, variables condicionales, colas de mensajes y protocolos de herencia de prioridad con mutexes. Todos estos tests han sido pasados satisfactoriamente cuando se ejecutaban en Android haciendo uso de la librería *glibc*, y por consiguiente podemos afirmar que es posible hacer uso de esta librería sobre el *kernel* Linux/Android.

6 Caracterización del sistema para tiempo real

Para poder caracterizar algunas funciones POSIX utilizadas habitualmente con aplicaciones de tiempo real se ha utilizado un teléfono Nexus 5 con un procesador de 4 núcleos a una frecuencia de 2,2 Ghz ejecutando la versión de Android 6.0. Además se han realizado las medidas para dos casos distintos, en el primero de ellos (test A) se miden las funciones en un procesador sin ninguno de sus núcleos aislados; únicamente todos los *threads* de los tests se ejecutan con prioridades de tiempo real. En el otro caso (test B) se ha aislado uno de los núcleos del procesador aplicando los mecanismos descritos en la sección 3. En ambos casos se utiliza la librería tradicional *glibc* y en el sistema se ha establecido una carga de trabajo alta (ejecución de benchmarks y descarga de paquetes por red) para tratar de simular el peor de los escenarios posibles.

	Test A: sin aislamiento	Test B: con un núcleo aislado
Bloqueo de un mutex libre (*lock*)	Mínimo: 1,874 µs Máximo: 1646,301 µs Media: 2,143 µs	Mínimo: 0,261 µs Máximo: 34,064 µs Media: 0,318 µs
Bloqueo de un mutex libre (*trylock*)	Mínimo: 1,874 µs Máximo: 91,303 µs Media: 2,208 µs	Mínimo: 0,261 µs Máximo: 36,304 µs Media: 0,323 µs
Desbloqueo de un mutex (*unlock*)	Mínimo: 1,770 µs Máximo: 111,980 Media: 2,121	Mínimo: 0,261 µs Máximo: 33,387 µs Media: 0,317 µs
Delay de 1000 µs: *clock_nanosleep* (absoluto)	Mínimo: 1030,924 µs Máximo: 5891,718 µs Media: 1112,932 µs	Mínimo: 1027,969 µs Máximo: 1108,230 µs Media: 1067,169 µs
Cambio de prioridad (*pthread_setschedparam*)	Mínimo: 7,970 µs Máximo: 180,053 µs	Mínimo: 7,454 µs Máximo: 64,147 µs

	Media: 15,793 µs	Media: 8,114 µs
Cambio de prioridad (*pthread_setschedprio*)	Mínimo: 8,229 µs Máximo: 130,345 µs Media: 16,162 µs	Mínimo: 7,811 µs Máximo: 68,396 µs Media: 8,446 µs

Tabla 1. Caracterización de algunas funciones POSIX utilizadas habitualmente en aplicaciones de tiempo real.

Los resultados que se muestran en la tabla 1 se han conseguido midiendo la duración de las distintas funciones durante 150000 ejecuciones para cada caso. La tabla 1 arroja una mejora sustancial para el test B (núcleo aislado) donde los tiempos de peor caso son en algunos casos unas 7 veces inferiores y la media de los resultados es alrededor de solo un 15% superior al valor mínimo obtenido en todas las pruebas para el test B (núcleo aislado). Los tiempos máximos observados en el test sin aislamiento pueden ser provocados por cualquier interrupción del sistema o por otras tareas que utilicen prioridades de tiempo real mayores.
Incluso en el caso aislado, en un sistema operativo como Android donde no tenemos pleno control sobre todas las actividades del sistema no se pueden determinar con total exactitud tiempos máximos de respuesta, aunque en los numerosos tests realizados hemos observado que se obtienen valores similares a los mostrados en la tabla 1.

7 Conclusiones y trabajos futuros

En este trabajo se ha presentado una solución para poder utilizar Android con aplicaciones de tiempo real laxo haciendo uso de mecanismos proporcionados por el propio sistema operativo, y aprovechando los procesadores multinúcleo cada vez más presentes en dispositivos Android. Debido a que la librería *bionic* implementada en Android no incorpora todas las funciones utilizadas habitualmente en aplicaciones de tiempo real se ha optado por utilizar la librería tradicional *glibc* usada en los sistemas operativos Linux y se ha testado su correcto funcionamiento en Android. De este modo no es necesaria ninguna modificación del *kernel* ni de ninguna librería proporcionada por la plataforma Android. Esto supone una alta portabilidad para los distintos dispositivos y versiones de Android disponibles actualmente. Algunos tests que hemos realizado sobre la solución propuesta demuestran que se obtienen mejoras sustanciales con respecto al uso de Android sin mecanismos de aislamiento.

Nuestro siguiente objetivo es medir el impacto del uso de drivers de propósito general de Android sobre las aplicaciones de tiempo real que ejecutan en un entorno aislado. Además también es necesario desarrollar un mecanismo de comunicación y sincronización entre las aplicaciones de tiempo real ejecutadas en núcleo aislado y el resto de aplicaciones de Android.

Agradecimientos. Este trabajo ha sido financiado parcialmente por el Gobierno de España con referencia TIN2014-56158-C4-2-P (M2C2) y por el programa de becas predoctorales de la Universidad de Cantabria.

Referencias bibliográficas

1. Andrei Frumusanu (July 1, 2014). "A Closer Look at Android RunTime (ART) in Android L". AnandTech. Retrieved July 5, 2014.
2. Alejandro Pérez Ruiz, Mario Aldea Rivas and Michael González Harbour. “CPU Isolation on the Android OS for running Real-Time Applications”.in Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2015.
3. Bhupinder S. Mongia y Vijak K. Madisetti, “Realiable Real-Time Applications on Android OS”. Whitepaper, 2010.
4. Luc Perneel, Hasan Fayyad-Kazan y Martin Timmerman. “Can Android be used for Real-Time purposes?” in International Conference on Computer Systems and Industrial Informatics, ICCSII ’12, pages 1–6, 2012.
5. C. Maia, L. Nogueira y L. M. Pinho. “Evaulating Android OS for Embedded Real-Time Systems”. En Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2010, pages 63- 70, Brussels, Belgium, 2010.
6. Luc Perneel, Hasan Fayyad-Kazan, and Martin Timmerman. “Android and Real-Time Applications: Take Care!”, in Journal of Emerging Trends in Computing and Information Sciences, Volume 4, Special Issue ICSSII.
7. W. Mauerer, G. Hillier, J. Sawallisch, S. Hönick, y S. Oberthür. “Real-time android: deterministic ease of use,” en Proceedings of the Embedded Linux Conference Europe (ELCE '12), 2012.
8. Igor Kalkov, Dominik Franke, John F. Schommer, and Stefan Kowalewski. “A real-time extension to the Android platform”. In Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems, JTRES ’12, pages 105–114, New York.
9. Igor Kalkov, Alexandru Gurghian, and Stefan Kowalewski. “Priority Inheritance during Remote Procedure Calls in Real-Time Android using Extended Binder Framework”. ”. In Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2015.
10. Yin Yan, Shaun Cosgrove, Varun Anand, Amit Kulkarni, Sree Harsha Konduri and Steven Y. Ko, Lukasz Ziarek. "Real-Time Android with RTDroid" in Proceedings of the 12th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2014.
11. Yin Yan, Sree Harsha Konduri, Amit Kulkarni, Varun Anand and Steven Y. Ko, Lukasz Ziarek. "RTDroid: A Design for Real-Time Android" in Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2013.
12. Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong y Eleni Stroulia. “Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs” in Working Conference on Reverse Engineering (WCRE), 2012.
13. Nichols, Bradford, Dick Buttlar, and Jacqueline Farrell. “Pthreads programming: A POSIX standard for better multiprocessing.” O'Reilly Media, Inc., 1996.
14. Open POSIX Test Suite from A GPL Open Source Project, http://posixtest.sourceforge.net/