pDI-Tools: Mecanismo de interposición dinámica de código

Descripción del proyecto

Gerardo García Peña

Jesús Labarta

Judit Giménez

1. Introducción

La instrumentación de código es una tecnología que permite analizar o variar el comportamiento de un programa durante su ejecución. Mediante este análisis se pueden descubrir muchas cosas del programa objetivo o incluso del sistema operativo que se está usando.

Hay muchísimas técnicas de instrumentación de código, desde las más sencillas como sería instrumentar estáticamente el mismo código fuente y luego compilar, hasta complejas técnicas de instrumentación en tiempo de ejecución (técnicas conocidas como instrumentación dinámica).

En el siguiente documento presentamos la implementación de un mecanismo para instrumentación dinámica basado en interceptar llamadas entre objetos compartidos dinámicamente (también conocidos como DSO[1]).

1.1. Introducción a la instrumentación

Cuando un programador escribe un software, éste tendría ya que ser correcto, debería estar validado y ser eficiente al máximo. No obstante la realidad es muy distinta: ordenadores complejos, recursos cuyo acceso no es uniforme, sistemas operativos de tiempo compartido, sistemas de comunicación con singularidades, miles de opciones de configuración (y de configuraciones posibles) y un largo etcétera, que añadido a la complejidad del software y/o de los datos con los que trabaja hacen que el comportamiento de un programa sea algo muy difícil de predecir. Y nos podemos encontrar en alguna de estas situaciones:

- El rendimiento del software desarrollado no se corresponde en absoluto con el rendimiento teórico calculado para la máquina.

- Cuesta demasiado imaginar la ejecución del programa y por lo tanto determinar dónde están exactamente los verdaderos cuellos de botella.
- Ocurren cosas raras en un determinado sistema y en otros no.
- Sabemos que los algoritmos que usamos ya son óptimos, pero aún así creemos que se puede mejorar la velocidad, ¿cómo averiguar dónde debemos optimizar?
- Aún peor: no se dispone del código fuente, sólo de una caja negra que está dando un rendimiento pésimo debido a alguna característica del sistema que no se consigue identificar.
- O simplemente: no se sabe que está pasando ahí dentro, pero pasa algo inesperado.

La instrumentación de código es una herramienta ideal en estas situaciones: permite analizar la ejecución del programa en los puntos clave sin llegar al extremo nivel de detalle e interferencia de las herramientas de depuración, sin sufrir la rigidez de las herramientas de "profiling". Además la instrumentación de código, al poder variar fácilmente la ejecución del programa, no se limita tan sólo al análisis de software: sus aplicaciones son virtualmente infinitas.

No obstante la flexibilidad ofrecida varía mucho según el tipo de instrumentación usada. A grandes rasgos hay los siguientes tipos:

- **Instrumentación estática.** Se puede realizar de dos maneras:
  - **Sobre el código fuente.** Consiste en insertar el código de instrumentación en el código fuente. Es necesario compilar de nuevo para obtener una versión instrumentada de la aplicación.
  - **Sobre el binario.** Consiste en usar una herramienta especial que inserta el código de instrumentación en el binario ya compilado. No es necesario recompilar, pero si puede ser necesario disponer de información de depuración o similar.
- **Instrumentación dinámica.** Consiste en alterar el código del ejecutable (o al menos las tablas de símbolos) en tiempo de ejecución para redirigir y controlar la ejecución del programa. No es necesaria la información de depuración aunque si se dispone de ella es posible aprovecharla.

Instrumentar estáticamente el código fuente es la técnica más potente ya que permite realizar cualquier tipo de estudio sobre el programa; en contrapartida, al introducir cambios en los fuentes, con la recompilación, a veces, se obtiene binarios muy distintos del original, obteniendo mediciones bastante trastocadas al instrumentar[2].

La instrumentación estática sobre el binario permite instrumentar con menos potencia que con la técnica anterior pero a cambio no altera tanto el resultado de la compilación e introduce menos ruido en la instrumentación. No obstante sigue alterando el ejecutable final (lo cual es un problema en caso de que el ejecutable vaya firmado y/o tenga algún mecanismo de protección). Otro problema es que esta técnica suele requerir información de la que no se suele disponer (código fuente, información de depuración, tablas de símbolos, ...).

La instrumentación dinámica se realiza en tiempo de ejecución y sin modificar el binario. Esta instrumentación se puede realizar de diversas formas: modificando el código en memoria[3] o trabajando directamente en las tablas de enlace dinámico del programa. La primera solución es algo lenta aunque permite instrumentar con mucha potencia el código, sin embargo necesita información detallada sobre el ejecutable. La segunda es el tipo de instrumentación más rápida, menos intrusiva y más ágil pero se limita tan sólo a las llamadas entre objetos compartidos (entre librerías y desde el ejecutable a librerías), pero a cambio no requiere ningún tipo de información de depuración.

1.2. Motivación

El CEPBA ha estado durante años desarrollando e investigando en herramientas de análisis del paralelismo en entornos de computación de alto rendimiento. Uno de sus productos, OMPItrace, tiene como objetivo instrumentar aplicaciones que usan las API MPI y OpenMP. No obstante hasta ahora sólo era posible instrumentar aplicaciones basadas en estas dos API en Irix sobre máquinas SGI® con ayuda de DITools y IBM AIX™ sobre hardware de IBM® con ayuda de DPCL. Sobre GNU/Linux™ sólo se podían instrumentar estáticamente aplicaciones MPI con ayuda de la API de "profiling" que el mismo MPI ofrece.

El proyecto surgió de la idea de portar DITools de Irix a otros sistemas operativos, y en especial a GNU/Linux™. Esta aplicación, también desarrollada por el CEPBA, implementa un mecanismo de interposición dinámica de código basado en interceptar las llamadas entre DSO. Portando esta herramienta se espera que en el futuro portar OMPItrace sea una tarea sencilla y rápida, a parte de eliminar dependencias de productos de terceros.

1.3. Objetivos

El objetivo es portar DITools, o en caso de ser imposible, implementar un software que permita interponer código dinámicamente de forma bastante similar. La herramienta tendría que cumplir los siguientes puntos y propiedades:

- **No se debe requerir nada más que un ejecutable para comenzar a trabajar.** La herramienta ha de permitir instrumentar un programa del cual tan sólo se tiene el binario.
- **Muy veloz y ligera.** El principal "target" de esta herramienta es la instrumentación de aplicaciones de cálculo intensivo. El impacto que debe tener sobre la velocidad de ejecución ha de ser mínimo. Durante la ejecución del programa que se instrumenta todo código de la librería debería tener un coste constante (O(1)); además de lo más breve posible.
- **Basada en estándares.** La herramienta ha de trabajar y respetar los estándares al máximo posible. De esta forma se asegura su robustez frente a nuevas versiones del sistema operativo "host" y frente a diferentes lenguajes y compiladores. Esta aplicación estará basada en el estándar ELF debido a que esta herramienta está pensada fundamentalmente para entornos Unix™ y similares.
- **Altamente portable.** En principio, la teoría dice que ciñiendose y respetando estrechamente los estándares al máximo, se puede obtener un software que funcione en cualquier plataforma realizando una serie de cambios mínimos. De antemano se sabe que esto es muy difícil, pero aún así el esfuerzo siempre facilitará el "porting" a otras plataformas.
- **Fácil de usar.** Debe ofrecer un control de errores adecuado y una interfaz sencilla de usar, apartando lo máximo posible al usuario de las internalidades del SO y del formato ELF. Además ha de ser resistente a errores, ya sean por parte del usuario o por sucesos inesperados provocados por la implementación de ELF del SO.
- **Debe ofrecer una API sencilla y potente.** La idea consiste en escribir un mecanismo de instrumentación mínimo, pero completo, para que el desarrollador goce de la máxima flexibilidad a la hora de crear una herramienta de instrumentación.
- **Una herramienta libre realizada con software libre.** Esta herramienta ha sido escrita desde cero con herramientas libres (GNU C Compiler, GNU Make, GNU/Linux™, DocBook, ...) y como respuesta se entregará como una aplicación libre. Debido a que en esencia es un DSO y se enlaza en tiempo de ejecución con la aplicación que se instrumenta se entregará bajo una licencia LGPL[4] la documentación bajo una licencia FDL[5].

- **Se pueden introducir cambios, pero en esencia el funcionamiento ha de ser muy similar al del DITools original.** Se ha de tener en cuenta que OMPItrace de SGI® confía en gran medida en el funcionamiento actual de DITools.

El proyecto se entregará como mínimo para GNU/Linux™ sobre la plataforma IA32. Inicialmente se dará sólo soporte a las versiones del kernel 2.4.x con GNU C Library 2.2.x, sin embargo también se intentará dar soporte a versiones 2.6.x con GNU C Library 2.3.x[6]

Además se presentarán una o dos versiones (no tan completas) para otras plataformas. Las posibles plataformas objetivo son:

- Solaris 8 sobre SPARC (en concreto sobre una UltraSPARC™ II)
- Irix sobre MIPS®

El objetivo de dar soporte a estas plataformas es demostrar que el código es altamente reciclable y muy fácil de adaptar a nuevos escenarios.

2. Presente y futuro

2.1. Trabajo realizado

La idea original era portar y adaptar DITools a GNU/Linux™.

Después de estudiar muy detenidamente la aplicación se concluyó que DITools estaba demasiado ligado a particularidades de Irix, y que el estándar ELF se trata como una simple particularidad más de Irix. Esto me llevo a reescribir totalmente la aplicación.

Para no caer en las mismas dependencias que el DITools original, lo siguiente fue recoger artículos, documentos, manuales y cualquier información sobre ELF, junto a su especificación. A continuación se estudió cómo se implementa en otros sistemas operativos, a qué nivel se suele respetar el estándar y aplicaciones que lo usan. El objetivo era ver si era posible emular el comportamiento de DITools intentando usar tan solo las estructuras de datos y mecanismos especificados por el estándar.

Se dedicó un buen tiempo a estudiar las limitaciones y decisiones de diseño de DITools para aprovechar toda la experiencia que DITools representa. Ha sido una buena inversión ya que ha servido para corregir cosas que en DITools hubieran sido difíciles de resolver sin reescribir grandes porciones del código. Además ha permitido dar una mejor estructura al programa mejorando su legibilidad, velocidad y facilidad a la hora de depurarlo.

En la actualidad se ha implementado una versión integra del software para GNU/Linux™ sobre plataformas IA32 y otra versión no tan completa para Solaris 8 para plataformas SPARC.

La versión para GNU/Linux™ sobre IA32 implementa tres tipos de interposiciones: reenlaces, redefiniciones y “callbacks”. Además funciona sobre versiones 2.4.x y 2.6.x del kernel. Además recientemente se ha dado soporte a las últimas versiones de GNU C Library[7].

La versión para Solaris 8 sobre SPARC implementa sólo dos de los tres tipos de interposiciones: reenlaces y redefiniciones. En el resto es tan funcional como la versión para GNU/Linux™ sobre IA32.

Se han añadido y potenciado los mecanismos que permiten definir los parámetros de funcionamiento del software. Antiguamente sólo se podían fijar mediante variables de entorno. Ahora, como novedad, se pueden ajustar mediante ficheros de configuración. Este método a mi parecer es claramente más limpio, potente y controlable.

Respecto al método de establecer las interposiciones de código es prácticamente idéntico al antiguo DITools: mediante una lista de interposiciones. La única diferencia estriba en que se puede usar más de un fichero de instrucciones y en consecuencia tener ficheros más pequeños y reciclables.

El entorno para desarrollar y mantener la documentación está establecido y aún en desarrollo. Se usa DocBook/XSL. El presente documento está escrito mediante este sistema.

El software usado para mantener el código, versiones y revisiones es el CVS. Los “bugs” y mejoras se controlan con ayuda de Bugzilla.

2.2. Trabajo pendiente

Empezar y terminar la documentación del proyecto. Se pretende incluir en él un manual de usuario, y otro de compilación e instalación (manual de administrador).

Todavía se deben resolver algunos fallos, inconsistencias y limitaciones y realizar más pruebas para asegurar un buen nivel de estabilidad.

Notas

1. Acrónimo para “Dynamic Shared Object”
2. Aun así existen técnicas basadas en el uso de macros y librerías compartidas para intentar insertar código alterando el binario resultante lo menos posible.
3. En este caso es similar a la instrumentación estática pero en tiempo de ejecución
4. Acrónimo para “The GNU Lesser General Public License (http://www.gnu.org/licenses/lgpl.txt)”.
5. Acrónimo para “The GNU Free Documentation License (http://www.gnu.org/licenses/fdl.txt)”.
6. Se han hecho algunos cambios en la carga y gestión de los DSO en memoria en estas últimas versiones que podrían provocar un mal funcionamiento.
7. Desde las versiones 2.2.x a las 2.3.x