EXHAUSTIF®: UNA HERRAMIENTA DE INYECCIÓN DE FALLOS POR SOFTWARE PARA SISTEMAS EMPOTRADOS DISTRIBUIDOS HETEROGÉNEOS

Antonio da Silva [1], José F. Martínez [1], Lourdes López [1,] Luis Redondo [2]
[1] *Depto. Ingeniería Telemática y Arquitecturas*
Universidad Politécnica de Madrid
Crta Valencia, Km 7, 28031 Madrid
E-Mail : {adasilva, jfmartin, llopez}@diatel.upm.es
[2] *Métodos y Tecnología*
Paseo de la Castellana 182, 28046 Madrid
E-Mail : lredondo@mtp.es

Resumen: En este artículo se presenta una nueva herramienta de inyección de fallos llamada Exhaustif®. Exhaustif® es una herramienta SWIFI (Software Implemented Fault Injection) para la verificación y validación de los mecanismos de tolerancia a fallos en sistemas empotrados distribuidos heterogéneos, así como para la realización de pruebas de caja "gris" muy útil durante las fases de integración y validación de sistemas. Principalmente Exhaustif® consta de dos partes: EEM y FIK. El Exhaustif® Executive Manager (EEM) es una interfaz gráfica desarrollada en Java que permite definir el conjunto de casos de prueba de inyección de fallos y que usa una base de datos SQL para almacenar los resultados del sistema bajo test (SUT), ello permite realizar análisis de los resultados obtenidos después de la inyección. El Fault Injector Kernel (FIK) es el módulo inyector y bajo las órdenes del EEM realiza inyecciones de fallos en aplicaciones que se ejecutan en el sistema bajo test usando técnicas SWIFI puras. Se ha implementado un FIK para una placa EADS-Astrium SPARC ERC32 ejecutando aplicaciones bajo el sistema operativo de tiempo real RTEMS. Exhaustif® realiza corrupciones de memoria y registros con disparos temporales y presenta un mecanismo de interceptación de funciones optimizado que permite corromper los argumentos de entrada y/o el valor de retorno de las funciones con una sobrecarga de ejecución mínima. EADS-Astrium es líder mundial en el diseño y manufactura de satélites, sus actividades cubren completamente las telecomunicaciones civiles y militares y sistemas de observación de la tierra. Este trabajo es liderado por Métodos y Tecnología (MTP: www.mtp.es y www.exhaustif.es) en colaboración con la Universidad Politécnica de Madrid y cofinanciado por la Consejería de Economía e Innovación Tecnológica de la Comunidad Autónoma de Madrid.

Palabras Clave: Tolerancia a fallos, SWIFI, sistemas empotrados, sistemas heterogéneos, confiabilidad, fiabilidad, verificación, validación, pruebas de caja gris.

.

1. INTRODUCCIÓN

La evaluación de la confiabilidad de un sistema incluye el estudio de los fallos y los errores. Para entender los fallos potenciales, se han desarrollado técnicas experimentales que pueden ser aplicadas tanto al hardware como al software. Estas técnicas no sólo son aptas durante la fase de análisis y diseño del sistema, sino también durante las de prototipado y producción.

En la sociedad actual cada vez cobran más importancia los sistemas informáticos que controlan cualquier proceso, desde un sencillo electrodoméstico o un sistema de control de automoción a los grandes sistemas bancarios o los de telecomunicaciones. En algunas de estas aplicaciones de su correcto funcionamiento dependen vidas humanas, como en el caso de los sistemas de control de tráfico, equipos de soporte vital o centrales de energía nuclear. Al ser estas aplicaciones de tiempo real, no sólo se debe garantizar que el resultado es del valor correcto, sino que se debe entregar dentro del plazo especificado.

Existen algunos ejemplos de "conocidos" fallos de sistemas con importantes repercusiones negativas. Como por ejemplo:

- El apagón de América Online (AOL) que afectó a seis millones de usuarios.
- La destrucción de Ariane 5 debido a un fallo software que costó 250 billones de dólares a la agencia Aeroespacial Europea (ESA). [1]

Los requisitos de confiabilidad de estos sistemas no pueden ser alcanzados exclusivamente con técnicas de prevención de defectos (diseño, aseguramiento de la calidad, blindaje, etc.), ya que la hipótesis de poder construir un sistema absolutamente exento de defectos no es realista [2]. Es necesario, por tanto, construir sistemas que sean capaces de prestar el servicio esperado incluso ante la presencia de defectos. La validación de los atributos de confiabilidad en un sistema se puede hacer de dos formas: teórica o experimental. La evaluación teórica se realiza evaluando un modelo del sistema mientras la validación experimental se realiza mediante la observación del sistema real en funcionamiento normal mientras se inyectan fallos de forma deliberada.

La inyección de fallos puede descubrir errores que los procedimientos normales no pueden. En primer lugar pone a prueba los mecanismos de excepción y tratamiento de errores que en circunstancias normales no son suficientemente probados y ayuda a evaluar el riesgo comprobando cuan defectuoso puede llegar a ser el comportamiento del sistema en presencia de errores. Todos los métodos de inyección de fallos están basados en características hardware/software concretas del sistema al que se aplica, por lo que la generalización es muy complicada.

La herramienta aquí presentada sirve precisamente para ejercitar los mecanismos de tolerancia a fallos introducidos en el sistema.

1.1. Herramientas basadas en SWIFI

La técnica de inyección de fallos por software, Software Implemented Fault Injection, es muy atractiva ya que no requiere de un hardware específico para realizar la inyección. Se puede usar para probar los mecanismos de tolerancia a fallos de aplicaciones, sistemas operativos y, en general, componentes software COTS (Commercial Off-The-Shelf) de los que no se dispone el código fuente. El uso de COTS y la reutilización de código [3] es cada vez más frecuente lo cual complica la evaluación de los mecanismos de excepción.

Si el objeto de la prueba es una aplicación, el inyector se inserta en la misma aplicación o bien en una capa intermedia entre la propia aplicación y el sistema operativo. Por el contrario, si el objeto es el sistema operativo, el inyector debe estar embebido en el mismo sistema operativo, cuando esto resulta muy complicado se debe añadir una capa intermedia (wrapper) entre el sistema y el hardware de la máquina.

Como ventaja de técnicas SWIFI para la inyección de fallos se podría destacar que:

- La intrusión temporal puede reducirse al tiempo necesario empleado en el manejo de una excepción y puede despreciarse si las restricciones temporales del sistema bajo prueba no son estrictas.
- Se pueden realizar muchos experimentos de forma sencilla y controlada.
- Los experimentos se realizan sobre el hardware real.
- No necesita ningún equipamiento especial y son fácilmente adaptables al sistema bajo prueba.

La principal característica de la inyección de defectos software es que es capaz de inyectar fallos en cualquier unidad funcional accesible mediante software, tales como memoria, registros, periféricos. El objetivo de la inyección de defectos software es reproducir, en el ámbito lógico, los errores que se

reproducen tras fallos en el hardware. Una buena caracterización del modelo de fallos, debería permitir que éste fuera lo más versátil posible, permitiendo un mayor número de combinaciones entre la localización, condición de disparo, tipo de defecto y duración, de modo que la cobertura fuese máxima.
La caracterización del modelo de fallos se lleva a cabo acotando el dónde, el cuándo y durante cuánto tiempo ha de inyectarse el defecto y de que tipo ha de ser éste. Al realizar la inyección de fallos con un trigger temporal y la corrupción de parámetros en llamadas a rutinas Exhaustif® introduce una sobrecarga temporal equivalente a la ejecución de unas pocas instrucciones máquina despreciable para la mayoría de las aplicaciones.

2. ESTADO DEL ARTE

Existen diversas herramientas para la inyección de errores mediante el uso de técnicas SWIFI puras o en conjunción con recursos de depuración hardware presentes en algunos procesadores, siendo la generalización para cualquier tipo de sistema bastante compleja. Como pequeña muestra se podría nombrar: XCEPTION, GOOFI, MAFALDA o FAUMachine.

- Xception[4] es una herramienta de inyección de defectos mediante software y posee una interfaz Scan Chain (SCIFI) para ERC32. Cuando funciona como herramienta SWIFI pura intenta usar las características de depuración y monitorización, presentes en la mayoría de los procesadores para inyectar defectos más realistas. Además, monitoriza la activación de los defectos y su impacto en el sistema, mostrando su comportamiento al detalle. Xception provee un conjunto de inyección de defectos, incluyendo defectos espaciales, temporales y defectos relacionados con la manipulación de los datos en la memoria. Se basa en la utilización de mecanismos de depuración implementados en el procesador a través de las excepciones para realizar la inyección. Primero deja ejecutar a la aplicación hasta el momento del disparo, en el que se entra en modo traza y se ejecuta paso a paso hasta la instrucción a inyectar. Una vez en la instrucción a inyectar se decodifica y se decide que parámetros de la misma modificar en función del tipo de fallo que se pretenda inyectar. Usando SWIFI o SCIFI Xception soporta diversas CPUs como: PowerPc750, Intel x86 y Sparc ERC32 sobre diversos sistemas operativos: LynxOS, VxWorks, Linux, y Win32. También ha sido usado para comprobar la robustez de RTEMS [8].
- Goofi [5] (Generic Object-Oriented Fault Injection) está diseñado para poder adaptarse a varios sistemas bajo prueba y a diferentes técnicas de inyección de defectos. Permite realizar inyección de defectos mediante software (SWIFI) y Scan Chain (SCIFI). Además posee interfaces BDM (Background Debug Mode) y Nexus. Goofi soporta el microprocesador Thor Rard Hard, de SAAB Ericsson Space, los microprocesadores MC68340, HC12 y el microcontrolador MPC565, ambos de Motorola.
- Mafalda [6] (Microkernel Assesment by Fault injection AnaLisys and Design Aid) está orientada a la validación de microkernels (núcleos del sistema operativo) y permite realizar dos tipos de inyecciones. La corrupción de los parámetros de entrada al llamar a una primitiva del microkernel y la inyección de fallos en segmentos de memoria, de código y datos. En el caso de la inyección en

los parámetros de las llamadas al sistema se interceptan las llamadas, bien mediante una librería (wrapper) de llamadas modificada para la inyección, o bien mediante un breakpoint en la entrada al microkernel, haciendo que se ejecute antes de la llamada un manejador encargado de corromper los parámetros. En el caso de la inyección en memoria, código o datos, se definen fallos permanentes o transitorios disparados por eventos espaciales o temporales, donde se invierten los bits elegidos cuidadosamente, con lo que se consigue emular defectos hardware en la memoria, o fallos software modificando los parámetros de entrada al núcleo. En caso de los transitorios, tras la inyección se entra en modo traza, se avanza en la ejecución paso a paso y se restaura la memoria.

- FAU Machine [7] es una máquina virtual que emula el hardware de un PC/AT y sus periféricos. Tiene como ventaja sobre otros sistemas de virtualización del hardware que permite simular el mal funcionamiento de alguno de sus componentes como la memoria. El uso de máquinas virtuales permite la simulación de fallos permanentes, lo cual no es posible con mediante técnicas SWIFI. El uso de entornos de simulación es imprescindible para realizar inyección de errores permanentes en el hardware.

El uso de mecanismos de depuración hardware es muy interesante e imprescindible para la especificación de triggers espaciales pero presenta problemas en entornos distribuidos heterogéneos. Cuando un sistema ha detectado un trigger pasa a un estado congelado (frozen) permitiendo el análisis de su estado mediante interfaces como Boundary-Scan. En este caso, hay que avisar a los demás sistemas de esta circunstancia para que detengan también su actividad. Exhaustif® es una herramienta diseñada para su uso en entornos distribuidos heterogéneos, lo que dificulta el uso de sistemas de depuración particulares siendo SWIFI la opción más fácil de adaptar a los diferentes entornos.

El uso de wrappers [9] sólo es posible si se dispone del código fuente de la aplicación y la inserción de puntos de ruptura software introduce una sobrecarga temporal apreciables.

Exhaustif® modifica el punto de entrada a la rutina insertando un salto a un bloque de código definido específicamente para el fallo que se desea insertar retornando inmediatamente al código de la rutina evitando el mecanismos de excepciones.

3. EXHAUSTIF® TOOL

Exhaustif® está diseñado para analizar dinámicamente sistemas en los que se necesite validar si se satisfacen los requisitos de Fiabilidad. Gracias a la validación flexible del Sistema Bajo Prueba se pueden probar los mecanismos de tolerancia a defectos y se puede verificar la correcta gestión de los defectos de acuerdo a su especificación.

La arquitectura hardware/software de la herramienta Exhaustif® ha sido diseñada para reducir al máximo el impacto de cualquier cambio (inclusión, modificación o borrado de cualquier módulo). Está herramienta diseñada se compone de dos elementos principales:

- EEM (Exhaustif® Executive Manager)
- FIK (Fault Injection Kernel)

El EEM es el responsable de la gestión y control de ejecución de los experimentos y ofrecerá al usuario una interfaz gráfica con la que podrá programar las inyecciones de los defectos de forma comprensible y sencilla,

así como la visualización de los resultados y los datos de la ejecución realizada.
El FIK es un componente software ligero que reside en el System Under Test (SUT) y tiene la función de inyectar los defectos. En esta primera versión el FIK dará soporte a una placa basada en ERC32 facilitada por EADS-ASTRIUM [10] y que ejecutará aplicaciones bajo RTEMS [11].

3.1. Arquitectura de la herramienta

En el diseño del EEM, se ha tenido en cuenta la capacidad de ampliación de la misma para soportar simultáneamente varios SUTs. De esta forma, sería posible inyectar defectos y/o errores en varios sistemas bajo prueba que a su vez formen parte de un sistema más grande, comprobando en que medida afectan en un subsistema los fallos inyectados en otro. De forma más detallada, Exhaustif® está formado por los siguientes componentes, figura 1:

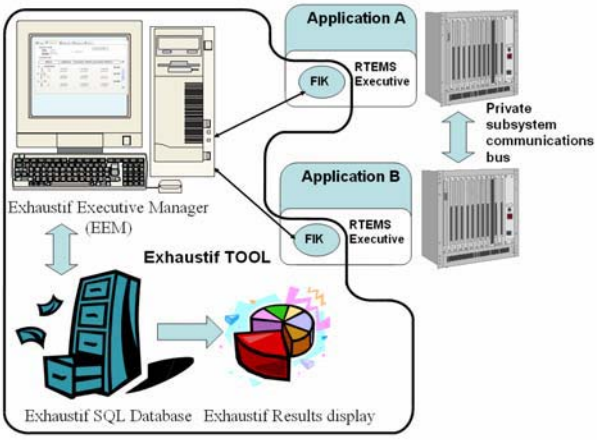

Figura 1. Exhaustif® Architecture

- Un gestor Ejecutivo de Exhaustif® (ExhaustiF® Executive Manager - EEM) que es el responsable de la gestión y control de ejecución de los experimentos. Permite al usuario la configuración de los casos de pruebas y la selección de los fallos que serán inyectados, siendo luego el encargado de secuenciar los fallos a inyectar dentro del SUT. Finalmente, recoge los datos de ejecución y los resultados obtenidos para su posterior visualización. En el desarrollo del EEM se ha tenido en cuenta su portabilidad a diferentes plataformas de ejecución tanto hardware como sistemas operativos por lo que ha sido implementado en Java.
- Un Inyector de Errores y/o Defectos (Fault Injector Kernel - FIK), componente software ligero, se ejecuta en la plataforma del sistema bajo prueba y es el responsable de inyectar los fallos. Es un componente de muy reducido tamaño, prácticamente no intrusivo, ver punto 3.3 y provoca defectos en la memoria, registros de la CPU, unidad de coma flotante y permite la interceptación de llamadas a funciones para la modificación de los argumentos de entrada y del valor retornado por la función.

Además de los ya mencionados, Exhaustif® tiene otros componentes importantes:

- Una línea de comunicación dedicada a conectar el Gestor EEM y el Inyector FIK. En el diseño de Exhaustif® se ha hecho abstracción del sistema de comunicación entre EEM y FIK para permitir una completa flexibilidad en las comunicaciones en función de las capacidades presentes en el SUT. Exhaustif® es así lo suficientemente flexible para admitir diversas interfaces de comunicación como RS-232-C, USB, Ethernet, etc.
- Una BBDD SQL estándar que contiene los datos recopilados de la ejecución del sistema bajo prueba.
- Utilidades de análisis estadístico y visualización de los datos almacenados en la base de datos.

Después del arranque del sistema se procede a la carga del FIK más la aplicación bajo prueba usando los servicios del monitor de arranque presente en la placa, figura 2. Una

vez arrancado el FIK espera el mensaje del EEM con los fallos a insertar. Transcurrido el tiempo de incubación el EEM solicita al FIK la información específica que desee almacenar en la base de datos.

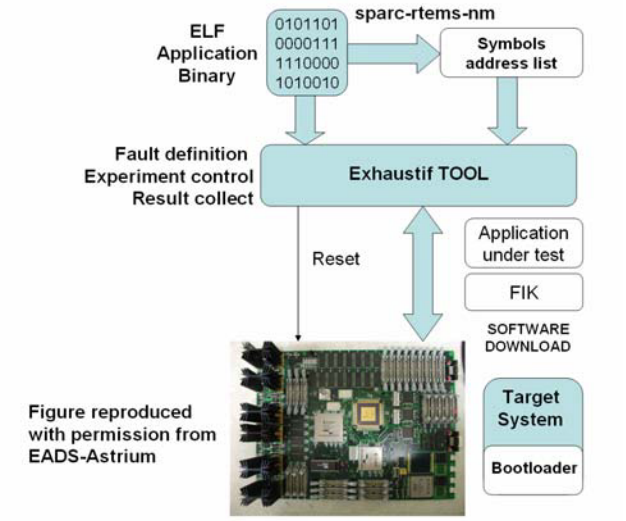

Figura 2. Mapa de símbolos del SW del SUT

Es posible ejecutar un experimento sin inyección de fallos (lo que hemos denominado Golden Run), de manera que sólo se envían al FIK mensajes de petición de información que serán almacenados en la base de datos con la intención de compararlos más tarde con los obtenidos con las ejecuciones con errores y observar los cambios producidos.

3.2. Exhaustif Executive Manager

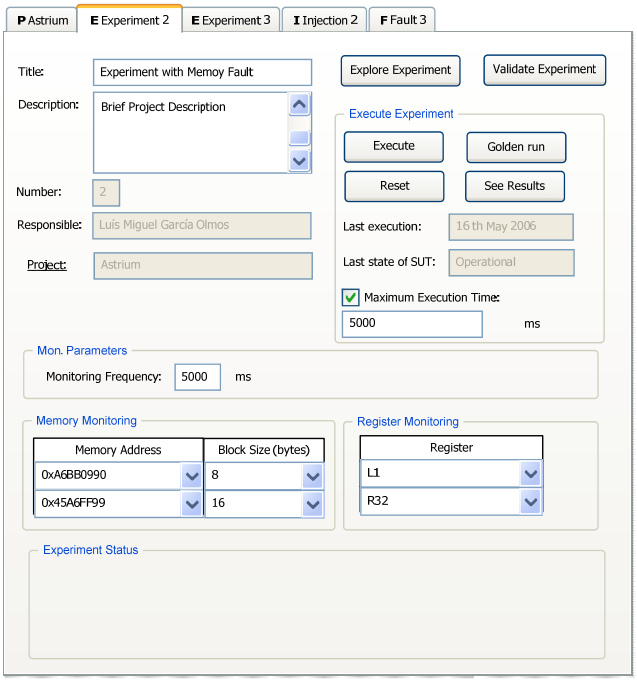

Figura 3. Definición de experimentos

La definición de la batería de pruebas en Exhautif® se realiza a través de proyectos. Un proyecto estará formado por uno o más experimentos, figura 3. Cada experimento está compuesto por inyecciones y éstas a su vez por fallos, para cada fallo se especifica el trigger temporal de disparo y el tiempo de incubación al cabo del cual se procederá a solicitar los resultados de la inyección.

Para cada inyección se define el instante, figura 4, de la inyección que podrá ser now (en el mismo instante en que envía el mensaje de inyección) o un timeout desde el inicio de la aplicación y el tiempo de incubación que se esperará antes de solicitar el contenido de las posiciones de memoria especificadas en el experimento.

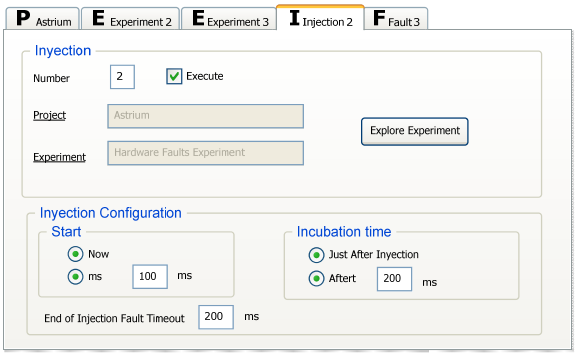

Figura 4. Definición del trigger temporal

De forma gráfica, el siguiente diagrama de secuencia, figura 5, muestra los mensajes intercambiados entre el EEM y el FIK para una inyección now y su relación con el tiempo de incubación.

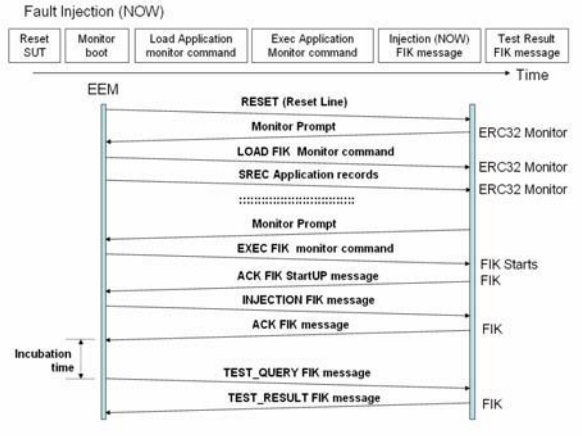

Figura 5. EEM/FIK messages

Exhaustif® permite la especificar corrupciones de memoria y registros del procesador aplicando diferentes patrones de modificación como pueden ser cambios en el estado de un bit (BitFlip), aplicación de una máscara (BitMask) o copia de un valor nuevo, figura 6. Estas modificaciones pueden realizarse instantáneamente en el momento que se ordenan o definiendo un timeout desde el inicio de la aplicación. Todas estas modificaciones son instantáneas y no permanentes.

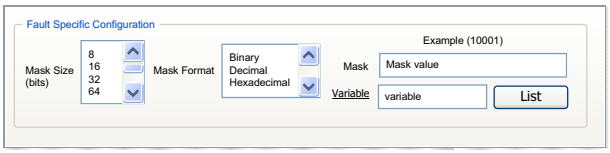

Figura 6. Corrupción de variables y de memoria

Además, Exhaustif® permite además interceptar las llamadas a funciones, figura 7, alterando el contenido de los parámetros de entrada y/o el valor retornado con una sobrecarga de ejecución mínima, ver apartado 3.3.

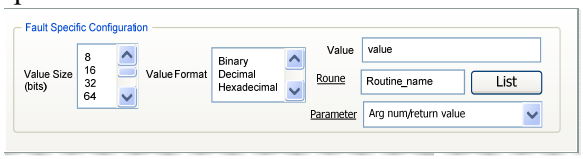

Figura 7. Routine/Call Intercepción

Finalmente, después de la inyección de todos los fallos especificados se visualizan las inyecciones que presentan diferencias respecto a la ejecución sin errores.

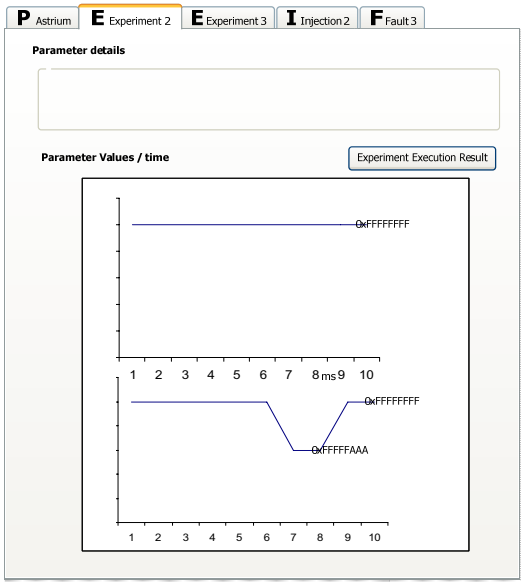

Figura 8. Resultados temporales

Además, es posible una visualización temporal de los contenidos de posiciones de memoria obtenidas durante la ejecución de todas las inyecciones del experimento, figura 8

3.3. Fault Injector Kernel

El FIK es el componente software encargado de inyectar los defectos especificados por el EEM. Las condiciones de diseño imponen que debe ser prácticamente no intrusivo y ocupar un máximo de 128 Kbytes. Por otra parte, el FIK es totalmente independiente de la aplicación y actúa bajo las órdenes del EEM para realizar las inyecciones programadas en los diferentes experimentos.

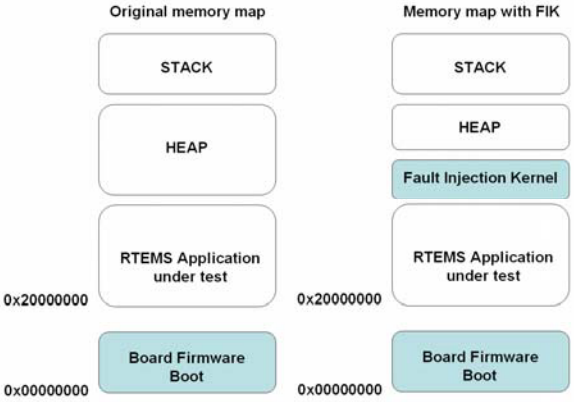

Figura 9. Mapa de memoria del FIK/APP

Como se ha comentado, el FIK es un componente software ligero que realiza la inyección de fallos programada por el EEM. En el caso concreto de la arquitectura ERC32 y bajo el sistema operativo RTEMS el mapa de memoria del sistema bajo prueba sería el mostrado en la figura 9.

- La aplicación se carga a partir de la dirección 0x20000000, el FIK se carga a continuación de la aplicación y usa la misma línea de comunicaciones que el monitor de arranque para comunicarse con el EEM.
- El FIK ocupa parte de la memoria dedicada inicialmente a heap por lo cual modifica la configuración de la librería estandar de C para que los servicios de memoria dinámica se adapten a la nueva configuración.
- El FIK captura el servicio a la interrupción de tiempo real programado por RTEMS, figura 10, para dar servicio a la inyección de fallos con trigger temporal especificados en el mensaje de inyección.

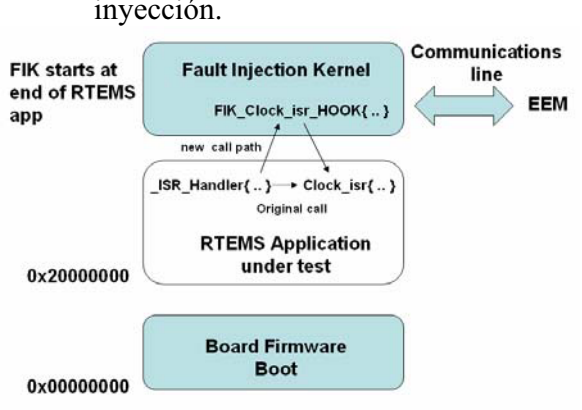

Figura 10. Timer ISR Hook

La elección de un conjunto de fallos representativos es una tarea compleja. Es posible realizar corrupciones de memoria y no provocar ningún fallo, bien porque éstas no están siendo usadas por la aplicación, bien porque el valor corrupto no es usado y es sobrescrito. Exhaustif® además de las corrupciones de memoria y registros permite la interceptación de llamadas a rutinas y la corrupción de los argumentos de entrada de las mismas, así como la alteración del valor de retorno. Mediante la corrupción de los parámetros de entrada es posible una mejor localización del error inyectado y la comprobación de la robustez de una API.

En los esquemas clásicos se inserta un punto ruptura software TRAP en la dirección de comienzo de la rutina y en el tratamiento de la excepción se procede a la modificación de los argumentos y se continua con la ejecución, figura 11.

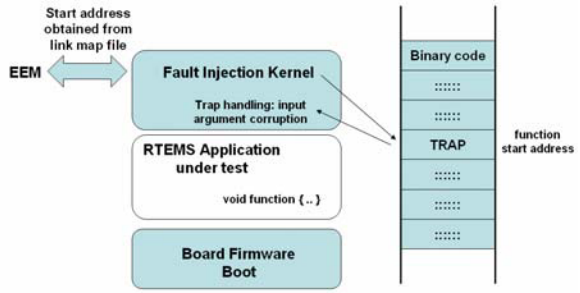

Figura 11. Aproximación tradicional

Este esquema es muy cómodo pero especialmente costoso en términos temporales en la arquitectura Sparc sobre la que se ha desarrollado el prototipo. Exhaustif/FIK realiza bajo demanda y cuando el EEM lo solicite una mutación del código en memoria, insertando un salto incondicional a un bloque de código que procede a la modificación de los argumentos y retorna de nuevo al código de la rutina original, figura 12.

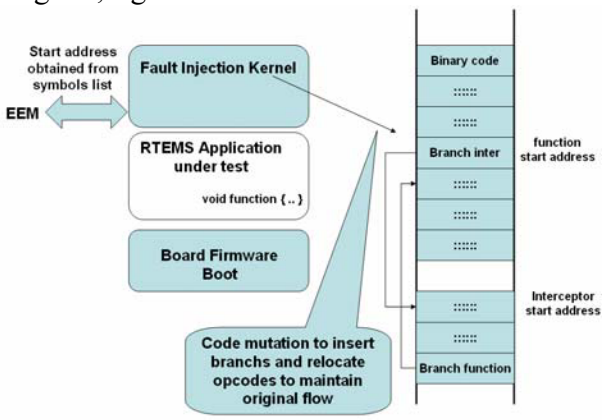

Figura 12. Rutina de intercepción rápida

Simulaciones realizadas con tsim, ERC32/Leon simulator [12], en las cuales se aplica una máscara sobre el segundo argumento de una función se provoca una sobrecarga de solamente 17 ciclos de reloj. En el caso de usar TRAPS son necesarios al menos 25 ciclos de reloj para saltar al comienzo del código que provoca la corrupción. Además en el caso de Sparc dentro de esta rutina habría añadir la gestión de las ventanas de registros, figura 13.

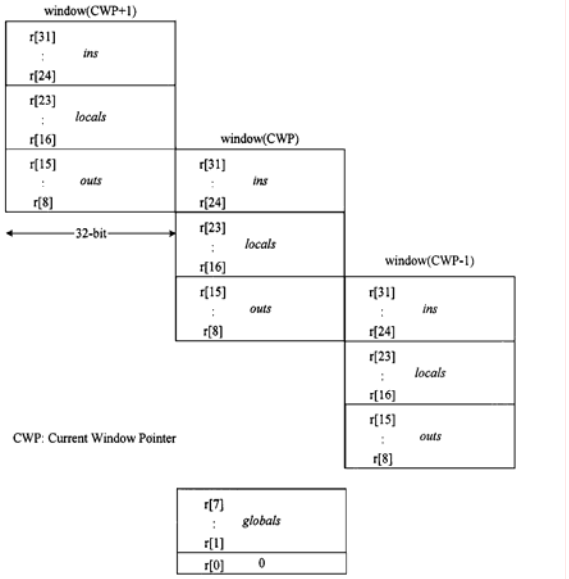

Figura 13. Sparc Register Windows

Sparc utiliza intensivamente el bloque de registros como mecanismo de paso de parámetros a funciones, almacenamiento de la dirección de retorno y valor devuelto por la función. Para ello, usa una ventana deslizante que modifica el conjunto de registros vistos desde el software en cada momento. En un instante dado, el programador tiene a su disposición 32 registros:

- 8 globales
- 8 registros de entrada (ins)
- 8 registros local (locals)
- 8 registros de salida (outs)

Mediante la pareja de instrucciones SAVE/RESTORE es posible mover la ventana actual hacia adelante/atrás. Al avanzar, los registros de salida outs de la ventana anterior se transforman en los registros de entrada ins de la ventana actual.

De acuerdo a Sparc Application Binary Interface (ABI) [13], las llamadas a rutinas en C siguen el siguiente convenio; los argumentos de la función se depositan en los registros de salida [O0 .. O5], en O7 se deposita la dirección de la instrucción de llamada a la función. En el caso de existir más parámetros se usará la pila. Si la función

necesita una nueva ventana de registros, ejecutará una instrucción save que provoca el movimiento de la ventana. En el caso de funciones finales que no invoquen a ninguna rutina, no suele ser necesario mover la ventana.

De esta forma, las acciones llevadas a cabo por el FIK para realizar una interceptación serán las siguientes, figura 14:

- Sustituir la primera instrucción de la función por un salto incondicional al comienzo del código interceptor.
- Corromper el contenido del registro que contiene el valor del parámetro que se desea alterar y saltar de nuevo al código de la función.

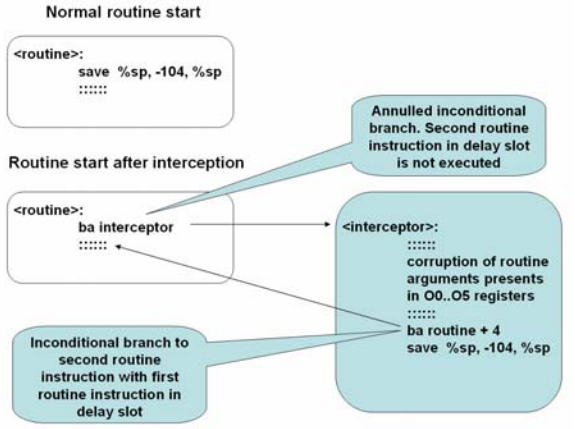

Figura 14. Código interceptor

4. Conclusiones y trabajos futuros

La capacidad de inyectar defectos mediante software constituye una herramienta de sumo interés para la verificación de los mecanismos de tolerancia a fallos previstos en la construcción de sistemas críticos y para la realización de pruebas de caja gris durante la integración HW/SW, integración SW/SW y pruebas de sistema. Va a permitir predecir las consecuencias del mal comportamiento de los sistemas debido a errores no detectados en las pruebas funcionales. Además, esta técnica es aplicable para la validación de componentes software de terceras partes sin necesidad de tener acceso al código fuente. El uso de Exhaustif® contribuye definitivamente a la tarea de construir software más fiable.

El diseño se ha realizado para que la migración a otros entornos sea fácilmente realizable. Así mismo, se están evaluando escenarios para explorar la capacidad de Exhaustif® de controlar varios SUTs simultáneamente y realizar pruebas en los cuales la inyección y la recolección de resultados no se realizan sobre el mismo sistema bajo.

Se dotará a Exhaustif® de un modo automático de generación de experimentos en función de modelos estadísticos de las corrupciones de memoria y registros provocadas por SEUs y EMC. De esta forma, los fallos donde se provocan y el tiempo de duración de los mismos vendrían determinadas por estos modelos y serían automáticamente seleccionados por la herramienta.

Otros trabajos futuros consisten en portar el FIK a otras plataformas hardware como i386, motorola ColdFire y/o sistemas operativos como uCLinux.

5. REFERENCIAS

[1] *J. Voas, "Software Fault-Injection: Growing 'safer' Systems", In Proc. Of IEEE Aerospace Conference, Snowmass, Febrero, 1997*

[2] J. Carreira y J. G. Silva, "Why do some (weird) people Inject Faults?", Software Engineering Notes vol 23 no 1, pp. 42, Enero 1998.

[3] Gacek C, de Lemos R, "Architectural description of dependable systems". In: Structure for Dependability: Computer-Bases Systems from an Interdisciplinary Perspective. Lecture notes in Computer Science 4527. Springer, Berlin, pp. 127-142, 2006

[4] J. Carreira, H. Madeira y J.G. Silva, "Xception: A technique for Experimental Evaluation of Dependability in Modern Computers", IEEE

Transactions On Software Engineering, Vol 24, pp 125-135, Feb 1998.

[5] J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson, "GOOFI: Generic Object-Oriented Fault Injection Tool", Procs International Conference on Dependable Systems and Networks, 2001.

[6] J. Arlat, Y. Crouzet and JC. Laprie, "Fault Injection for dependability validation of fault-tolerant computing systems", Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S, Toulouse, Francia. 1989.

[7] Höxer H.-J., Sieh V.; Waitz M.:
"Fast Simulation of Stuck-At and Coupling Memory Faults Using FAUmachine"
In Supplement to Proc. HASE 2005: International Symposium on High Assurance Systems Engineering. 2005

[8] R. Maia, L. Enriques, R. Barbosa, D. Costa and H. Madeira, "Xception Fault Injection and Robusteness testing Framework: a case-study of testing RTEMS", WTF: VI Workshop de Testes e Tolerancia a Falhas, 2005

[9] J. Arlat, JC Fabre, M. Rodriguez and F. Salles, "MAFALDA: A series of prototype tools for the assessment of real time COTS microkernel-based systems" in Fault injection techniques and tools for embedded systems reliability evaluation, Boston: Kluwer academic, 2003.

[10] EADS-Astrium: www.astrium.eads.net

[11] RTEMS: Real Time Executive for Multiprocesor Systems. www.rtems.org

[12] TSIM: ERC32/Leon Simulator. www.gaisler.com

[13] Sparc Standards: www.sparc.org

Artículos anteriores publicados en RPM-AEMES

Nombre	Autor/es	Vol	Nº	Fecha
Estimación de variables en proyectos de desarrollo de software (PDS)	J. Aroba, I. Ramos, C. Riquelme	1	2	Agosto 2004
Una propuesta para la verificación de Requisitos basada en métricas	B. Bernárdez , A. Durán, M. Toro, M. Genero	1	2	Agosto 2004
Modelos segmentados de estimación del esfuerzo de desarrollo del Software: Un caso de estudio con la base de datos ISBSG	J. Cuadrado-Gallego, D. Rodríguez, M.A. Sicilia	1	2	Agosto 2004
Proceso y herramientas para la productividad en el aseguramiento y medición de calidad en desarrollos java	L. Fernández, P. Lara	1	2	Agosto 2004
Lecciones aprendidas al determinar el estado actual del área de proceso de gestión de requisitos utilizando el CMMI	J. Calvo-Manzano, G. Cuevas, T. San Feliu, A. Serrano, M. Arcilla	1	3	Diciembre 2004
Mejora de la calidad en desarrollos orientados a objetos utilizando especificaciones UML para la Obtención de precedencia de Casos de Prueba	L. Fernández, P. Lara, J. Cuadrado-Gallego	1	3	Diciembre 2004
Modelado dinámico y aprendizaje automático aplicado a la gestión de proyectos software	HERACLES	1	3	Diciembre 2004
Un procedimiento de medición de tamaño funcional: diseño y aplicación	N. Condori-Fernandez, S. Abraão, O. Pastor, S. Martí	1	3	Diciembre 2004
Estimación del esfuerzo de implantación en sistemas ERP	A. Cano, J. Tuya	2	1	Marzo 2005
Un enfoque de modelado y simulación para la comprensión del proceso de diseño centrado en el usuario	N. Hurtado, M. Ruíz, J. Torres	2	1	Marzo 2005
Estimación del esfuerzo de un proyecto software utilizando el criterio mdl-em y componentes normales n-dimensionales	Miguel Garre Rubio, Mario Charro Cubero	2	1	Marzo 2005
Optimización de Métrica Versión 3 en entornos orientados a objetos	J. L. López-Cuadrado, Á. García-Crespo, B. Ruiz-Mezcua, I. González-Carrasco	2	2	Agosto 2005
Experiencias de las administraciones públicas españolas en los procesos de gestión de requisitos y gestión de subcontratación	J.A. Calvo-Manzano, G. Cuevas, I. García, T. San Feliu, A. Serrano, F. Arboledas, F. Ruiz de Ojeda	2	2	Agosto 2005