(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 291 278**

(51) Int. Cl.:
G06F 11/34 (2006.01)
G06F 11/32 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Número de solicitud europea: **01305865 .6**
(86) Fecha de presentación : **06.07.2001**
(87) Número de publicación de la solicitud: **1172729**
(87) Fecha de publicación de la solicitud: **16.01.2002**

(54) Título: **Aparato y método para catalogar datos simbólicos para su empleo en análisis de funcionamiento de programas de ordenador.**

(30) Prioridad: **10.07.2000 US 613190**

(45) Fecha de publicación de la mención BOPI:
01.03.2008

(45) Fecha de la publicación del folleto de la patente:
01.03.2008

(73) Titular/es:
International Business Machines Corporation
New Orchard Road
Armonk, New York 10504, US

(72) Inventor/es: **Hussain, Riaz Y.;**
John, Chester Charles;
Levine, Frank Eliot y
Richardson, Christopher Michael

(74) Agente: **Elzaburu Márquez, Alberto**

DESCRIPCIÓN

Aparato y método para catalogar datos simbólicos para su empleo en análisis de funcionamiento de programas de ordenador.

Campo técnico del invento

El presente invento está dirigido a un aparato y método para catalogar datos simbólicos para su uso en análisis de rendimiento de programas de ordenador.

Antecedentes del invento

Al analizar y mejorar el rendimiento de un sistema de tratamiento de datos y las aplicaciones que se ejecutan dentro del sistema de tratamiento de datos, es útil saber qué módulos de software dentro de un sistema de tratamiento de datos están usando recursos del sistema. Una gestión y mejora efectivas de los sistemas de tratamiento de datos requieren saber cómo y cuándo se están usando los distintos recursos de sistema. Las herramientas de rendimiento son usadas para vigilar y examinar un sistema de tratamiento de datos para determinar el consumo de recursos cuando distintas aplicaciones de software se están ejecutando dentro del sistema de tratamiento de datos. Por ejemplo, una herramienta de rendimiento puede identificar la mayoría de los módulos e instrucciones ejecutados frecuentemente en un sistema de tratamiento de datos, o puede identificar aquellos módulos e instrucciones en un sistema de tratamiento de datos, o puede identificar aquellos módulos que asignan la mayor cantidad de memoria o realizan la mayor parte de las solicitudes de I/O. Las herramientas de rendimiento del hardware pueden estar integradas en el sistema o ser añadidas en un punto posterior en el tiempo.

Las herramientas de rendimiento del software también son útiles en los sistemas de tratamiento de datos, tales como sistemas de ordenador personal, que típicamente no contienen muchas, si las hay, herramientas de rendimiento de hardware integradas. Una herramienta de rastreo puede usar más de una técnica para proporcionar datos de rastreo que indican los flujos de ejecución para un programa que se ejecuta. Una técnica mantiene la pista de secuencias particulares de instrucciones registrando ciertos eventos cuando tienen lugar, denominada técnica de elaboración de perfiles basada en eventos. Por ejemplo, una herramienta de rastreo puede registrar cada entrada en y cada salida desde, un módulo, subrutina, método, función, o componente de sistema. Alternativamente, una herramienta de rastreo puede registrar al solicitante y las cantidades de memoria asignadas para cada solicitud de asignación de memoria.

Típicamente, un registro sellado en el tiempo, donde "tiempo" es definido como cualquier métrica creciente monitónicamente, tal como, número de instrucciones ejecutadas, es producido para cada uno de tales eventos. Pares correspondientes de registros similares a registros de entrada-salida también son usados para trazar la ejecución de segmentos de código arbitrario, que inician o terminan I/O o transmisión de datos, y para muchos otros eventos de interés.

Con el fin de mejorar el rendimiento de código generado por distintas familias de ordenadores, es a menudo necesario determinar donde está siendo gastando el tiempo por el procesador en la ejecución del código, siendo tales esfuerzos corrientemente conocidos en las técnicas de tratamiento de ordenador como situar "puntos calientes". Idealmente, a uno le gustaría aislar tales puntos calientes en la líneas de instrucción y/o de fuente de nivel de código con el fin de centrar la atención en áreas que podrían beneficiar la mayoría de las mejoras al código.

Otra técnica de rastreo implica muestrear periódicamente unos flujos de ejecución de programa para identificar ciertas localizaciones en el programa en las que el programa parece consumir grandes cantidades de tiempo. Esta técnica está basada en la idea de interrumpir periódicamente la aplicación o ejecución del sistema de tratamiento de datos a intervalos regulares, denominada generación o elaboración de perfiles basada en muestra. En cada interrupción, la información es grabada durante una longitud de tiempo predeterminada o para un número predeterminado de eventos de interés. Por ejemplo, el contador de programa del hilo o unidad básica de ejecución de OS/2 que se ejecuta actualmente puede ser grabado durante los intervalos. Estos valores puede ser resueltos contra un mapa de carga y una información de símbolo para el sistema de tratamiento de datos en tiempo de análisis, y puede obtenerse un perfil de en dónde se está gastando el tiempo a partir de este análisis.

Por ejemplo, aislar tales puntos calientes al nivel de instrucción puede identificar áreas significativas de código sub-óptimo que ayuda a los analistas de rendimiento a centrar su atención en mejorar el rendimiento del código "importante". Esto puede también ayudar a los escritores de compilador a centrar su atención en mejorar la eficiencia del código generado. Esto es especialmente cierto para el código "Jitted" (compilado Jit) (que se ha descrito posteriormente en esta aplicación). Otro uso potencial del detalle de nivel de instrucción es proporcionar orientación al diseñador de futuros sistemas. Tales diseñadores emplean herramientas de generación de perfiles para encontrar secuencias de código características y/o instrucciones únicas que requieren optimización para el software disponible para un tipo dado de hardware.

Las aplicaciones del sistema de tratamiento de datos son típicamente construidas con datos simbólicos y pueden incluso ser transportadas a dispositivos de clientes con datos simbólicos aún presentes en los módulos. Los datos simbólicos son, por ejemplo, representaciones alfanuméricas de nombres de módulo de aplicación, nombres de subrutina, nombres de función, nombres de variables, y similares.

La aplicación está comprendida de módulos escritos como código fuente en un lenguaje simbólico, tal como FORTRAN o C++, y a continuación convertida a un código máquina a través de la compilación del código fuente. El código máquina es la lengua nativa del ordenador. Con el fin de que un programa funcione, debe ser presentado al ordenador como instrucciones de máquina codificadas en binario que son específicas a ese modelo o familia de CPU.

El lenguaje de máquina dice al ordenador qué hacer y dónde hacerlo. Cuando un programador escribe: total = total + subtotal, esta declaración es convertida en una instrucción de máquina que dice al ordenador que añada el contenido de dos áreas de memoria donde TOTAL y SUBTOTAL están almacenados.

Como la aplicación es ejecutada como código máquina, los datos de rastreo de rendimiento del código máquina ejecutado, generados por las herramientas de rastreo, son proporcionados en términos de código máquina, es decir identificadores de procesos, direcciones, y similares. Así, puede ser difícil para un usuario de las herramientas de rastreo identificar los módulos, instrucciones, y similares, a partir de representaciones del código máquina puro en los datos de rastreo de rendimiento. Por ello, los datos de rastreo deben estar correlacionados con datos simbólicos para generar datos de rastreo que son fácilmente interpretados por un usuario de las herramientas de rastreo.

Los datos simbólicos con los que los datos de rastreo deben ser correlacionados pueden estar distribuidos entre una pluralidad de archivos. Por ejemplo, los datos simbólicos pueden estar presentes en archivos depurados, archivos de mapa, otras versiones de la aplicación, y similares. En los sistemas de herramienta de rendimiento conocido, con el fin de correlacionar los datos simbólicos con los datos de rastreo de rendimiento, la herramienta de rendimiento debe conocer las localizaciones de una o más fuentes de datos simbólicos y tener un método complejo de ser capaz de manejar redundancias en los datos simbólicos.

Además, tal correlación es típicamente realizada durante el post-tratamiento de los datos de rastreo de rendimiento. Así, se requiere una operación separada adicional para convertir datos de rastreo de rendimiento en representaciones simbólicas que pueden ser comprendidas por un analista de rendimiento.

La conversión de datos de rastreo de rendimiento en representaciones simbólicas es realizada en un momento que puede estar alejado del instante en que el rastreo de rendimiento es realizada. Como resultado, los datos simbólicos pueden no ser consistentes con la versión particular del programa de ordenador ejecutado durante la traza. Esto puede ser debido al hecho de que, por ejemplo, un versión más nueva de la aplicación fue ejecutada durante la traza y los datos simbólicos corresponden a una versión más antigua de la aplicación.

Esto puede ser especialmente cierto para aplicaciones cuyos datos simbólicos son mantenidos en una localización de proveedor estando el código de máquina distribuido a una pluralidad de clientes. En tal caso, el proveedor puede continuar actualizando los datos simbólicos, es decir crear nuevas versiones de la aplicación, pero falla en el suministro de versión más nueva de la aplicación a todos los clientes. En este escenario, si una traza de rendimiento tuviera que ser realizada, los datos simbólicos mantenidos por el proveedor pueden no ser la misma versión que el código de máquina en el que es realizada la traza de rendimiento.

El documento EP-0.767.430 describe un sistema de análisis de software para capturar etiquetas generadas por estados de etiquetas en código fuente instrumentalizado. El sistema de análisis del software incluye una prueba que vigila la dirección y la línea de transmisión de datos del sistema objetivo. Cuando es ejecutado un estado de etiqueta en el sistema objetivo, es escrita una etiqueta a una posición predeterminada en el espacio de dirección del sistema objetivo. La etiqueta contiene un valor de etiqueta que es indicativo de la posición en el código fuente del estado de etiqueta que genera la etiqueta. Vigilando la dirección predeterminada, la prueba es capaz de capturar etiquetas cuando están escritas en la línea de transmisión de datos del sistema objetivo. Instrumentalizando apropiadamente el código fuente, el análisis de software es capaz de realizar una variedad de funciones de análisis en tiempo real esencialmente, incluyendo cobertura de código, tiempo de ejecución de función y tarea, asignación de memoria, pares de llamada, y rastreo de programa.

Así, sería beneficioso tener un mecanismo por el que los datos simbólicos para una pluralidad de fuentes puedan ser combinados en una única fuente de datos simbólicos para una aplicación que sufre un análisis de rendimiento y que está siendo trazada. Sería además beneficioso tener un mecanismo para verificar los datos simbólicos cuando corresponden a la misma versión de la aplicación que sufre análisis de rendimiento y está siendo trazada. Adicionalmente, sería beneficioso tener un mecanismo que permita que la resolución simbólica sea realizada como una operación integrada al rastreo de rendimiento de la aplicación.

Descripción del invento

El presente invento proporciona un aparato y método para catalogar datos simbólicos para su uso en análisis de rendimiento de programas de ordenador. En particular, el presente invento proporciona un aparato y método de almacenar datos simbólicos para módulos ejecutables. Los datos simbólicos son usados cuando se realiza una rastreo de rendimiento.

El presente invento incluye un mecanismo por el que un archivo de símbolos de fusión es generado para un programa de ordenador, o aplicación, bajo traza. El archivo de símbolos de fusión contiene información útil para realizar la resolución simbólica de información de dirección en archivos de traza para cada caso de un módulo.

Durante el post-tratamiento de la información de rastreo generada por una rastreo de rendimiento de un programa de ordenador, la información simbólica almacenada en el archivo de símbolo de fusión es comparada con la información de rastreo almacenada en el archivo de rastreo. El post-tratamiento típicamente tiene lugar poco después de la traza o en algún instante alejado después de la traza del programa de ordenador.

La información de rastreo incluye información que identifica los módulos que son cargados durante el rastreo de la aplicación de ordenador. Esta información de rastreo y el archivo de símbolo de fusión son usados para producir informes. La información simbólica correcta en el archivo de símbolo de fusión para los módulos cargados es identificada basándose en un número de criterios de validación. Alternativamente, la información simbólica correcta en el archivo de símbolo de fusión para los módulos usados en el rastreo, o interrumpida en el caso de generar un perfil, es identificada basándose en un número de criterios de validación.

La información simbólica correcta para los módulos requeridos puede a continuación ser almacenada como una base de datos indexada que es indexada, por ejemplo por identificadores de proceso y dirección. La base de datos indexada de información simbólica puede ser almacenada como un archivo separado o como una parte separada de un archivo de rastreo para la aplicación de ordenador. Esta base de datos indexada puede a continuación ser usada para resolver información de dirección en información simbólica correspondiente cuando proporciona la información de rastreo para usar por un usuario, tal como un analista de rendimiento.

Breve descripción de los dibujos

El presente invento va a ser descrito a continuación sólo a modo de ejemplo, con referencia a los dibujos adjuntos, en los que:

La fig. 1 es un diagrama de bloques ejemplar de un sistema de tratamiento de datos distribuidos de acuerdo con el presente invento;

La fig. 2A es un diagrama de bloques ejemplar de un sistema de tratamiento de datos de acuerdo con el presente invento;

La fig. 2B es un diagrama de bloques ejemplar de un sistema de tratamiento de datos de acuerdo con el presente invento;

La fig. 3A es un diagrama de bloques que ilustra la relación de componentes de software que funcionan dentro de un sistema de ordenador que pueden poner en práctica el presente invento;

La fig. 3B es un diagrama de bloques ejemplar de una Máquina Virtual Java (JVM) de acuerdo con el presente invento;

La fig. 4 es un diagrama de bloques que representa componentes usados para procesos de generación de perfil en un sistema de tratamiento de datos;

La fig. 5 es una ilustración que representa distintas fases en la generación de perfiles de los procesos activos en un sistema operativo;

La fig. 6 es un diagrama ejemplar que ilustra una secuencia de tiempo de eventos de acuerdo con el presente invento;

La fig. 7 es un diagrama de flujo que representa una operación ejemplar de un programa de rastreo para generar grabaciones de rastreo a partir de procesos que se ejecutan en un sistema de tratamiento de datos;

La fig. 8 es un diagrama de flujo que representa una operación ejemplar de un gancho de rastreo de manipulador de interrupción del sistema;

La fig. 9 es un diagrama ejemplar que ilustra la generación de un archivo de símbolo de fusión de acuerdo con el presente invento;

La fig. 10A es un diagrama ejemplar que ilustra la organización de un archivo de símbolo de fusión de acuerdo con el presente invento;

La fig. 10B es un diagrama ejemplar de un archivo de símbolo de fusión;

La fig. 11 es un diagrama ejemplar de datos de rastreo de rendimiento que pueden ser almacenados como un archivo de rastreo o mantenidos en la memoria tampón de rastreo;

La fig. 12 es un diagrama ejemplar de un archivo de Entrada de Tabla de Módulo de acuerdo con el presente invento;

La fig. 13A es un diagrama ejemplar de una base de datos indexada de acuerdo con el presente invento;

La fig. 13B es un diagrama de flujo que esboza una operación ejemplar de un post-procesador para generar una base de datos indexada basándose en los datos MTE y el archivo de símbolo de fusión;

La fig. 14 es un diagrama de flujo que esboza una operación ejemplar del presente invento cuando genera una base de datos indexada de datos simbólicos;

La fig. 15 es un diagrama de flujo que esboza una operación ejemplar del presente invento cuando genera una base de datos indexada de datos simbólicos a partir de datos de rastreo de rendimiento almacenados en la memoria tampón de rastreo de una forma dinámica;

La fig. 16 es un diagrama de flujo que perfila una operación ejemplar del presente invento cuando verifica los datos simbólicos y la información de módulo cargada;

La fig. 17 es una diagrama de flujo que esboza una operación ejemplar del presente invento cuando obtiene la mejor entrada de módulo de coincidencia a partir del archivo de símbolo de fusión; y

La fig. 18 es un diagrama de flujo que esboza una operación ejemplar del presente invento cuando genera una presentación de datos de rastreo simbólicos;

La fig. 19 es un diagrama ejemplar de una parte de un Archivo de Bloque Básico típico (.bbf) para un programa de ordenador;

La fig. 20 es un diagrama ejemplar de una parte de un .bbf para un programa de ordenador de acuerdo con el presente invento; y

La fig. 21 es un diagrama de flujo que esboza una operación ejemplar de otra realización del presente invento.

Descripción detallada del invento

Con referencia a la fig. 1, se ha ilustrado una representación gráfica de un sistema de tratamiento de datos distribuidos en el que el presente invento puede ser puesto en práctica. El sistema 100 de tratamiento de datos distribuidos es una red de ordenadores en la que el presente invento puede ser puesto en práctica. El sistema 100 de tratamiento de datos distribuidos contiene una red 102, que es el medio usado para proporcionar enlaces de comunicaciones entre distintos dispositivos y ordenadores conectados juntos dentro del sistema 100 de tratamiento de datos distribuidos. La red 102 puede incluir conexiones permanentes, tales como hilo o cables de fibra óptica, o conexiones temporales hechas a través de conexiones telefónicas.

En el ejemplo representado, un servidor 104 está conectado a la red 102 junto con la unidad de almacenamiento 106. Además, los clientes 108, 110, y 112 también están conectados a una red 102. Estos clientes 108, 110, y 112 pueden ser, por ejemplo, ordenadores personales u ordenadores de red. Para los propósitos de esta aplicación, un ordenador de red es cualquier ordenador, acoplado a una red, que recibe un programa u otra aplicación desde otro ordenador acoplado a la red. En el ejemplo representado, el servidor 104 proporciona datos, tales como archivos de auto-arranque, imágenes operativas del sistema, y aplicaciones para el cliente 108-112. Los clientes 108, 110, y 112 son clientes para el servidor 104. El sistema 100 de tratamiento de datos distribuidos puede incluir servidores adicionales, clientes, y otros dispositivos no mostrados. En el ejemplo representado, el sistema 100 de tratamiento de datos distribuidos es Internet con la red 102 representando una colección mundial de redes y pasarelas que usan la sucesión TCP/IP de protocolos para comunicar entre sí. En el corazón de Internet hay una espina dorsal de líneas de comunicación de datos de alta velocidad entre nodos principales u ordenadores anfitriones, que consisten de miles de sistemas de ordenador comerciales, del gobierno, educativos, y otros, que encaminan datos y mensajes. Por supuesto, el sistema 100 de tratamiento de datos distribuidos puede también ser puesto en práctica como un número de diferentes tipos de redes, tal como, por ejemplo, Intranet o una red de área local.

Con referencia ahora a la fig. 2A, un diagrama de bloques de un sistema de tratamiento de datos que puede ser puesto en práctica como un servidor, tal como el servidor 104 en la fig. 1, está representado de acuerdo con el presente invento. El sistema 200 de tratamiento de datos puede ser un sistema de multiprocesador simétrico (SMP) que incluye una pluralidad de procesadores 202 y 204 conectados a la línea de transmisión 206 del sistema. Alternativamente, puede emplearse un sistema procesador único. También conectado a la línea 206 de transmisión del sistema hay un controlador de memoria/caché 208, que proporciona un enlace a la memoria local 209. El Puente 210 de Línea de Transmisión de I/O está conectado a la línea de transmisión 206 del sistema y proporciona un enlace a la línea de transmisión 212 de I/O. El controlador de memoria/caché 208 y el Puente 210 de Línea de Transmisión de I/O pueden estar integrados como se ha representado.

El puente 214 de línea de transmisión de interconexión de componente periférico (PCI) conectado a la línea de transmisión 212 de I/O proporciona un enlace a la línea de transmisión local 216 de PCI. Un módem 218 puede estar conectado a la línea de transmisión local 216 de PCI. Las implantaciones típicas de la línea de transmisión de PCI soportarán cuatro ranuras de expansión PCI o conectores añadidos. Los enlaces de comunicaciones a los ordenadores

de red 108-112 en la fig. 1 pueden ser proporcionados a través de un módem 218 y un adaptador de red 220 conectados a la línea de transmisión local 216 de PCI a través de placas añadidas.

Los puentes de línea de transmisión 222 y 224 de PCI adicionales proporcionan enlaces para líneas de transmisión adicionales 226 y 228 de PCI, desde las que pueden ser soportados módems o adaptadores de red adicionales. De esta manera, el servidor 200 permite conexiones a múltiples ordenadores de red múltiple. Un adaptador de gráficos 230 de memoria cartografiada y un disco duro 232 pueden también ser conectados a la línea de transmisión 212 de I/O como se ha representado, bien directa o indirectamente.

Los expertos en la técnica apreciarán que el hardware representado en la fig. 2A puede variar. Por ejemplo, otros dispositivos periféricos, tales como una unidad de disco óptico y similar pueden también ser usados además de o en lugar del hardware representado. El ejemplo representado no significa que implique limitaciones arquitectónicas con respecto al presente invento.

El sistema de tratamiento de datos representado en la fig. 2A puede ser, por ejemplo, un sistema IBM RISC/System 6000, un producto de International Business Machines Corporation, que hace funcionar el sistema operativo Ejecutivo Interactivo Avanzado (AIX).

Con referencia ahora a la fig. 2B, se ha ilustrado un diagrama de bloques de un sistema de tratamiento de datos en el que el presente invento puede ser puesto en práctica. El sistema de tratamiento de datos 250 es un ejemplo de un ordenador de cliente. El sistema 250 de tratamiento de datos emplea una arquitectura de línea de transmisión local de interconexión de componente periférico (PCI). Aunque el ejemplo representado emplea una línea de transmisión de PCI, pueden usarse otras arquitecturas de línea de transmisión tales como Micro Canal e ISA. El procesador 252 y la memoria principal 254 están conectados a la línea de transmisión local 256 de PCI a través de un Puente 258 de PCI. El Puente 258 de PCI puede también incluir un controlador de memoria integrado y una memoria caché para el procesador 252. Las conexiones adicionales para la línea de transmisión local 256 de PCI pueden ser hechas a través de una interconexión de componente directo o a través de placas añadidas. En el ejemplo representado, el adaptador 260 de red de área local (LAN), el adaptador 262 de línea de transmisión anfitrión de SCSI, y un enlace 264 de línea de transmisión de expansión están conectados a la línea de transmisión 256 de PCI por conexión de componente directo. En contraste, el adaptador de audio 266, el adaptador de gráficos 268, y el adaptador de audio/video (A/V) 269 son conectados a la línea de transmisión local 266 de PCI por placas añadidas insertadas en las ranuras de expansión. El enlace 264 de línea de transmisión de expansión proporciona una conexión para un teclado y un adaptador de ratón 270, un módem 272, y una memoria adicional 274. El adaptador 262 de línea de transmisión anfitrión de SCSI proporciona una conexión para una unidad de disco duro 276, una unidad de cinta 278, y un CD-ROM 280 en el ejemplo representado. Las implantaciones típicas de la línea de transmisión local de PCI soportarán tres o cuatro ranuras de expansión de PCI o conectadores añadidos.

Un sistema operativo funciona sobre el procesador 252 y es usado para coordinar y proporcionar control de distintos componentes dentro de un sistema de tratamiento de datos 250 en la fig. 2B. El sistema operativo puede ser un sistema operativo disponible comercialmente tal como JavaOS Para Negocios u OS/2, que están disponibles en Internacional Business Machines Corporation. El JavaOS es cargado desde un servidor en una red a un cliente de red y soporta programas y applets Java. Un par de características de JavaOS que son favorables para realizar rastreos con desarrollos de apilamientos, como se ha descrito más adelante, son que JavaOS no soporta paginación o memoria virtual. Un sistema de programación orientado al objeto tal como Java puede funcionar en unión con el sistema operativo y puede proporcionar llamadas al sistema operativo desde programas o aplicaciones Java que se ejecutan en el sistema de tratamiento de datos 250. Las instrucciones para el sistema operativo, el sistema operativo orientado al objeto, y aplicaciones o programas están situados en dispositivos de almacenamiento, tales como una unidad de disco duro 276 y pueden ser cargados en la memoria principal 254 para su ejecución por el procesador 252. Las unidades de disco duro están a menudo ausentes y la memoria es restringida cuando el sistema de tratamiento de datos 250 es usado como un cliente de red.

Los expertos en la técnica apreciarán que el hardware de la fig. 2B puede variar dependiendo de la puesta en práctica. Por ejemplo, otros dispositivos periféricos, tales como unidades de disco óptico y similar pueden ser usados además o en lugar del hardware representado en la fig. 2B. El ejemplo representado no significa que implique limitaciones arquitectónicas con relación al presente invento. Por ejemplo, los procesos del presente invento pueden ser aplicados a un sistema de tratamiento de datos de multiprocesador.

El presente invento proporciona un método y sistema para tratar datos de rastreo de rendimiento de aplicaciones de software. Aunque el presente invento puede funcionar en una variedad de plataformas de ordenador y sistemas operativos, puede también funcionar dentro de un entorno interpretativo, tal como un REXX, un Smalltalk, o un entorno de tiempo de ejecución Java, y similares. Por ejemplo, el presente invento puede funcionar junto con una máquina virtual Java (JVM) aún dentro de los límites de una JVM como se ha definido por especificaciones estándar de Java. Con el fin de proporcionar un contexto para el presente invento con respecto a un entorno interpretativo ejemplar, se han descrito aquí partes del funcionamiento de una JVM de acuerdo con especificaciones Java.

Con referencia ahora a la fig. 3A, un diagrama de bloques ilustra la relación de componentes de software que funcionan dentro de un sistema de ordenador que puede poner en práctica el presente invento. El sistema 300 basado en Java contiene un sistema operativo específico de plataforma 302 que proporciona soporte al hardware y al sistema

para ejecutar el software en una plataforma de hardware específica. La JVM 304 es una aplicación de software que se puede ejecutar en unión con el sistema operativo. La JVM 304 proporciona un entorno de tiempo de ejecución de Java con la capacidad para ejecutar una aplicación o applet de Java 306, que es un programa, servlet, o componente de software escrito en el lenguaje de programación Java. El sistema de ordenador en el que funciona la JVM 304 puede ser similar al sistema 200 de tratamiento de datos u ordenador 100 antes descrito. Sin embargo, la JVM 304 puede ser implantada en hardware dedicado sobre un denominado chip de Java, Java-en-silicio, o procesador Java con un núcleo picoJava integrado. En el centro de un entorno de tiempo de ejecución de Java está la JVM, que soporta todos los aspectos de entorno de Java, incluyendo su arquitectura, características de seguridad, movilidad a través de redes, e independencia de plataforma.

La JVM es un ordenador virtual, es decir un ordenador que está especificado de modo abstracto. La especificación define ciertas características que cada JVM debe poner en práctica, con algún intervalo de elecciones de diseño que puede depender de la plataforma en la que la JVM está diseñada para ejecutar. Por ejemplo, todas las JVM deben ejecutar códigos de byte de Java y puede usar un conjunto de técnicas para ejecutar las instrucciones representadas por los códigos de byte. Una JVM puede ser puesta en práctica completamente en software o parcialmente en hardware. Esta flexibilidad permite que JVM diferentes sean diseñadas para ordenadores principales y PDA.

La JVM es el nombre de un componente de ordenador virtual que realmente ejecuta programas Java. Los programas Java no son ejecutados directamente por el procesador central sino por la JVM, que es en sí mismo una pieza de software que se ejecuta en el procesador. La JVM permite que los programas Java sean ejecutados en una plataforma diferente en contraposición a sólo la plataforma para la que el código fue compilado. Los programas Java son compilados para la JVM. De esta forma, Java es capaz de soportar aplicaciones para muchos tipos de sistemas de tratamiento de datos, que pueden contener una variedad de unidades de tratamiento central y arquitecturas de sistemas operativos. Para permitir que una aplicación Java sea ejecutada en diferentes tipos de sistemas de tratamiento de datos, un compilador genera típicamente un formato de archivo de arquitectura neutra - el código compilado es ejecutable en muchos procesadores, dada la presencia del sistema de tiempo de ejecución Java.

El compilador de Java genera instrucciones de código de byte que no son específicas para una arquitectura de ordenador particular. Un código de byte es un código de máquina independiente generado por el compilador de Java y ejecutado por un interpretador de java. Un interpretador de Java es una parte de la JVM que alternativamente descodifica e interpreta un código de byte o códigos de byte. Estas instrucciones de código de byte están diseñadas para ser fáciles de interpretar en cualquier ordenador y fácilmente traducidas sobre la marcha a código de máquina nativo.

Una JVM debe cargar archivos de clase y ejecutar los códigos de byte dentro de éstos. La JVM contiene un cargador de clase, que carga archivos de clase desde una aplicación y los archivos de clase desde los enlaces de programación de aplicación Java (API) que son necesitados por la aplicación. La máquina de ejecución que ejecuta los códigos de byte puede variar a través de plataformas e implantaciones.

Un tipo de máquina de ejecución basada en un software es un compilador “justo en el momento” (JIT). Con este tipo de ejecución, los códigos de byte de un método son compilados a código de máquina nativo a la satisfacción con éxito de algún tipo de criterio para “jitting” (compilar Jit) un método. El código de máquina nativo para el método es a continuación cogido y vuelto a utilizar a la siguiente invocación del método. La máquina de ejecución puede también ser puesta en práctica en hardware e integrada en un chip de modo que los códigos de byte Java son ejecutados nativamente. Las JVM normalmente interpretan códigos de byte, pero las JVM también pueden usar otras técnicas, tales como compilación “justo a tiempo”, para ejecutar códigos de byte.

Interpretar el código proporciona un beneficio adicional. En vez de instrumentar el código fuente de Java, el intérprete puede ser instrumentado. Los datos de rastreo pueden ser generados a través de eventos y temporizadores seleccionados a través del intérprete instrumentado sin modificar el código fuente. La instrumentación de rastreo de rendimiento es descrita en mayor detalle más adelante.

Cuando una aplicación es ejecutada en una JVM que es puesto en práctica en software sobre un sistema operativo de plataforma específica, una aplicación Java puede interactuar con el sistema operativo anfitrión invocando métodos nativos. Un método Java es escrito en el lenguaje Java, compilado a códigos de byte, y almacenado en archivos de clase. Un método nativo es escrito en algún otro lenguaje y compilado al código de máquina nativo de un procesador particular. Los métodos nativos son almacenados en una biblioteca enlazada dinámicamente cuya forma exacta es específica de plataforma.

Con referencia ahora a la fig. 3B, un diagrama de bloques de una JVM está representado de acuerdo con una ilustración ejemplar del presente invento. La JVM 350 incluye un subsistema 352 cargador de clase, que es un mecanismo para cargar tipos, tales como clases y enlaces, nombres dados completamente cualificados. La JVM 350 también contiene áreas 354 de datos de tiempo de ejecución, una máquina de ejecución 356, un enlace de método nativo 358, y una gestión de memoria 374. La máquina de ejecución 356 es un mecanismo para ejecutar instrucciones contenido en los métodos de clases cargadas por el subsistema 352 cargador de clase. La máquina de ejecución 356 puede ser, por ejemplo, un intérprete de Java 362 o un compilador “justo a tiempo” 360. El enlace de método nativo 358 permite el acceso a recursos en el sistema operativo subyacente. El enlace de método nativo 358 puede ser; por ejemplo, un enlace nativo Java.

Las áreas 354 de datos de tiempo de ejecución contienen apilamientos 364 de métodos nativos, apilamientos Java 366, registros de PC 368, un área de método 370, y heap 372. Estas áreas de datos diferentes representan la organización de memoria necesitada por JVM 350 para ejecutar un programa.

Los apilamientos de Java 366 son usados para almacenar el estado de invocaciones del método Java. Cuando es lanzado un nuevo hilo o unidad básica de ejecución, la JVM crea un nuevo apilamiento de datos para el hilo. La JVM realiza solo dos operaciones directamente sobre los apilamientos de Java: empuja y pops cuadros. Un apilamiento de hilos de Java almacena el estado de invocaciones del método Java para el hilo. El estado de una invocación de método Java incluye sus variables locales, los parámetros con los que fue invocado, su valor de retorno, si lo hay, y cálculos intermedios. Los apilamientos de Java están compuestos de cuadros de apilamiento. Un cuadro de apilamiento contiene el estado de una única invocación de método Java. Cuando un hilo invoca un método, la JVM empuja un nuevo cuadro sobre el apilamiento Java del hilo. Cuando el método es completado, la JVM pops el cuadro para ese método y lo desecha.

La JVM no tiene ningún registro para soportar valores intermedios; cualquier instrucción de Java que requiere o produce un valor intermedio usa el apilamiento para soportar los valores intermedios. De esta forma, la instrucción de Java ajustada está bien definida para una variedad de arquitecturas de plataforma.

Los registros 368 de PC son usados para indicar la instrucción siguiente que ha de ser ejecutada. Cada hilo particular tiene su propio registro de pc (contador de programa) y apilamiento Java. Si el hilo está ejecutando un método JVM, el valor del registro de pc indica la siguiente instrucción a ejecutar. Si el hilo está ejecutando un método nativo, entonces el contenido del registro de pc están sin definir.

Los apilamientos 364 de método nativo almacenan el estado de invocaciones de métodos nativos. El estado de invocaciones del método nativo es almacenado de un modo dependiente de la puesta en práctica en apilamientos de métodos nativos, registros, u otras áreas de memoria dependientes de puesta en práctica. En algunas implantaciones de JVM, los apilamientos 364 de métodos nativos y los apilamientos Java 366 son combinados.

El área 370 del método contiene datos de clase mientras que un heap 372 contiene todos los objetos instanciados. La especificación de JVM define estrictamente tipos de datos y operaciones. La mayoría de las JVM elige tener un área de método y un heap, cada uno de los cuales son compartidos por todos los hilos que funcionan dentro de la JVM. Cuando la JVM carga un archivo de clase, analiza la información sobre un tipo a partir de los datos binarios contenidos en el archivo de clase. Coloca este tipo de información en el área de método. Cada vez que un caso de clase o matriz o disposición es creado, la memoria para el nuevo objeto es asignada desde el heap 372. La JVM 350 incluye una instrucción que asigna espacio de memoria dentro de la memoria para el heap 372 pero no incluye instrucciones para liberar ese espacio dentro de la memoria.

La gestión de memoria 374 en el ejemplo representado gestiona el espacio de memoria dentro de la memoria asignada al heap 370. La gestión de memoria 374 puede incluir un colector de desechos que automáticamente reclama la memoria usada por objetos que no están ya referenciados. Adicionalmente, un colector de desechos también puede mover objetos para reducir la fragmentación del heap.

El presente invento es igualmente aplicable bien a un entorno específico de plataforma, es decir, un entorno de aplicación de ordenador tradicional que carga módulos o métodos nativos, o un entorno independiente de plataforma, tal como un entorno interpretativo, por ejemplo, un entorno Java que carga clases, métodos y similares. Para propósitos de explicación de las características y ventajas del presente invento y para acentuar la capacidad del presente invento para funcionar en cualquier entorno, se describirán ejemplos del funcionamiento del presente invento en términos tanto de un entorno Java como de un entorno operativo de ordenador tradicional.

El presente invento proporciona un mecanismo por el que un archivo de fusión de datos simbólicos es generado. El presente invento también proporciona un mecanismo por el que trazas de rendimiento de aplicaciones, tales como aplicaciones Java, y la resolución simbólica puede ser realizada una en la que los datos simbólicos son verificados como siendo los datos simbólicos correctos para una resolución de direcciones incremental o bajo demanda, tal como con datos de rastreo de rendimiento. Además, el presente invento proporciona un mecanismo por el que una base de datos indexada de datos simbólicos es generada como un archivo separado o como una sección separada de un archivo de rastreo. Mientras el presente invento es aplicable a cualquier resolución incremental o bajo demanda de información simbólica, el presente invento será explicado en términos de una rastreo de rendimiento de un programa de ordenador para propósitos ilustrativos.

Con referencia ahora a la fig. 4, un diagrama de bloques representa componentes usados para realizar trazas de rendimiento de procesos en un sistema de tratamiento de datos. Un programa de rastreo 400 es usado para generar perfil de procesos 402. El programa de rastreo 400 puede ser usado para grabar datos durante la ejecución de un gancho, que es una pieza de código especializada en una situación específica en una rutina o programa en el que otras rutinas puede ser conectadas. Los ganchos de traza son típicamente insertados para el propósito de eliminar fallos, análisis de rendimiento, o mejorar la funcionalidad. Estos ganchos de traza son empleados para enviar datos de rastreo al programa de rastreo 400, que almacena los datos de rastreo en la memoria tampón 404. Los datos de rastreo en la memoria tampón 404, pueden ser almacenados subsiguientemente en un archivo para tratamiento posterior, o los datos de rastreo pueden ser procesados en tiempo real. Los datos de rastreo bien en la memoria tampón 404 o bien en el

archivo de rastreo, son a continuación procesados por el post-procesador 406 para generar una base de datos indexada de datos simbólicos para módulos cargados, como se ha descrito más completamente a continuación.

En un entorno que no es de Java, el presente invento emplea ganchos de rastreo que ayudan en la identificación de módulos que son usados en una aplicación baja rastreo. Con sistemas operativos Java, el presente invento emplea ganchos de rastreo que ayudan en la identificación de clases y métodos cargados.

Además, como las clases y módulos pueden ser cargados o descargados, estos cambios pueden también ser identificados usando datos de rastreo. Esto es especialmente relevante con sistemas de tratamiento de datos de "cliente de red", tales como los que pueden operar bajo Java OS, ya que clases y métodos compilados Jit pueden ser cargados y descargados más frecuentemente debido a la memoria y misión restringida como un cliente de red. Obsérvese que la información de carga y descarga de clase o módulo es también relevante en entornos de aplicación integrados, que tienden a ser restringidos de memoria.

Con referencia ahora a la fig. 5, un diagrama representa distintas fases en realizar un rastreo de rendimiento de la carga de trabajo que funciona en un sistema. Sujetos a restricciones de memoria, la salida de rastreo generada puede ser tan larga y detallada como el analista requiera para el propósito de perfilar un programa particular.

Una fase de inicialización 500 es usada para capturar el estado de la máquina cliente en el instante en el que es iniciada el rastreo. Estos datos de inicialización de rastreo incluyen registros de rastreo que identifican todos los hilos, todas las clases cargadas (módulos), y todos los métodos (secciones) existentes para las clases cargadas (módulos). Hay escritos registros desde los datos de rastreo capturados a partir de ganchos para indicar conmutadores de hilo, interrupciones, y carga y descarga de clases (módulos) y métodos (secciones) "jitted" (compilados Jit).

Cualquier clase (módulo) que es cargada tiene registros de rastreo que indican el nombre de la clase (módulo) y sus métodos (secciones). En el ejemplo representado, son usados cuatro ID de bits como identificadores para hilos, clases, y métodos. Estos ID están asociados con nombres que han sido emitidos en los registros de rastreo. Un registro de rastreo es escrito para indicar cuando ha sido escrita toda la información de inicio.

A continuación, durante la fase de perfilado 502, los registros de rastreo son escritos a una memoria tampón de rastreo o archivo de rastreo. En el presente invento, una memoria tampón de rastreo puede tener una combinación de tipos de registros, tales como las que se pueden originar desde un gancho de rastreo ejecutado en respuesta a un tipo particular de evento, por ejemplo, una entrada de método o salida de método, y los que se pueden originar desde una función que hace caminar el apilamiento ejecutado en respuesta a una interrupción del temporizador, por ejemplo, un registro de desarrollo de apilamiento, también llamado un registro de apilamiento de llamadas.

Por ejemplo, las operaciones siguientes pueden tener lugar durante la fase de perfilado si el usuario de la utilidad de perfilado tiene información de perfil basada en muestras requerida. Cada vez que tiene lugar un tipo particular de interrupción del temporizador, es escrito un registro de rastreo, que indica el contador de programa del sistema. Este contador de programa del sistema puede ser usado para identificar la rutina que es interrumpida. En el ejemplo representado, una interrupción del temporizador es usada para iniciar el agrupamiento de datos de rastreo. Por supuesto, pueden usarse otros tipos de interrupciones distintas de las interrupciones del temporizador. Las interrupciones basadas en un evento de monitor de rendimiento programadas u otros tipos de eventos periódicos pueden ser empleadas, por ejemplo.

En la fase de post-tratamiento 504, los datos recogidos en la memoria tampón de rastreo son procesados o enviados a un archivo de rastreo para post-tratamiento. En una configuración, puede enviarse el archivo a un servidor, que determina el perfil para los procesos en la máquina cliente. Por supuesto, dependiendo de los recursos disponibles, el post-tratamiento puede también ser realizado en la máquina cliente.

Con el presente invento, de acuerdo con una primera realización ejemplar, el post-tratamiento consiste en utilizar un archivo de símbolo de fusión para correlacionar datos simbólicos con datos de rastreo de rendimiento, es decir, para realizar la resolución simbólica. Esto puede ser hecho, bien con los datos de rastreo de rendimiento almacenados en la memoria tampón de rastreo o datos de rastreo de rendimiento en el archivo de rastreo. El post-tratamiento puede ser realizado como una operación incorporada de tal modo que el post-tratamiento es realizado inmediatamente después de que el rastreo de rendimiento es realizada, durante el rastreo de rendimiento en tiempo real, o en un instante alejado del instante en que es realizada el rastreo de rendimiento.

Como parte del proceso de resolución simbólico, los datos simbólicos para los módulos/procesos son verificados como si fueran los datos simbólicos correctos para las versiones de los módulos/procesos en los datos de rastreo de rendimiento. Esta verificación está basada en diferentes criterios incluyendo comprobación de suma, fechado, trayecto completamente cualificado, tamaños de segmento, y similares.

La resolución simbólica proporciona datos simbólicos para módulos/procesos cargados de la aplicación bajo rastreo. Como resultado de la resolución simbólica, es generada una base de datos indexada de los datos simbólicos para los módulos/procesos cargados. La base de datos indexada puede estar basada en los datos de rastreo de rendimiento en la memoria tampón de rastreo o los datos de rastreo de rendimiento en el archivo de rastreo, como se describirá en mayor detalle más adelante.

La fig. 6 es un diagrama ejemplar que ilustra la relación de tiempo de los distintos procesos empleados durante un rastreo de rendimiento de una aplicación y la generación subsiguiente de una base de datos indexada para módulos/procesos cargados. La fig. 6 supone que el post-tratamiento de los datos de rastreo de rendimiento es realizado en algún instante después de que el rastreo de rendimiento haya sido completada. Sin embargo, como se ha señalado antes, el post-tratamiento puede también ser realizado durante el rastreo de rendimiento de tal modo, que cuando los datos de rastreo de rendimiento son escritos a la memoria tampón de rastreo, el post-tratamiento es realizado en los datos de rastreo de rendimiento escritos. De este modo, la cantidad de tiempo necesario para completar el rastreo de rendimiento y el post-tratamiento es reducida.

Como se ha mostrado en la fig. 6, el rastreo de rendimiento es iniciada en el instante t_0 cuando la ejecución de aplicación ha sido iniciada. El rastreo de rendimiento termina en el instante t_1 cuando la ejecución de aplicación ha terminado.

Después del rastreo de rendimiento, en el instante t_2, es generado un archivo de símbolo de fusión de los datos simbólicos para la aplicación bajo rastreo. Mientras la fig. 6 muestra la generación del archivo de símbolo de fusión que es realizada después de que la aplicación de rastreo ha terminado, el invento no está limitado a esta ilustración ejemplar. En vez de ello, el archivo de símbolo de fusión puede ser generado antes de que se haya iniciado el rastreo de rendimiento o como parte de finalización del rastreo. Una ilustración ejemplar alternativa puede realizar la resolución simbólica en tiempo real (durante el rastreo) para la presentación concurrente de la información de rastreo.

En algún instante t_n después del rastreo de rendimiento y la generación del archivo de símbolo de fusión, los módulos/procesos usados durante el rastreo de rendimiento son determinados y es generada una base de datos indexada de los datos simbólicos para los módulos/procesos usados o cargados. Esta base de datos indexada puede ser generada como un post-tratamiento de los datos de rastreo de rendimiento en la memoria tampón de rastreo inmediatamente después de que haya terminado el rastreo de rendimiento. Alternativamente, la base de datos indexada puede ser generada como un post-tratamiento de los datos de rastreo de rendimiento almacenados en el archivo de rastreo en algún instante alejado a partir del rastreo de rendimiento real.

Con referencia ahora a la fig. 7, un diagrama de flujo representa una operación ejemplar de una herramienta de rastreo de rendimiento para generar grabaciones de rastreo a partir de módulos/procesos que se ejecutan en un sistema de tratamiento de datos. Los registros de rastreo pueden ser producidos por la ejecución de pequeñas piezas de código denominadas “ganchos”. Los ganchos pueden ser insertados de diferentes formas en el código ejecutado por procesos, incluyendo estáticamente (código fuente) y dinámicamente (a través de la modificación de un ejecutable cargado). La operación representada en la fig. 7 ha sido empleada después de que los ganchos de rastreo hayan sido ya insertados en el proceso o procesos de interés. La operación comienza asignando una memoria tampón (operación 700), tal como una memoria tampón 404 en la fig. 4. A continuación, en el ejemplo representado, los ganchos de rastreo son activados (operación 702), y el rastreo de los procesos en el sistema comienza (operación 704). Los datos de rastreo son recibidos desde los procesos de interés (operación 706). Este tipo de rastreo puede ser realizado durante fases 500 y/o 502, por ejemplo. Estos datos de rastreo son almacenados como registros de rastreo en la memoria tampón (operación 708).

Se ha determinado si el rastreo ha terminado (operación 710). El rastreo finaliza cuando la memoria tampón de rastreo ha sido llenada o el usuario detiene el rastreo a través de una orden y solicita que el contenido de la memoria tampón sea enviado al archivo. Por otro lado, cuando el rastreo ha terminado, el contenido de la memoria tampón es enviado a un archivo para post-tratamiento (operación 712). A continuación es generado un informa en post-tratamiento (operación 714) terminado la operación después de ello.

Aunque el ejemplo representado usa post-tratamiento para analizar los registros de rastreo, las operaciones del presente invento pueden ser usadas para procesar datos de rastreo en tiempo real dependiendo de la puesta en práctica. Si los datos de rastreo son procesados en tiempo real, el tratamiento de los datos de rastreo en la memoria tampón de rastreo comenzaría inmediatamente después de la operación 710 anterior. Tratando los datos de rastreo en tiempo real, los informes del elaborador de perfil puede ser presentado al mismo tiempo que la ejecución de programa.

Esta aproximación es especialmente útil para métodos compilados JIT. Un método compilado JIT es convertido desde códigos byte a código de máquina justo antes de que el programa esté funcionando. En el caso de Java, los métodos compilados JIT son convertidos desde código byte a código nativo. Así, la naturaleza dinámica de métodos compilados JIT puede ser acomodada tratando dinámicamente los datos de rastreo. Con referencia ahora a la fig. 8, un diagrama de flujo representa una operación ejemplar que puede ser usada durante una interrupción de un gancho de rastreo de manipulador. La operación comienza obteniendo un contador de programa (operación 800). Típicamente, el contador de programa está disponible en una de las áreas de apilamiento de programa salvado o guardado. Después de ello, se determina si el código que está siendo interrumpido es un código interpretado (operación 802). Esta determinación puede ser realizada determinando si el contador de programa está dentro de un intervalo de dirección para el intérprete usado para interpretar códigos de byte.

Si el código que está siendo interrumpido es interpretado, una dirección de bloque de método es obtenida para el código que está siendo interpretado. Un registro de rastreo es escrito entonces (operación 806). El registro de rastreo es escrito enviando los datos de rastreo a un programa de rastreo, tal como el programa de rastreo 400, que genera registros de rastreo para post-tratamiento en el ejemplo representado. Este registro de rastreo es denominado como un registro de interrupción, o un gancho de interrupción.

Este tipo de rastreo puede ser realizado durante la fase 502. Alternativamente, un proceso similar, es decir, determinar si el código que ha sido interrumpido es un código interpretado, puede ocurrir durante el post-tratamiento de un archivo de rastreo. En este caso, el último método interpretado que está siendo ejecutado es escrito siempre como parte del registro de rastreo.

Como se ha descrito más arriba, o bien antes, durante o después de que el rastreo de rendimiento es realizado, es generado un archivo de símbolo de fusión de los datos simbólicos para la aplicación bajo rastreo. La fig. 9 es una representación gráfica de la generación del archivo de símbolo de fusión de acuerdo con el presente invento para un entorno de ejecución de ordenador tradicional.

Como se ha mostrado en la fig. 9, el archivo 910 de símbolo de fusión está comprendido de datos simbólicos para módulos obtenidos a partir de archivos de mapa 920, archivos de ajuste 930, versiones no desnudadas de módulos 930, y otros archivos 940 de datos simbólicos. Estas fuentes de datos simbólicos puede ser almacenadas, por ejemplo, en la memoria local 209, el disco duro 232, uno o más de los dispositivos 276-282, o cualquier otro tipo de dispositivo de almacenamiento de datos. El archivo 910 de símbolo de fusión puede de modo parecido, ser almacenado en cualquiera de estos dispositivos de almacenamiento o similar.

El sistema de tratamiento de datos del presente invento es proporcionado con el trayecto completamente cualificado de las distintas fuentes de datos simbólicos y combina la información simbólica que describe distintos archivos ejecutables en un solo archivo de símbolo de fusión. Una realización ejemplar del formato de este archivo está descrita en la fig. 10A.

El archivo de símbolo de fusión resultante tiene una entrada (representada abstractamente por una entrada de Datos de Encabezamiento en el archivo de símbolo de fusión) para cada módulo. Puede haber múltiples entradas para módulos con el mismo nombre si, por ejemplo, existen múltiples versiones de un módulo en el sistema o si hay distintos módulos con nombres idénticos en diferentes trayectos en el sistema.

La fig. 10A es un diagrama ejemplar que ilustra la organización de un archivo de símbolo de fusión de acuerdo con el presente invento. Como se ha mostrado en la fig. 10A, el archivo de símbolo de fusión está organizado de una manera jerárquica. En la parte superior de la jerarquía está la información 1001 que identifica la plataforma particular en la que está situada la aplicación. Esta información incluye un encabezamiento, un indicador o bandera de sensibilidad de caso, un carácter de barra inclinada, y similares.

En el siguiente nivel de la jerarquía los elementos de fusión 1002 están identificados. Los elementos de fusión incluyen n números de módulos que están identificados por su nombre de módulo, que es el nombre de base sin extensiones de los módulos particulares en la aplicación.

Cada elemento de fusión puede representar 1 a n módulos distintos que ha sucedido que tienen el mismo nombre de base. Así, por ejemplo, durante la creación del archivo de símbolo de fusión, si un ejecutable, foo.exe, es encontrado y un archivo depurado correspondiente, foo.dbg, es también encontrado, los datos simbólicos procedente de ambos de estos archivos son combinados en una única imagen (descrita por un único elemento de dato 1002). Si, sin embargo, un ejecutable, foo.exe y un archivo depurado con el mismo nombre de base, foo.dbg, son encontrados pero se ha determinado que éstos no corresponden al mismo módulo (por ejemplo, si ellos contiene diferentes sumas de comprobación o sellos de tiempo, que indican posiblemente que corresponden a diferentes versiones del módulo), entonces son creadas dos imágenes distintas de los módulos (representadas por elementos de datos distintos 1002) con distinta información simbólica.

Estas imágenes del módulo son identificadas por encabezamientos de módulo 1003 que incluyen el trayecto de módulo, extensión, suma de control, y sello de tiempo. Cada imagen del módulo puede contener de 1 a n secciones, representando cada una colección de rutinas, una colección de elementos de datos que se pueden escribir o elementos de datos sólo de lectura, y similares.

Estas secciones son identificadas por un encabezamiento de sección 1004 que contiene el nombre de sección, desplazamiento, y longitud. Cada sección puede contener de 1 a n datos simbólicos 1005. Los datos simbólicos 1005 son identificados por el nombre simbólico, el desplazamiento desde la parte superior del módulo y/o una longitud.

La fig. 10B es una ilustración ejemplar de un archivo de símbolo de fusión de acuerdo con el presente invento. La fig. 10B supone un entorno no Java y está dirigida a módulos particulares de una aplicación. Sin embargo, el presente invento, como se ha señalado antes, es aplicable igualmente a un entorno Java.

Como se ha mostrado en la fig. 10B, el archivo 1000 de símbolo de fusión incluye un encabezamiento mergesym 1010, un identificador 1020 de elemento de fusión, y un nombre de módulo 1030. El encabezamiento mergesym 1010, el identificador de elemento de fusión 1020 y el nombre de módulo 1030 almacenan información acerca de cómo se ha generado el archivo de símbolo de fusión 1000. Además, estos elementos almacenan información sobre el sistema en el que se ha generado el archivo (tal como el número de procesadores o el sistema operativo en uso). El identificador de elemento de fusión 1020 forma un índice de nivel superior en el archivo de símbolo de fusión 1000 por el nombre de base.

El archivo de símbolo de fusión incluye además información perteneciente a cada módulo con el nombre de módulo. Así, en el ejemplo mostrado en la fig. 10B, dos módulos que tienen el nombre de módulo "foo" están presentes en el archivo de símbolo de fusión. Las entradas 1040 y 1050 para cada uno de los módulos están previstas en el archivo de símbolo de fusión.

Cada entrada 1040 y 1050 proporciona información 1060 perteneciente a la identificación de un módulo particular y los datos simbólicos 1070 asociados con el módulo. Los datos simbólicos están divididos en secciones que se pueden cargar que tienen encabezamientos de sección. Cada sección que se puede cargar tiene un nombre de sección, desplazamiento y longitud.

La información 1060 perteneciente a la identificación de un módulo particular incluye tal información como el trayecto completamente cualificado del módulo, la extensión del módulo, una suma de control, y un sello de tiempo para el módulo. Los datos simbólicos proporcionan el nombre de símbolo, desplazamiento y longitud para cada símbolo. Usando el desplazamiento y la longitud asociada con la sección y los datos simbólicos, puede determinarse la identidad exacta de los datos simbólicos y correlacionarla con direcciones en datos de rastreo de rendimiento.

Además de lo anterior, el archivo de símbolo de fusión puede incluir una medida de "confianza", o grado de calidad de los símbolos, para cada módulo. La medida de confianza puede ser, por ejemplo, un indicador de los tipos de datos simbólicos que fueron obtenidos durante el proceso de combinación. Por ejemplo, la medida de confianza puede proporcionar una indicación de si todos los símbolos exportables, símbolos internos y símbolos estáticos han sido obtenidos para el módulo particular. Esta medida de confianza puede ser referida a un usuario para su uso en la determinación de la calidad de la resolución simbólica de acuerdo con el presente invento.

Mientras los módulos mostrados en la fig. 10B tienen el mismo nombre de módulo, son módulos diferentes como resulta claro a partir de la información de módulo almacenada en el archivo de símbolo de fusión. Las entradas 1040 y 1050 representan diferentes módulos en los que el trayecto, la suma de control, el sello de tiempo, la longitud, y los datos simbólicos son diferentes para los dos módulos. Los propios módulos pueden ser dos versiones diferentes del mismo módulo, sin embargo. Por ejemplo, una versión posterior del módulo "foo.exe" en el directorio "C:\temp\" puede haber sido creada y almacenada en el directorio "C:\WINNT\.".

Cuando la suma de control y el sello de tiempo no están disponibles o el nombre del trayecto completamente cualificado no ha sido usado, los sistemas conocidos de rastreo de rendimiento no son capaces de discernir cual de los módulos es el módulo correcto para identificar los datos simbólicos asociados con los datos de rastreo de rendimiento. Los sistemas conocidos coinciden basándose en el nombre de base y son dependientes del usuario para asegurar que los símbolos que ellos proporcionan son para las versiones correctas de los módulos.

Por ejemplo, Windows 2000, disponible en Microsoft Corporation, requiere que el usuario especifique el nombre del trayecto completamente cualificado al archivo fuente y a la información simbólica con la excepción de algunos acuerdos fijos, tales como el directorio del sistema en los sistemas operativos Windows. Este directorio es identificado por el entorno SystemRoot variable. Así, una situación por defecto puede ser accedida por, por ejemplo, el trayecto "%SystemRoot%/Symbols/.". Así, si hay más de un módulo con el mismo nombre de módulo, bien como módulos diferentes, o bien versiones diferentes del mismo módulo, puede tener lugar un error en el que el módulo equivocado es usado para realizar la resolución simbólica.

Basarse solamente en el trayecto completamente cualificado no proporciona una solución a este problema porque:

1. el trayecto completamente cualificado puede no estar disponible en todos los sistemas;

2. algunas veces es conveniente generar símbolos fuera de una directorio diferente que aquél desde el cual el sistema carga los módulos; y

3. El trayecto completamente cualificado no es un criterio de seguridad contra fallos para la coincidencia. Si el rastreo es procesado con posterioridad en un instante alejado de la recogida de la propia información de rastreo, entonces es posible que un módulo haya sido modernizado a una versión más reciente en el tiempo medio. En este caso, los trayectos completamente cualificados coincidirían, pero uno no querría usar los símbolos desde el módulo en esa situación.

El presente invento proporciona un mecanismo que trabaja incluso en el caso en el que no es posible obtener información de rastreo que contenga el trayecto completamente cualificado. Además, el presente invento permite generar símbolos fuera de un directorio diferente que aquél desde el cual el sistema carga los módulos. Por ejemplo, el presente invento permite el tratamiento posterior de información de rastreo y generación de archivos de símbolo de fusión en un sistema que no es el sistema bajo ensayo. Además, el presente invento proporciona un mecanismo por el cual los datos simbólicos correctos son hechos coincidir con los datos de rastreo de rendimiento. El mecanismo hace uso de un número de comprobaciones para determinar si el módulo identificado en el archivo de símbolo de fusión es el mismo módulo que en los datos de rastreo de rendimiento.

La fig. 11 es un diagrama ejemplar de datos de rastreo de rendimiento. Los datos de rastreo de rendimiento 1100 en la fig. 11 pueden ser mantenidos en la memoria tampón de rastreo o pueden ser escritos a un archivo de rastreo

que sigue al rastreo de rendimiento. El archivo de rastreo puede estar almacenado, por ejemplo, en cualquiera de los dispositivos de almacenamiento 209, 232, 276-282, o similar. Los datos de rastreo de rendimiento incluyen los campos siguientes:

Campo 1: Código principal de gancho de rastreo;

Campo 2: Código menor de gancho de rastreo;

Campo 3: Sello de tiempo (superior 32 bits; inferior 32 bits);

Campo 4: No usado

Campo 5: Identificación de proceso (pid);

Campo 6: Dirección de carga de segmento;

Campo 7: Longitud de segmento;

Campo 8: Indicadores o banderas de segmento (Estos son indicadores que indican los niveles de permisión en las páginas en las que el segmento resulta cargado y similares);

Campo 9: Suma de control de módulo;

Campo 10: Sello de tiempo de módulo;

Campo 11: Nombre de segmento; y

Campo 12: Nombre del módulo.

Los datos de rastreo de rendimiento 1100 incluyen datos de rastreo de rendimiento para los ganchos de rastreo de Entrada de Tabla de Módulo (MTE) así como para los ganchos de rastreo del generador de perfil de tiempo (Tprof).

Los campos para ganchos de rastreo MTE en el archivo de rastreo están descritos anteriormente. Los datos de rastreo MTE están previstos en las entradas que tienen un código de principal de gancho de rastreo de 19 y un código menor de 38. Los códigos principal y menor 19 y 38 de gancho de rastreo son los códigos principal y menor que son usados en la ilustración ejemplar para indicar un gancho MTE. Puede usarse otros códigos sin ilustración que se salen del marco del presente invento.

Para un gancho de rastreo Tprof (el código principal 10 y el código menor 03), los campos serán ligeramente diferentes porque el campo 5 corresponderá a un contador de programa, el campo 6 corresponderá a un pid, el campo 7 corresponderá a un hilo id, el campo 8 corresponderá a un nivel privilegiado de código. El nivel de privilegio de código indica los privilegios que tiene el código de ejecución. Por ejemplo, el nivel de privilegios de código puede indicar si el código de ejecución está en un espacio de usuario o espacio kernel.

Los ganchos tprof contienen los datos de rastreo que son usados para generar el perfil del sistema bajo ensayo. En el instante del tratamiento posterior, las combinaciones de pid y dirección en los ganchos tprof están resueltos en símbolos. El post procesador combina la información MTE y el archivo de símbolo de fusión en una base de datos indexada. Cuando el post procesador encuentra un gancho tprof (o cualquier otro tipo de datos de rastreo que contiene información de dirección que necesita ser traducida en un símbolo) el post procesador busca la combinación de pid-dirección en la base de datos para obtener un símbolo correspondiente.

La información MTE incluye una entrada que representa la carga o descarga de cada sección en un módulo. Así, hay una entrada separada para cargar la sección de .text, cargar la sección de PAGE, y descargar la sección de .text (si cada una de las operaciones hubiera ocurrido) de C:\WINNT\foo.exe. En el ejemplo representado, la carga de estas secciones está mostrada en las líneas que comienzan con “19 38”. Ejemplos de entradas para descargar están mostrados en las líneas que comienzan con “19 39” y “19 44”. Las entradas de descarga que comienzan con “19 39” corresponden a un gancho de descarga estándar. Las entradas de descarga que comienzan con “19 44” corresponden a un gancho de descarga para un método compilado JIT.

Los datos de rastreo de gancho MTE en los datos de rastreo de rendimiento pueden ser almacenados como un archivo MTE. La fig. 12 proporciona un diagrama ejemplar de un archivo MTE 1200. Como se ha mostrado en la fig. 12, el archivo MTE 1200 contiene sólo las entradas MTE en los datos de rastreo de rendimiento y así, sólo identifica la carga y descarga de módulos.

En una realización preferida del presente invento, el archivo MTE 1200 está correlacionado con el archivo de símbolo de fusión para identificar los datos simbólicos para los módulos cargados. Sin embargo, la correlación de datos de rastreo de rendimiento con el archivo de símbolo de fusión puede ser realizada basándose en las entradas

MTE en los datos de rastreo de rendimiento en la memoria tampón de rastreo o el archivo de rastreo, tal como los datos de rastreo de rendimiento mostrados en la fig. 11.

Con el fin de verificar que la información de archivo de símbolo de fusión para el módulo corresponde al mismo módulo identificado en el archivo MTE, se han realizado varias comparaciones. Primero, es realizada una comparación de la suma de control y el sello de tiempo para el módulo. Si la suma de control y el sello de tiempo indicados en el archivo de símbolo de fusión corresponden a la suma de control y el sello de tiempo en el archivo MTE, a continuación se determinarán los identificadores de módulo para corresponder y los datos simbólicos en el archivo de símbolo de fusión son usados con la información de archivo MTE para generar información de módulo cargada.

Algunos archivos no contienen la información de suma de control y de sello de tiempo. Por ejemplo, los archivos de objeto Elf usados en Linux no contienen información de suma de control y sello de tiempo ni tampoco archivos de mapa. Así, para estos archivos, la comprobación de la suma de control y la comprobación del sello de tiempo tendrán normalmente un resultado negativo. Sin embargo, con los archivos de mapa, por ejemplo, otros archivos relacionados, tales como archivos .dbg, pueden ser usados en unión con los archivos de mapa para proporcionar la información necesaria para comprobar la validez de los archivos de mapa. Si la suma de control y el sello de tiempo no coinciden o no están disponibles, el trayecto completamente cualificado identificado en el archivo MTE es hecho coincidir con el trayecto completamente cualificado en el archivo de símbolo de fusión. Si hay una coincidencia, el módulo es verificado y los datos simbólicos en el archivo de símbolo de fusión correspondiente a la entrada de módulo verificada son usados para generar información de módulo cargada.

Si el trayecto completamente cualificado no coincide, se realiza una comparación de tamaños de segmento entre el archivo de símbolo de fusión y el archivo MTE. Así, por ejemplo, la longitud de segmento en el campo 7 del archivo MTE, es comparada con la longitud de segmento para el segmento, en el archivo de símbolo de fusión, del módulo identificado en el campo 11 del archivo MTE. Si la longitud de segmento corresponde, entonces ese segmento es "hecho coincidir". Cuando todos los segmentos han coincidido, el módulo es verificado y los datos simbólicos en el archivo de símbolo de fusión son usados para generar información en el módulo cargado.

Estas series de comparaciones puede ser realizadas para cada módulo en el archivo de símbolo de fusión que tiene el nombre de módulo apropiado. Así, por ejemplo, las comparaciones anteriores son realizadas para el primer módulo "foo" (Encabezamiento de Módulo (0)) y si no hay coincidencia, a continuación para el segundo módulo "foo" (Encabezamiento de Módulo (1)).

En una realización alternativa, cada comparación puede ser hecha independientemente de si una comparación anterior da como resultado un módulo verificado. Así, por ejemplo, la suma de control, el sello de tiempo, el trayecto completamente cualificado, y los tamaños de segmento son comparados para cada uno de los módulos "foo" y es elegido aquel con mejor correlación como el módulo correcto para ser usado para generar información de módulo cargada. Por ejemplo, si el primer módulo "foo" ha sido verificado basándose en los tamaños de segmento y el segundo módulo "foo" ha sido verificado basándose en el trayecto completamente cualificado, como el trayecto completamente cualificado tiene una mayor probabilidad de identificar la entrada de módulo correcta, el segundo módulo "foo" es elegido para generar información de módulo cargado.

Una vez que un módulo ha sido verificado, es creada una entrada de base de datos indexada basándose en los datos simbólicos de módulo verificado. Esta operación es realizada para cada entrada MTE en el archivo de rastreo de rendimiento o archivo MTE.

Las entradas de base de datos indexada pueden ser indexadas basándose en cualquier valor que se pueda buscar. Por ejemplo, la base de datos indexada es indexada basándose en el identificador de proceso (pid) y la dirección de carga de segmento, sin embargo, otros índices que pueden buscarse pueden ser usados son salirse del marco del presente invento.

Durante el post-tratamiento, cuando el post-procesador encuentra una entrada MTE en el archivo de rastreo de rendimiento o archivo MTE, dependiendo de la puesta en práctica particular, el segmento es hecho coincidir con un segmento correspondiente en el archivo de símbolo de fusión, como se ha descrito antes. Cuando es procesada la entrada MTE, se crea una entrada de base de datos indexada con el pid y la dirección de carga de segmento obtenidos desde el archivo de rastreo de rendimiento y el nombre de segmento es obtenido desde el archivo de símbolo de fusión.

La fig. 13A es una parte extraída ejemplar de un ejemplo de una base de datos indexada simplificada 1300 de acuerdo con el presente invento. Como se ha mostrado en la fig. 13A, las entradas en la base de datos indexada 1300 incluyen un índice 1310 (pid:dirección) y datos simbólicos correspondientes 1320, es decir los nombres de subrutina. Así, cuando se ha encontrado un pid:dirección particular en el archivo de rastreo de rendimiento, el pid:dirección puede ser convertido en una posición simbólica particular de una posición particular dentro de un archivo ejecutable. El propio símbolo corresponde a una subrutina (o método java).

Un segmento contiene normalmente múltiples subrutinas. Así, por ejemplo, si un registro tprof es encontrado en el pid 2 y la dirección 80298000, sería resuelto a 18000 bytes más allá del comienzo de la subrutina 2 en la versión de foo.exe en el directorio C:\\Temp\. Esto puede estar representado como; C:\\Temp.\foo.exe(subroutine2+0x18000).

Como se ha mencionado antes, la base de datos indexada 13000 es obtenida a través de un proceso de hacer coincidir combinaciones de pid:dirección obtenidas desde los datos de archivo MTE, tal como un archivo MTE 1200, con datos de sección en el archivo de símbolo de fusión, tal como el archivo de símbolo de fusión 1000. La fig. 13B es un diagrama de flujo que esboza una operación ejemplar de un post-procesador para generar la base de datos indexada 1300 basado en los datos MTE y el archivo de símbolo de fusión. Como se ha mostrado en la fig. 13B, la operación comienza con el post-procesador que encuentra un gancho MTE en los datos MTE (operación 1310). Los datos MTE identifican un pid y una dirección. El pid y la dirección son usados por el post-procesador para identificar un módulo y sección dentro del módulo en el archivo de símbolo de fusión (operación 1320).

El post-procesador calcula a continuación un desplazamiento de la dirección desde la parte superior del módulo que contiene la sección (operación 1330). Este desplazamiento es usado por el post-procesador para identificar los datos simbólicos para el símbolo (operación 1340). Los datos simbólicos resultantes son almacenados en la base de datos indexada en asociación con el pid:dirección (operación 1350). La base de datos indexada 1300 puede ser almacenada como un archivo separado en un medio de almacenamiento, tal como un disco duro 232 o disco 276, en memoria, tal como la memoria local 209 o la memoria 274, o puede ser almacenado como una parte separada del archivo de rastreo de rendimiento cuando el archivo de rastreo de rendimiento es escrito en un medio de almacenamiento. Por ejemplo, la base de datos indexada 1300 puede ser almacenada al final del archivo de rastreo de rendimiento de tal forma, que cuando se ha realizado el análisis de rendimiento por un usuario, la información del archivo de rastreo de rendimiento puede ser usada para identificar los módulos y segmentos particulares de la aplicación que ha sido cargada.

De este modo, un usuario de las herramientas de rastreo de rendimiento del presente invento puede realizar el análisis del rendimiento de una aplicación identificando los segmentos y módulos cargados particulares de la aplicación. Además, el usuario puede identificar la cantidad de tiempo de cálculo usado por los segmentos y módulos particulares para identificar partes de la aplicación que pueden ser optimizadas por la plataforma particular en la que está funcionando la aplicación.

La fig. 14 es un diagrama de flujo que esboza una operación ejemplar del sistema de tratamiento de datos de acuerdo con el presente invento cuando genera una base de datos indexada de datos simbólicos basada en un rastreo de rendimiento de un programa de ordenador, que es una aplicación. Mientras el diagrama de flujo muestra un orden particular en las operaciones, no se quiere decir que esté implicado ningún orden. En su lugar, muchas de las operaciones pueden ser realizadas en diferentes instantes durante el funcionamiento del sistema de tratamiento de datos, tal como la captura de datos simbólicos y el almacenamiento de datos simbólicos en un archivo de símbolo de fusión, que puede ser realizado antes, durante, o después de la ejecución de un rastreo.

Como se ha mostrado en la fig. 14, un rastreo del programa de ordenador es ejecutado (operación 1410) y es generado un archivo de rastreo (operación 1420). Como se ha descrito antes, este archivo de rastreo puede residir en la memoria tampón de rastreo o puede ser escrito en un dispositivo de almacenamiento.

La información de módulo cargado es generada y almacenada (operación 1430). Esto puede hacerse, por ejemplo, generando el archivo MTE que identifica sólo la carga y descarga de segmentos de módulo, como se ha descrito antes. Los datos simbólicos para el programa de ordenador son capturados (operación 1440) y almacenados en un archivo de símbolo de fusión (operación 1450).

Los datos simbólicos pueden ser capturados basándose en un usuario que identifica la posición de los archivos que contienen los datos simbólicos. Alternativamente, la captura de los datos simbólicos puede basarse en archivos que tienen el mismo nombre de archivo que el programa de ordenador bajo rastreo o almacenado en un directorio predefinido.

El archivo de símbolo de fusión es a continuación combinado con la información de módulo cargado para generar datos simbólicos de módulo cargado (operación 1460). Esta combinación puede incluir las comparaciones y verificación de módulos antes descritos. Los datos simbólicos de módulo cargado son a continuación indexados y almacenados como un archivo de base de datos indexada (operación 1470). El archivo de la base de datos indexada puede ser almacenado en memoria, como un archivo separado escrito en un dispositivo de almacenamiento, o como una sección separada del archivo de rastreo de rendimiento escrito en un dispositivo de almacenamiento, como se ha descrito antes.

El diagrama de flujo en la fig. 14 describe la operación del presente invento con el uso de un archivo de símbolo de fusión para alimentar datos simbólicos, sin embargo, el presente invento no está limitado al uso de un archivo de símbolo de fusión. En lugar de ello, cualquier fuente de datos simbólicos que puede ser verificada puede ser usada sin salirse del marco del presente invento.

La fig. 15 es un diagrama de flujo que esboza una operación ejemplar del sistema de tratamiento de datos cuando genera dinámicamente una base de datos indexada de datos simbólicos, basada en datos de rastreo de rendimiento almacenados en la memoria tampón de rastreo. Como con la fig. 14, aunque el diagrama de flujo muestra un orden particular de operaciones, no quiere decir que haya ningún orden implicado y muchas de las operaciones pueden ser realizadas en órdenes diferentes del que se ha mostrado.

Las operaciones mostradas en la fig. 15 son repetidas para nuevos datos de rastreo de rendimiento escritos en la memoria tampón de rastreo. De este modo, una base de datos indexada de datos simbólicos es creada dinámicamente cuando la aplicación está bajo rastreo.

Como se ha mostrado en la fig. 15, la operación se inicia con la realización de un rastreo de rendimiento del programa de ordenador (operación 1510) y con la generación de un archivo de rastreo (operación 1520). El archivo de rastreo es buscado para entradas del módulo cargado (operación 1530) y son obtenidos datos simbólicos para los módulos cargados (operación 1540). Los datos simbólicos son obtenidos preferiblemente a partir de un archivo de símbolo de fusión como se ha descrito antes, sin embargo, puede usarse cualquier fuente de datos simbólicos que pueda ser verificada sin salirse del marco del presente invento.

Una vez que se han obtenido los datos simbólicos para los módulos cargados, los datos simbólicos son almacenados como una sección separada del archivo de rastreo que contiene sólo los datos simbólicos para los módulos cargados (operación 1550). Estos datos simbólicos son a continuación indexados para generar una base de datos indexada de datos simbólicos para los módulos cargados como una sección separada del archivo de rastreo (operación 1560).

Así, usando cualquier operación descrita anteriormente, se obtiene una base de datos indexada de datos simbólicos para módulos cargados. Esta base de datos indexada, en una realización preferida, es obtenida reuniendo datos simbólicos a partir de una pluralidad de fuentes en un archivo de símbolo de fusión y comparando a continuación este archivo de símbolo de fusión con los datos de rastreo de rendimiento que están almacenados o bien en la memoria tampón de rastreo o bien en un archivo de rastreo en un dispositivo de almacenamiento. Los datos simbólicos coincidentes son a continuación escritos en una base de datos indexada en correspondencia con los datos de rastreo de rendimiento.

La fig. 16 es un diagrama de flujo que esboza una operación del presente invento cuando compara el archivo de símbolo de fusión con los datos de rastreo de rendimiento con el fin de verificar los datos simbólicos del módulo. Aunque la fig. 16 muestra un orden particular para las operaciones, muchas de las operaciones pueden ser realizadas en diferente orden. Así, por ejemplo, la verificación de tamaño de segmento puede ser realizada antes de la verificación de trayecto completamente cualificado, y de modo similar.

Como se ha mostrado en la fig. 16, la operación comienza con una verificación de la suma de control y el sello de tiempo para los datos simbólicos almacenados en el archivo de símbolo de fusión y los datos de rastreo de rendimiento (operación 1610). Se determina a continuación si hay una coincidencia de los datos simbólicos del archivo de símbolo de fusión y los datos de rastreo de rendimiento (operación 1620). Si hay una coincidencia, la operación continúa a la operación 1670, de otro modo, se determina si los datos simbólicos proceden de un módulo ejecutable (operación 1630). Esta determinación puede ser hecha, por ejemplo, determinando si la extensión del módulo como se ha previsto en el símbolo de fusión es ".exe".

Si los datos simbólicos no proceden de un ejecutable, la operación continúa a la operación 1660, de otro modo, se realiza una verificación del trayecto completamente cualificado del módulo (operación 1640). Se determina si la verificación de trayecto completamente cualificado indica que los datos simbólicos del módulo en el archivo de símbolo de fusión coinciden con los datos de rastreo de rendimiento (operación 1650). Si hay una coincidencia, la operación continúa a la operación 1670, de otro modo, es verificado el tamaño de segmento (operación 1660).

Se determina si el módulo ha sido verificado a través de una de las comprobaciones anteriores (operación 1670). Si no, es devuelto un mensaje de error (operación 1680). Si el módulo ha sido verificado, los datos simbólicos para el módulo en el archivo de símbolo de fusión coinciden con los datos de rastreo de rendimiento (operación 1690) y la operación termina.

Como se ha descrito antes, la verificación de datos simbólicos para un módulo con los datos de rastreo de rendimiento puede basarse en la primera entrada de módulo coincidente en el archivo de símbolo de fusión o puede implicar una determinación de "mejor coincidencia" de los datos simbólicos para el módulo. Esta determinación de "mejor coincidencia" puede incluir determinar una coincidencia para cada entrada de módulo en el archivo de símbolo de fusión para un nombre de módulo particular e identificar qué entrada de módulo es una mejor coincidencia. La mejor coincidencia puede ser determinada basándose en los atributos particulares que se han usado para establecer la coincidencia.

Así, los atributos pueden ser priorizados para proporcionar un medio para determinar la mejor coincidencia. Como ejemplo, la suma de control y el sello de tiempo pueden tener una prioridad más elevada, el trayecto completamente cualificado una segunda prioridad más elevada, y el tamaño de segmento una tercera prioridad más elevada.

La fig. 17 es un diagrama de flujo de una operación ejemplar del presente invento cuando determina una mejor coincidencia de los datos simbólicos en el archivo de símbolo de fusión con los datos de rastreo de rendimiento. Como se ha mostrado en la fig. 17, la operación comienza con la verificación de una primera entrada del módulo en el archivo de símbolo de fusión con la información de módulo cargado en los datos de rastreo de rendimiento (operación 1710). Se determina si hay una coincidencia de los datos simbólicos con los datos de rastreo de rendimiento (operación 1720). Si no la hay, es verificada la siguiente entrada del módulo en el archivo de símbolo de fusión (operación 1740). Si hay una coincidencia, se determina si la coincidencia está basada en la suma de control y el sello de tiempo (operación 1730). Si la coincidencia está basada en la suma de control y el sello de tiempo, entonces ésta es la mejor coincidencia

y la operación termina. Si la coincidencia no está basada en la suma de control y el sello de tiempo, la siguiente entrada del módulo en el archivo de símbolo de fusión es verificada (operación 1740) y se determina si la siguiente entrada del módulo es una mejor coincidencia que la primera entrada del módulo (operación 1750).

Como se ha descrito antes, esto puede basarse en un esquema de prioridad ajustado para los atributos particulares usados para verificar las entradas del módulo. Por ejemplo, puede ajustarse un indicador que indique un puntero para el módulo en al archivo de símbolo de fusión que ha coincidido y un número que indica el grado de confianza en la coincidencia. Los criterios de coincidencia pueden ser clasificados con suma de control y sello de tiempo en primer lugar, con trayecto completamente cualificado en segundo lugar, y con longitudes de sección en tercer lugar. Así, un 1, 2, o 3 serían registrados para indicar la calidad de la coincidencia. Esta coincidencia es comparada a continuación con una coincidencia subsiguiente y aquella que tenga la mayor medida de confianza es retenida. Este indicador de confianza puede ser traducido en un mensaje que es informado a un usuario.

Volviendo a la fig. 17, si la siguiente entrada del módulo es una mejor coincidencia, la siguiente entrada del módulo es seleccionada como el módulo coincidente en el archivo de símbolo de fusión (operación 1760) y el proceso vuelve a la operación 1730. Si el siguiente módulo no tiene una mejor coincidencia, se determina si hay más entradas de módulo a verificar (operación 1770). Si es así, el proceso vuelve a la operación 1740, de lo contrario, el proceso termina.

Como se ha descrito antes, el presente invento proporciona un mecanismo por el que se genera una base de datos indexada de datos simbólicos para módulos cargados. La base de datos indexada puede ser usada por una herramienta de análisis de tal modo que el usuario es presentado con una representación simbólica de los módulos cargados en vez de los identificadores de proceso y direcciones que pueden ser más difíciles de comprender.

La fig. 18 es un diagrama de flujo que esboza una operación ejemplar del presente invento cuando usa la base de datos indexada para proporcionar una representación simbólica de datos de rastreo de rendimiento para su análisis por un usuario. Como se ha mostrado en la fig. 18, la operación se inicia leyendo el archivo de rastreo (operación 1810). La información de identificador de proceso (pid) y de dirección son obtenidas a partir del archivo de rastreo (operación 1820).

La base de datos indexada es a continuación buscada para una entrada correspondiente al pid y a la dirección (operación 1830). Se determina si se ha encontrado una coincidencia (operación 1840). Si es así, los datos simbólicos correspondientes son usados de acuerdo con el archivo de rastreo (operación 1850). La forma particular en la que son usados los datos simbólicos dependerá de los propósitos y/o aplicaciones de análisis particular de estas aplicaciones. Después de ello, o si no hay coincidencia, se determina si se ha encontrado el final del archivo de rastreo (operación 1860). Si no, el proceso vuelve a la operación 1810, de lo contrario, el proceso termina.

Así, con el presente invento, se ha proporcionado un mecanismo por el que es generado un archivo de fusión de datos simbólicos. El presente invento también proporciona un mecanismo por el que rastreos de rendimiento de aplicaciones, tales como aplicaciones Java, y la resolución simbólica puede ser realizada en la que los datos simbólicos son verificados como si fueran los datos simbólicos correctos para los datos de rastreo de rendimiento. Además, el presente invento proporciona un mecanismo por el cual una base de datos indexada de datos simbólicos es generada bien como un archivo separado o bien como una sección separada de un archivo de rastreo.

Como se ha descrito antes el invento es capaz de proporcionar representaciones dinámicas del rastreo de rendimiento usando información de archivo MTE para identificar la carga y descarga de módulos. En algunos casos, es preferible tener representaciones estáticas del rastreo de rendimiento en diferentes momentos durante el rastreo.

Las herramientas de post-tratamiento actualmente conocidas hacen uso de una única representación estática para la dirección simbólica a información de nombre para el rastreo de un programa de ordenador. Esta representación estática es generada típicamente en dos partes. La primera parte es la generación de los datos MTE que representan los módulos cargados al inicio del rastreo. La segunda parte toma estos datos MTE y el símbolo para aquellos módulos cargados y crea la representación estática conocida por su extensión como un .bbf. Estos datos MTE ocurren típicamente como parte de la inicialización del rastreo inicial ("strace"). Alternativamente, los datos MTE pueden ser recogidos al final del rastreo. Obteniendo los datos MTE al comienzo del rastreo no se maneja el caso en el que los módulos son cargados durante el rastreo. Obtener los datos MTE al comienzo del rastreo no maneja el caso en que los módulos son cargados durante el rastreo. Obtener los datos MTE al final del rastreo no maneja el caso en que los módulos son descargados durante el rastreo después del rastreo y antes de que los datos MTE sean recogidos.

El .bbf es una imagen estática de qué módulos son cargados en un momento particular del rastreo y los símbolos correspondientes de los módulos cargados. El .bbf difiere del archivo de símbolo de fusión porque el archivo de símbolo de fusión contiene información simbólica para todos los módulos de un sistema de ordenador, el .bbf sólo contiene información simbólica para módulos cargados. El .bbf representa una colección de programas y otros códigos ejecutables cargados en todos los procesos del sistema de ordenador.

La fig. 19 es un ejemplo de una parte de un .bbf típico para un programa de ordenador. Como se ha mostrado, el .bbf tiene un formato de pid orientado en el que los métodos ejecutables están ordenados por direcciones dentro del pid, los segmentos están ordenados por direcciones, y los símbolos dentro de los métodos ejecutables están ordenados por direcciones dentro del segmento.

Como se ha mencionado antes, el .bbf, en herramientas de post-tratamiento conocidas, se ha generado bien al comienzo (strace) o bien al final del rastreo del programa de ordenador. Así, la única información que el análisis puede determinar a partir del .bbf son los métodos que fueron cargados en el instante en que el rastreo del programa de ordenador fue iniciado o en el instante de finalización del rastreo. Así, con las herramientas de post-tratamiento conocidas, no hay manera de proporcionar información simbólica para módulos que son cargados y descargados dinámicamente después de la iniciación del rastreo de comienzo ("strace") y antes de la finalización.

El presente invento usa el archivo de símbolo de fusión y la información de rastreo para generar múltiples archivos .bbf para determinar que módulos fueron cargados o usados durante el rastreo. La resolución simbólica puede ser realizada usando todos los archivos .bbf de tal modo, que si un módulo no es encontrado en un .bbf, puede encontrarse en uno de los otros archivos .bbf.

En esta segunda ilustración ejemplar del presente invento, el archivo de símbolo de fusión es utilizado por el post-procesador, junto con la información de archivo MTE, para generar representaciones estáticas, por ejemplo, archivos .bbf, del rastreo del programa de ordenador. Estas representaciones estáticas, en la ilustración ejemplar, son creadas al comienzo ("strace") y final del rastreo. De este modo, la representación estática que comienza incluye los módulos cargados cuando el programa de ordenador es iniciado. La representación estática final identifica los módulos que fueron cargados durante el rastreo. A partir de esta información, es posible identificar módulos que han sido cargados al comienzo del rastreo y descargados. También es posible identificar módulos que fueron cargados dinámicamente durante el rastreo.

La diferencia entre un módulo cargado y un módulo usado es que un módulo puede ser cargado y nunca usado, es decir, nunca referenciado por los registros de rastreo. Esto sucede cuando un módulo no es ejecutado durante un tiempo suficiente para ser interrumpido por un tic del generador de perfiles del temporizador. Similarmente, un módulo puede haber sido cargado en un punto, usado, y a continuación descargado. Construyendo una representación estática del rastreo al comienzo y final del rastreo, puede determinarse que módulos fueron cargados en la inicialización, cuáles de estos módulos fueron usados, cuáles de estos módulos no fueron usados, y cuáles de estos módulos fueron cargados durante el rastreo y usados o no usados. Por ejemplo, si un módulo tiene una entrada en la representación estática generada al comienzo del rastreo, pero no tiene una entrada en la representación estática al final del rastreo, puede determinarse que el módulo fue cargado, usado y a continuación descargado. Similarmente, si la representación estática al final del rastreo tiene una entrada para un módulo que no tiene una entrada en la representación estática generada al comienzo del rastreo, el módulo debe haber sido cargado durante el rastreo y no usado.

El archivo MTE contiene información relativa a los módulos cargados. Usando el archivo de símbolo de fusión, de la manera descrita anteriormente con respecto a la resolución simbólica de funcionamiento para generar una base de datos indexada, puede realizarse la resolución simbólica de información de direcciones para los módulos cargados. Por ejemplo, la información de módulo en el archivo de rastreo/memoria tampón de rastreo es usada para identificar módulos en el archivo de símbolo de fusión para generar por ello una base de datos indexada de información simbólica. Esta base de datos indexada de información simbólica puede a continuación ser usada junto con el archivo MTE para generar un archivo .bbf, usando desplazamientos simbólicos desde el comienzo de los módulos, y similares, para un caso particular en el rastreo del programa de ordenador. La generación del archivo .bbf puede ser realizada tanto al comienzo como al final del rastreo, por ejemplo.

Así, usando el archivo MTE y el archivo de símbolo de fusión, puede generarse una representación estática del rastreo del programa de ordenador para diferentes instantes durante el rastreo, por ejemplo al comienzo y al final del rastreo. Esta información puede ser a continuación almacenada y usada para proporcionar representaciones simbólicas de los datos rastreados. Debido a que las representaciones estáticas sólo representan módulos cargados, y debido a que las representaciones estáticas han sido generadas para un número finito de puntos en el tiempo en el rastreo, la cantidad de información almacenada para la resolución simbólica puede ser minimizada.

Así, con el presente invento, durante el post-tratamiento, el post-procesador puede hacer uso del rastreo de comienzo .bbf para realizar la resolución simbólica de información de dirección. Si no puede encontrarse un módulo en el rastreo de comienzo .bbf, es decir, el módulo ha sido cargado dinámicamente durante el rastreo, el .bbf generado al final del rastreo puede ser usado para realizar la resolución simbólica. Así, generando múltiples archivos .bbf durante la ejecución de un rastreo de un programa de ordenador, puede realizarse la resolución simbólica de módulos cargados dinámicamente.

A partir del ejemplo del .bbf mostrado en la fig. 19 anterior, es evidente que, cuando hay muchos métodos, puede haber un montón de información duplicativa almacenada en los archivos .bbf. Por ejemplo, si un módulo tiene una pluralidad de segmentos, la información del módulo para el segmento se repetirá para cada proceso que tenga el mismo módulo.

El presente invento, en otro ejemplo, elimina una gran mayoría de esta información duplicativa en sistemas en los que el trayecto completamente cualificado para un módulo es conocido durante el rastreo identificando cada módulo cargado por su trayecto completamente cualificado durante la inicialización del rastreo de comienzo (strace). La fig. 20 es un diagrama ejemplar de una parte de un .bbf de acuerdo con el presente invento.

Como se ha mostrado en la fig. 20, el .bbf es un trayecto orientado. Es decir, los módulos son identificados por el trayecto completamente cualificado. El .bbf es construido en un trayecto orientado de forma que sólo tiene una entrada para cada módulo. Esto puede hacerse ajustando el pid para que el módulo sea un carácter comodín, por ejemplo, "????". Esta entada de comodín indica que la entrada del módulo en el .bbf es independiente del pid. Con módulos que son independientes del pid, la dirección de partida es ajustada a cero y todas las direcciones para los segmentos y la información simbólica son direcciones relativas. Es decir, las direcciones son relativas a la dirección de comienzo de cero. Cuando se conoce el trayecto completamente cualificado de un módulo, el .bbf es construido con el "????" para el pid. Cuando el trayecto del módulo completamente cualificado no es conocido, el pid es identificado en el .bbf.

Cuando la resolución simbólica es realizada usando el .bbf de acuerdo con esta otra ilustración ejemplar del presente invento, el módulo puede ser "buscado" en el .bbf por su trayecto completamente cualificado. Después de ello, si no hay coincidencia basada en el trayecto completamente cualificado, el módulo puede ser "buscado" basándose en el pid. Mientras el pid es ajustado a un comodín para los módulos en el .bbf del presente invento, cada entrada de módulo en el .bbf será comprobada para ver si hay coincidencia basada en el tamaño de segmento, información de dirección simbólica, y similares, en una forma similar a como se ha descrito antes con vistas a la verificación de módulos que usan el archivo de símbolos de fusión.

Así, con el presente invento, la cantidad de información almacenada en el .bbf es minimizada mientras que se mantiene aún la capacidad para buscar el .bbf para entradas de módulo coincidentes durante la resolución simbólica.

Es común para un sistema operativo cargar segmentos, o secciones, de un módulo gradual. Así, la ejecución de un segmento particular de un módulo puede ocurrir antes de que todos los segmentos para el módulo sean cargados. Además, algunos segmentos del módulo nunca pueden ser cargados durante el rastreo o sus registros de rastreo que han sido cargados pueden estar no disponibles. El presente invento proporciona un mecanismo por el que una resolución simbólica para los segmentos de un módulo puede ser realizada sin requerir que el módulo completo sea cargado o los registros de rastreo para el módulo entero estén disponibles.

Como se ha mencionado antes, y como se ha mostrado en el archivo de rastreo de muestra y el archivo MTE en las figs. 11 y 12, el presente invento puede escribir información redundante a los datos de rastreo. Esta información redundante incluye, por ejemplo, la suma de control del módulo, el sello de tiempo del módulo y el trayecto completamente cualificado del módulo.

Debido a que no es posible saber *a priori* el orden en el que los segmentos serán cargados, cada uno de los registros de rastreo contiene información suficiente para que el post-procesador construya una imagen del módulo. Esta información es usada para hacer coincidir el segmento en la grabación de rastreo con la sección del módulo en el archivo de símbolo de fusión.

Con el fin de hacer coincidir un segmento representado por un registro de rastreo con una sección particular dentro de un módulo representado en el archivo de símbolo de fusión, se han considerado los criterios siguientes. Si tanto el nombre de segmento en el registro de rastreo como el nombre de sección en el archivo de símbolo de fusión no son nulos y coinciden, a continuación el segmento y la sección son una coincidencia. Si tanto el nombre de segmento como el nombre de sección son nulos y sólo hay un segmento en el módulo, entonces ese debe ser el segmento identificado en el registro de rastreo. Si tanto el nombre de segmento como el nombre de sección son nulos y las direcciones coinciden, entonces el segmento y la sección son una coincidencia. Si tanto los nombres son nulos como los tamaños en bytes coinciden, entonces el segmento y la sección son una coincidencia.

Una vez que el segmento y la sección son hechos coincidir, la información simbólica puede ser escrita en la base de datos indexada en la forma antes descrita. Así, el presente invento proporciona unos medios para realizar la resolución simbólica de segmentos dentro de módulos incluso cuando el módulo entero no ha sido cargado o los registros de rastreo para todos los segmentos de un módulo no están disponibles.

En las ilustraciones ejemplares antes descritas, la información de rastreo es escrita al archivo de rastreo, o archivo MTE, cuando los segmentos de módulos son cargados y descargados. Así, hay entradas redundantes para cada segmento que pueden ser eliminadas y aún ser capaces de realizar la resolución simbólica. Eliminando estas entradas redundantes, el tamaño del archivo de rastreo puede ser ampliamente reducido.

En algunos sistemas, uno puede ser capaz de actualizar el kernel para añadir un estado de campo asociado con los módulos cargados. Alternativamente, puede usarse una extensión kernel para proporcionar este estado de campo.

Con el presente invento, cuando un módulo es cargado, el kernel actualizado tiene asociado con el módulo un indicador de rastreo "usado (o referenciado)" que está asociado con el pid del módulo. Cuando el módulo es cargado, este indicador es borrado a cero. Así, el indicador indica que el módulo ha sido cargado pero aun no ha sido usado o referenciado.

Como un ejemplo cuando se ejecuta una aplicación que genera un perfil de tiempo, cuando un gancho de rastreo de perfil de tiempo es encontrado durante el rastreo, el indicador "usado" para el módulo interrumpido en el pid interrumpido es ajustado a uno por el programa de rastreo. Cuando el módulo es descargado, el kernel modificado

puede comprobar el indicador "usado" (denominado UF más adelante) para determinar si ha sido ajustado. Si el UF es ajustado, el programa de rastreo puede emitir registros de rastreo MTE asociados con el módulo anterior para descargar el módulo. Similarmente al final del rastreo todos los módulos cargados pueden ser comprobados y cada uno con el UF ajustado pueden tener registros de rastreo MTE escritos. Durante el post-tratamiento, la información simbólica es recogida para todos los módulos que tienen registros de datos MTE; es decir, para todos los módulos para los que existen referencias de datos de rastreo.

Aunque tratar posteriormente el rastreo e intentar realizar una resolución simbólica, los registros de rastreo son procesados secuencialmente, buscando la primera entrada MTE después de la referencia de rastreo. A partir de esta entrada MTE y la información simbólica en el archivo de símbolo de fusión, la dirección a resolución de nombre puede ser determinada. En un ejemplo alternativo, en la primera referencia, los datos MTE para el módulo de referencia son escritos antes de escribir el registro de rastreo. Con esta aproximación el post-procesador no tiene que buscar los datos MTE después de la referencia de rastreo debido a que han sido ya leídos por el post-procesador.

En otro ejemplo alternativo, puede crearse una tabla hash (tabla de estructura de datos que asocia claves con valores) siendo una clave a la tabla hash el pid. Los datos en la tabla hash pueden incluir una lista de módulos asociados con el pid. La tabla hash puede incluir indicadores que indican un número de estados del módulo que incluyen si los datos de rastreo para el módulo han sido ya escritos, si el módulo ha sido referencia antes, si el módulo ha sido cargado y usado, y similares. Estos indicadores pueden ser usados de la misma manera que los UF descrito antes. En otras palabras, basado en los ajustes de estos indicadores, puede determinarse si escribir o no los datos de rastreo en un archivo de rastreo. De este modo, el mismo tipo de esquema que se ha descrito antes puede ser desarrollado por una extensión kernel que no modifica las estructuras de datos kernel.

Así, en este otro ejemplo del presente invento, el número de registros de rastreo es reducido y así, es minimizado el archivo de rastreo. Minimizando el tamaño del archivo de rastreo, es también reducido el tiempo de post-tratamiento. Además, escribiendo los datos de rastreo de módulo antes de escribir el registro de rastreo, la magnitud de búsqueda realizada por el post-procesador es también reducida, haciendo por ello el post-tratamiento más rápido.

La fig. 21 es un diagrama de flujo que esboza una operación ejemplar del presente invento de acuerdo con este otro ejemplo. Como se ha mostrado en la fig. 21, la operación comienza con la inicialización del archivo de rastreo al comenzar un rastreo de un programa de ordenador. Durante la inicialización, los datos de módulo cargado iniciales, por ejemplo, datos MTE, son escritos en el archivo de rastreo para aquellos procesos y métodos que son cargados al comienzo del rastreo (operación 2110). Una tabla hash es construida para todos los id de proceso cargados actualmente y los módulos asociados (operación 2120). Esto implica crear una entrada en la tabla hash para cada pid y colgar del pid una lista de módulos asociados con el pid. La información de módulo, tal como dirección y, opcionalmente, la longitud del módulo, pueden ser incluidas en la tabla hash.

Cada módulo en la tabla hash incluye además un indicador de datos de rastreo que indica si los datos de rastreo para el módulo han sido escritos en el archivo de rastreo o memoria tampón de rastreo. Al producirse la inicialización, como todas las entradas en la tabla hash corresponde a procesos y métodos que han sido escritos en el archivo de rastreo o memoria tampón de rastreo en la operación 2110, los indicadores de datos de rastreo son ajustados a verdadero (operación 2130).

El rastreo es ejecutado a continuación (operación 2140) y se determina si se ha encontrado un gancho de rastreo MTE durante el rastreo (operación 2150). Si no se encuentra, se determina si se ha encontrado un gancho de perfil (operación 2160). Si no se ha encontrado un gancho de perfil, el rastreo continúa volviendo a la operación 2140. Si se encuentra un gancho de perfil, el módulo en el que es encontrado el gancho de perfil, es buscado por pid y dirección de módulo en la tabla hash (operación 2170). Se determina entonces si el indicador de datos de rastreo para el módulo ha sido ajustado a falso, es decir, no se han escrito los datos de rastreo en el archivo de rastreo o memoria tampón de rastreo (operación 2180). Si el indicador de datos de rastreo es falso, los datos de rastreo son escritos en el archivo de rastreo y el indicador de datos de rastreo es ajustado a verdadero (operación 2190). Después de ello, o si el indicador de datos de rastreo es verdadero en la operación 2180, los datos de rastreo del gancho de perfil son escritos en el archivo de rastreo (operación 2200). El rastreo puede entonces continuar si se desea (operación 2300).

Si en la operación 3150 es encontrado un gancho MTE, la tabla hash es explorada para el pid asociado con el gancho MTE (operación 2210). Se determina si hay presente una entrada para el pid en la tabla hash (operación 2220). Si no lo hay, es añadida una entrada a la tabla hash que usa el pid (operación 2230).

Después de ello, o si se ha encontrado una entrada basada en el pid en la tabla hash, la tabla hash es explorada para la dirección de módulo asociada con el gancho MTE (operación 2240). Se determina si se ha encontrado una entrada de módulo basada en la dirección de módulo (operación 2250). Si no se encuentra, es añadida una entrada de módulo a la tabla hash usando la dirección de módulo (operación 2260). La adición de la entrada de módulo es hecha en asociación con el pid en el gancho de MTE.

Si se ha encontrado una entrada de módulo en la operación 2250, se determina si es necesario un solapamiento parcial o completo de la entrada del módulo (operación 2270). Si es así, la entrada de módulo es solapada con la información de módulo asociada con el gancho de MTE (operación 2280). Para un solapamiento parcial, esto puede incluir ajustar la longitud de entradas de módulo existentes asociadas con el pid y entonces insertar la información

de módulo asociada con el gancho MTE. Para un solapamiento completo de módulo, este puede incluir borrar una entrada de módulo existente y reemplazarla con una nueva entrada de módulo basada en la información de módulo asociada con el gancho MTE.

Puede ocurrir un solapamiento parcial o completo cuando, por ejemplo, un proceso es detenido y un nuevo proceso es creado usando el mismo pid que el del proceso previo. En tal caso, la entrada de módulo puede ser solapada con una nueva entrada de módulo. En un ejemplo alternativo el archivo de rastreo puede contener una entrada de rastreo separada que indica la detención de un proceso y la creación de un nuevo proceso que usa el mismo pid. Después de ello, cualesquiera otras referencias al pid serán resueltas usando la nueva entrada del módulo.

Después de ello el indicador de datos de rastreo para el módulo es ajustado a falso (operación 2290). A continuación se determina si hay que continuar el rastreo (operación 2300). Si es así, el proceso vuelve a la operación 2140. De otro modo, el proceso termina. Durante el post-tratamiento, los datos MTE para el o los archivos son leídos y usados en el orden de secuencia en el tiempo.

Como se ha descrito antes, la funcionalidad de la tabla hash para almacenar indicadores de estado y similares, puede ser realizada actualizando el kernel para añadir un indicador de estado asociado con módulos cargados o proporcionando una extensión kernel. Similarmente, un almacenamiento local de proceso puede ser utilizado para mantener este indicador de estado. Similarmente, un almacenamiento local de proceso puede ser utilizado para mantener este indicador de estado. Alternativamente, un bloque de control de proceso del sistema operativo puede ser modificado directamente para mantener este indicador de estado.

Es importante observar que aunque el presente invento ha sido descrito en el contexto de un sistema de tratamiento de datos que funciona completamente, los expertos en la técnica apreciarán que los procesos del presente invento son capaces de ser distribuidos en forma de un medio legible por ordenador de instrucciones y una variedad de formas y que el presente invento se aplica igualmente independientemente del tipo particular de medios que soporten la señal realmente usados para llevar a la práctica la distribución. Ejemplos de medios legibles por ordenador incluyen medios de tipo grabable tales como un disquete, una unidad de disco duro, una RAM, y CD-ROM y medios de tipo de transmisión tales como enlaces de comunicaciones digitales y analógicas.

La descripción del presente invento ha sido presentada con propósitos ilustrativos y descriptivos, pero no está destinada a ser exhaustiva o limitada al invento en la forma descrita. Serán evidentes muchas modificaciones y variaciones para los expertos en la técnica. La realización fue elegida y descrita a fin de explicar mejor los principios del invento, la aplicación práctica, y de habilitar a personas que no son expertas en la técnica para comprender el invento para distintas realizaciones con distintas modificaciones cuando son adecuadas al uso particular considerado.

En resumen, el invento proporciona un aparato y método para catalogar datos simbólicos para usar en análisis de rendimiento de un programa de ordenador. El aparato y método almacenan datos simbólicos para módulos cargados durante un rastreo de rendimiento o poco tiempo después del mismo y utiliza los datos simbólicos almacenados cuando realiza un análisis de rendimiento en un instante posterior. Un archivo de símbolo de fusión es generado para un programa de ordenador, o aplicación, bajo rastreo. El archivo de símbolo de fusión contiene información útil en la realización de la resolución simbólica de información de dirección en archivos de rastreo para cada caso de un módulo. Durante el post-tratamiento de la información de rastreo generada por un rastreo de rendimiento de un programa de ordenador, la información simbólica almacenada en el archivo de símbolo de fusión es comparada con la información de rastreo almacenada en el archivo de rastreo. La información simbólica correcta en el archivo de símbolo de fusión para módulos cargados es identificada basada en varios criterios de validación. La información simbólica correcta para los módulos cargados puede ser almacenada a continuación como una base de datos indexada que es usada para resolver información de dirección en información simbólica correspondiente cuando se proporciona la información de rastreo a una pantalla de presentación para usar por un usuario.

REIVINDICACIONES

1. Un método de identificación de datos simbólicos para módulos cargados asociado con un programa de ordenador bajo análisis de rendimiento, que comprende: leer los datos de rastreo para un módulo cargado, comprendiendo dicho rastreo al menos un atributo para identificar el módulo cargado; hacer corresponder al menos un atributo con el correspondiente atributo o atributos cargados identificando el módulo cargado comprendido de un archivo de símbolo, en el que el archivo de símbolo comprende datos simbólicos combinados a partir de una pluralidad de fuentes de datos simbólicos asociados con el programa de ordenador y en el que cada atributo tiene una prioridad asociada; y usar la prioridad para determinar los datos simbólicos en el archivo de símbolos que mejor coinciden con los datos de rastreo.

2. El método según la reivindicación 1ª, en el que el atributo comprende uno o más de una suma de control, un sello de tiempo, un trayecto completamente cualificado, y un tamaño de segmento.

3. El método según la reivindicación 1ª o 2ª, en el que los datos de rastreo son leídos desde una memoria tampón de rastreo.

4. El método según cualquiera de las reivindicaciones 1ª a 3ª, en el que los datos de rastreo son leídos desde un archivo de rastreo escrito en un dispositivo de almacenamiento.

5. El método según cualquiera de las reivindicaciones 1ª a 4ª, en el que las operaciones de lectura, y coincidencia son realizadas dinámicamente cuando los datos de rastreo son escritos en una memoria tampón.

6. El método según cualquiera de las reivindicaciones 1ª a 4ª, en el que las operaciones de lectura, y coincidencia son realizadas en un instante alejado de cuando los datos de rastreo son escritos en un archivo de rastreo.

7. El método según cualquier reivindicación precedente, en el que la operación de coincidencia comprende la operación de comparar una suma de control y un sello de tiempo en los datos de rastreo con una suma de control y un sello de tiempo en los datos simbólicos en el archivo de símbolos.

8. El método según la reivindicación 7ª, en el que la operación de coincidencia comprende la operación de comparar un trayecto completamente cualificado en los datos de rastreo con un trayecto completamente cualificado en los datos simbólicos, si la suma de control y el sello de tiempo en los datos de rastreo no coinciden con la suma de control y el sello de tiempo en los datos simbólicos o la suma de control y el sello de tiempo en los datos de rastreo no están disponibles.

9. El método según la reivindicación 8ª, en el que la operación de coincidencia comprende la operación de comparar una longitud de segmento en los datos de rastreo con una longitud de segmento en los datos simbólicos, si el trayecto completamente cualificado en los datos de rastreo no coincide con el trayecto completamente cualificado en los datos simbólicos de módulo.

10. El método según cualquier reivindicación precedente, en el que el atributo comprende suma de control y sello de tiempo, un trayecto completamente cualificado, y un tamaño de segmento, y en el que la suma de control y el sello de tiempo tienen una mayor prioridad y el tamaño de segmento tiene una menor prioridad.

11. El método según cualquier reivindicación precedente, en el que los datos de rastreo incluyen información redundante que identifica un módulo para cada segmento del módulo.

12. El método según la reivindicación 11ª, en el que la información redundante incluye al menos una suma de control de módulo, sello de tiempo de módulo y trayecto completamente cualificado.

13. El método según cualquier reivindicación precedente, que comprende además las operaciones de: correlacionar los datos simbólicos con los datos de rastreo para generar datos correlacionados; y presentar los datos correlacionados.

14. Un aparato para identificar datos simbólicos para módulos cargados asociado con un programa de ordenador bajo análisis de rendimiento, que comprende: un procesador acoplable a un dispositivo de almacenamiento de datos de rastreo y un dispositivo de almacenamiento de datos simbólicos, en el que el procesador es operable para: leer datos de rastreo para un módulo cargado desde el dispositivo de almacenamiento de datos de rastreo, comprendiendo dichos datos de rastreo al menos un atributo para identificar el módulo cargado; hacer corresponder al menos un atributo con el correspondiente atributo o atributos cargados identificando el módulo cargado comprendido de un archivo de símbolo, en el que el archivo de símbolo comprende datos simbólicos combinados a partir de una pluralidad de fuentes de datos simbólicos asociados con el programa de ordenador y en el que cada atributo tiene una prioridad asociada; y usar la prioridad para determinar los datos simbólicos en el archivo de símbolos que mejor coinciden con los datos de rastreo.

15. Un programa de ordenador que comprende medios de código de programa destinados a realizar todas las operaciones de cualquiera de las reivindicaciones 1ª a 13ª en el que dicho programa es ejecutable en un ordenador.

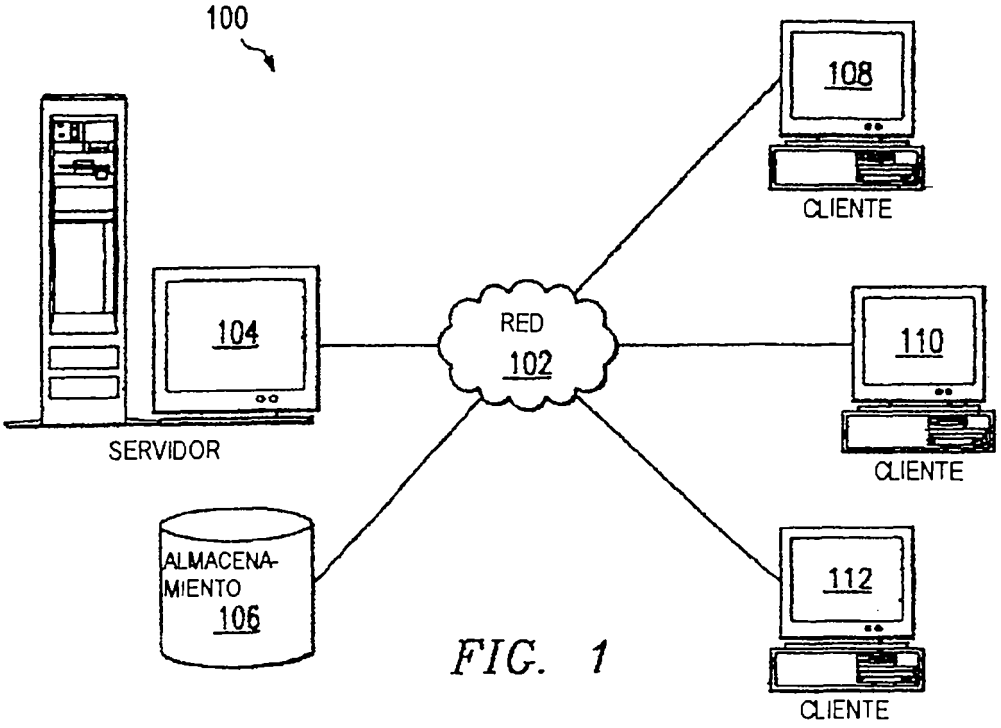
100
108
CLIENTE
104
SERVIDOR
RED
102
110
CLIENTE
ALMACENA-
MIENTO
106
112
CLIENTE

FIG. 1

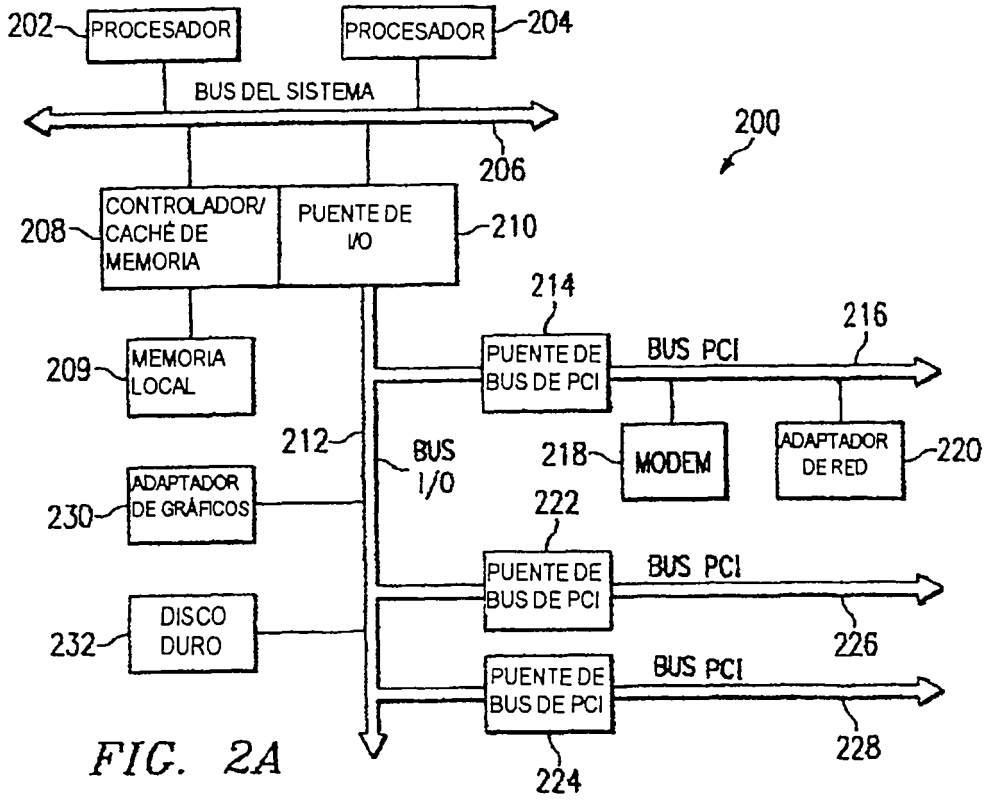
202
PROCESADOR
PROCESADOR
204
BUS DEL SISTEMA
206
200
208
CONTROLADOR/
CACHÉ DE
MEMORIA
PUENTE DE
I/O
210
214
216
209
MEMORIA
LOCAL
PUENTE DE
BUS DE PCI
BUS PCI
212
BUS
I/O
218
MODEM
ADAPTADOR
DE RED
220
230
ADAPTADOR
DE GRÁFICOS
222
PUENTE DE
BUS DE PCI
BUS PCI
232
DISCO
DURO
226
PUENTE DE
BUS DE PCI
BUS PCI
228
224

FIG. 2A

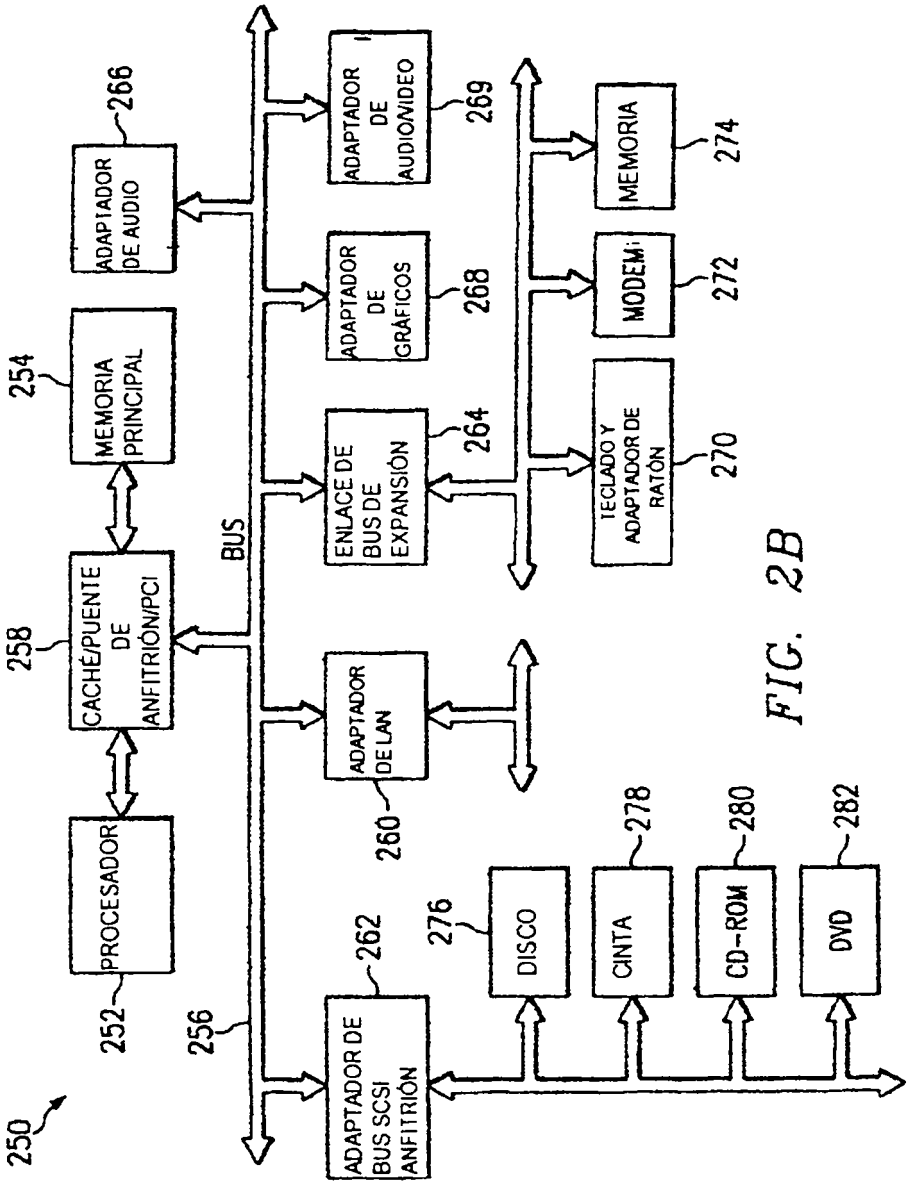
250
PROCESADOR
252
CACHÉ/PUENTE DE ANFITRIÓN/PCI
258
MEMORIA PRINCIPAL
254
ADAPTADOR DE AUDIO
266
BUS
256
ADAPTADOR DE BUS SCSI ANFITRIÓN
262
ADAPTADOR DE LAN
260
ENLACE DE BUS DE EXPANSIÓN
264
ADAPTADOR DE GRÁFICOS
268
ADAPTADOR DE AUDIO/VIDEO
269
DISCO
276
CINTA
278
CD-ROM
280
DVD
282
TECLADO Y ADAPTADOR DE RATÓN
270
MODEM
272
MEMORIA
274

FIG. 2B

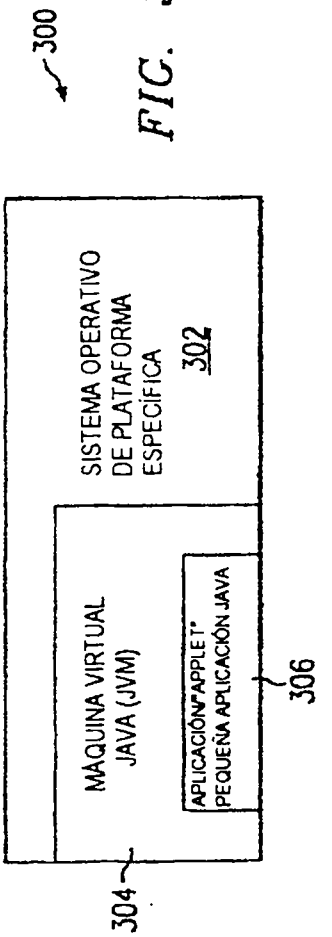
300
SISTEMA OPERATIVO DE PLATAFORMA ESPECÍFICA
302
MÁQUINA VIRTUAL JAVA (JVM)
APLICACIÓN "APPLET" PEQUEÑA APLICACIÓN JAVA
306
304

FIG. 3A

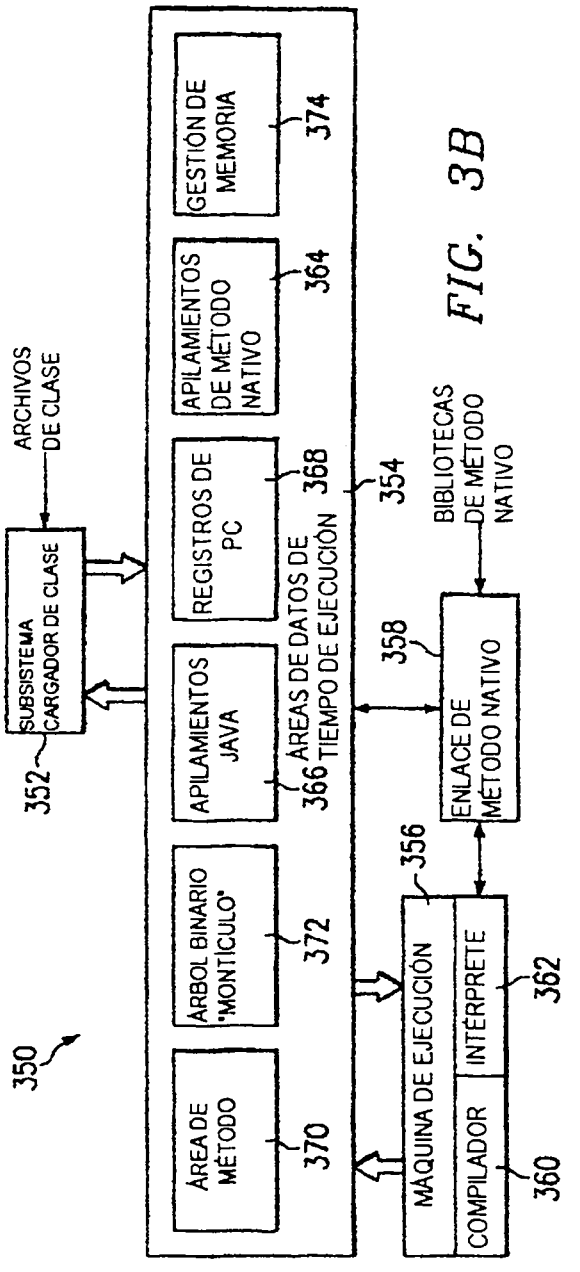
ARCHIVOS DE CLASE
SUBSISTEMA CARGADOR DE CLASE
352
350
ÁREA DE MÉTODO
370
ÁRBOL BINARIO "MONTÍCULO"
372
APILAMIENTOS JAVA
366
REGISTROS DE PC
368
ÁREAS DE DATOS DE TIEMPO DE EJECUCIÓN
354
APILAMIENTOS DE MÉTODO NATIVO
364
GESTIÓN DE MEMORIA
374
BIBLIOTECAS DE MÉTODO NATIVO
358
ENLACE DE MÉTODO NATIVO
356
MÁQUINA DE EJECUCIÓN
INTÉRPRETE
362
COMPILADOR
360

FIG. 3B

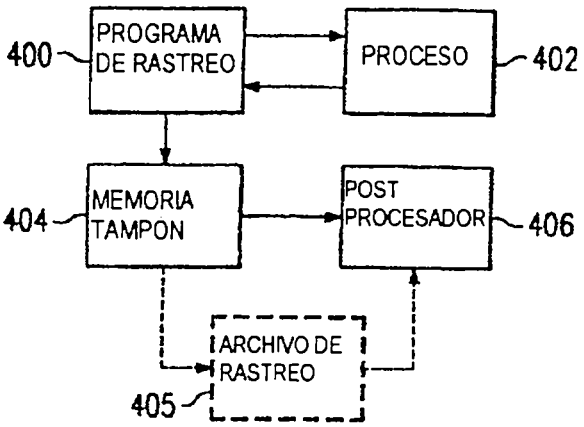

FIG. 4

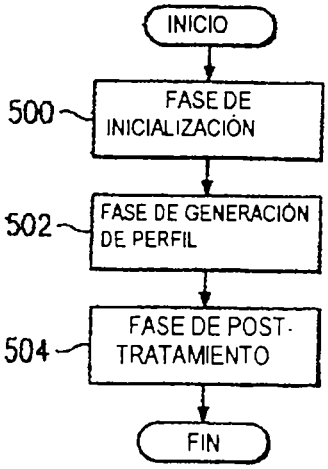

FIG. 5

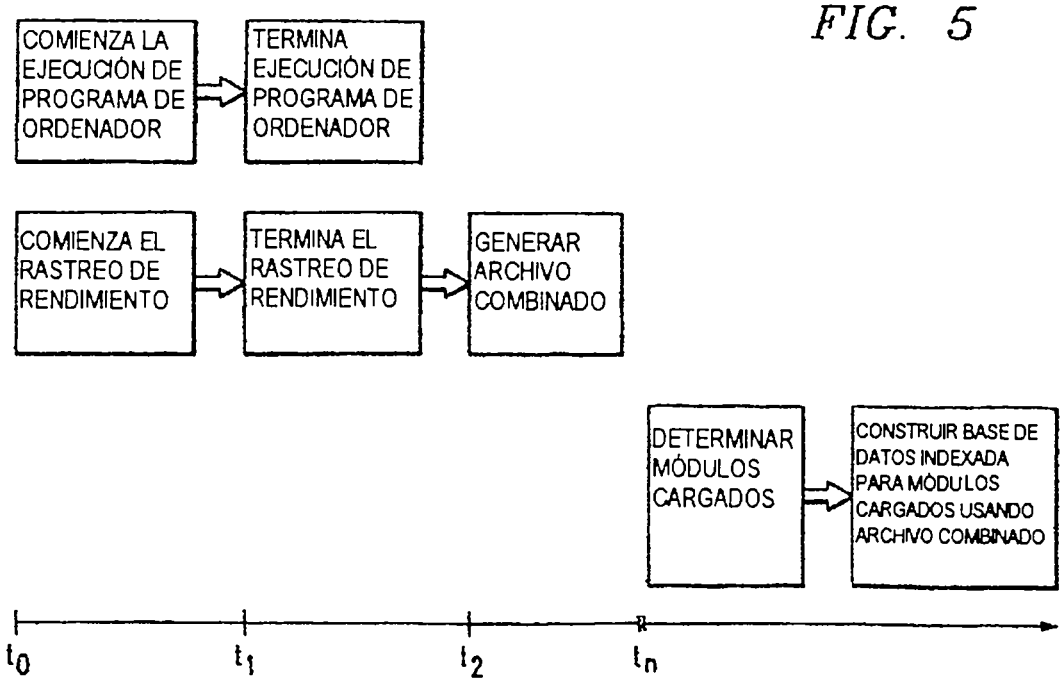

FIG. 6

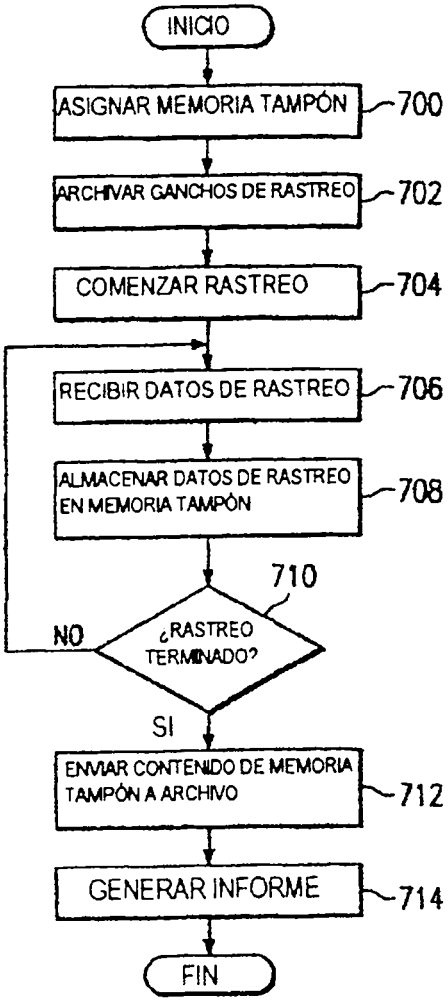
INICIO
ASIGNAR MEMORIA TAMPÓN
700
ARCHIVAR GANCHOS DE RASTREO
702
COMENZAR RASTREO
704
RECIBIR DATOS DE RASTREO
706
ALMACENAR DATOS DE RASTREO EN MEMORIA TAMPÓN
708
710
¿RASTREO TERMINADO?
NO
SI
ENVIAR CONTENIDO DE MEMORIA TAMPÓN A ARCHIVO
712
GENERAR INFORME
714
FIN

FIG. 7

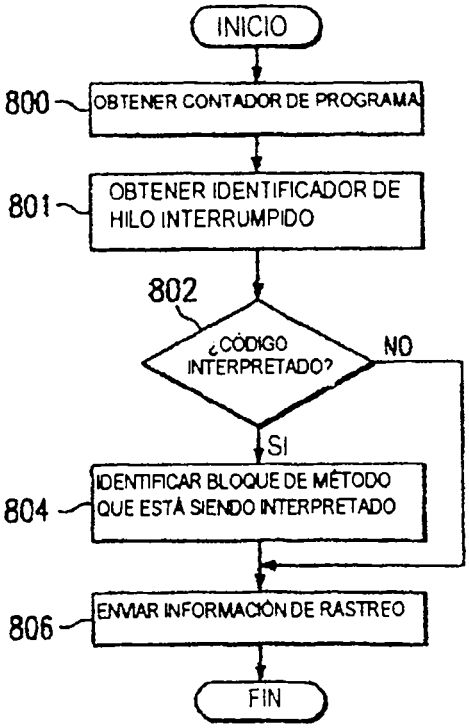
INICIO
800
OBTENER CONTADOR DE PROGRAMA
801
OBTENER IDENTIFICADOR DE HILO INTERRUMPIDO
802
¿CÓDIGO INTERPRETADO?
NO
SI
804
IDENTIFICAR BLOQUE DE MÉTODO QUE ESTÁ SIENDO INTERPRETADO
806
ENVIAR INFORMACIÓN DE RASTREO
FIN

FIG. 8

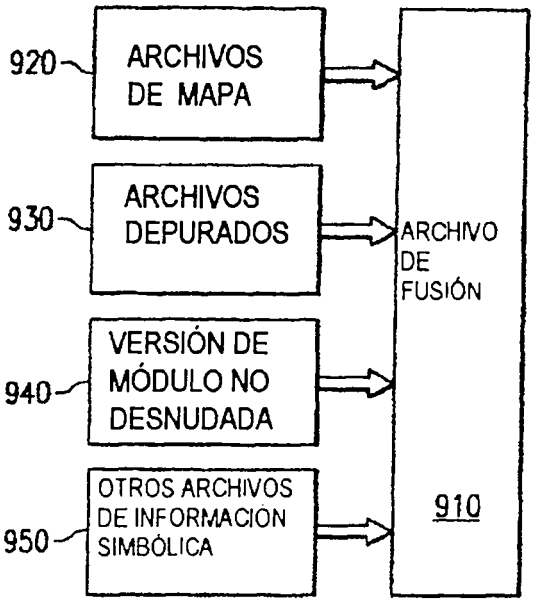
920
ARCHIVOS DE MAPA
930
ARCHIVOS DEPURADOS
940
VERSIÓN DE MÓDULO NO DESNUDADA
950
OTROS ARCHIVOS DE INFORMACIÓN SIMBÓLICA
ARCHIVO DE FUSIÓN
910

FIG. 9

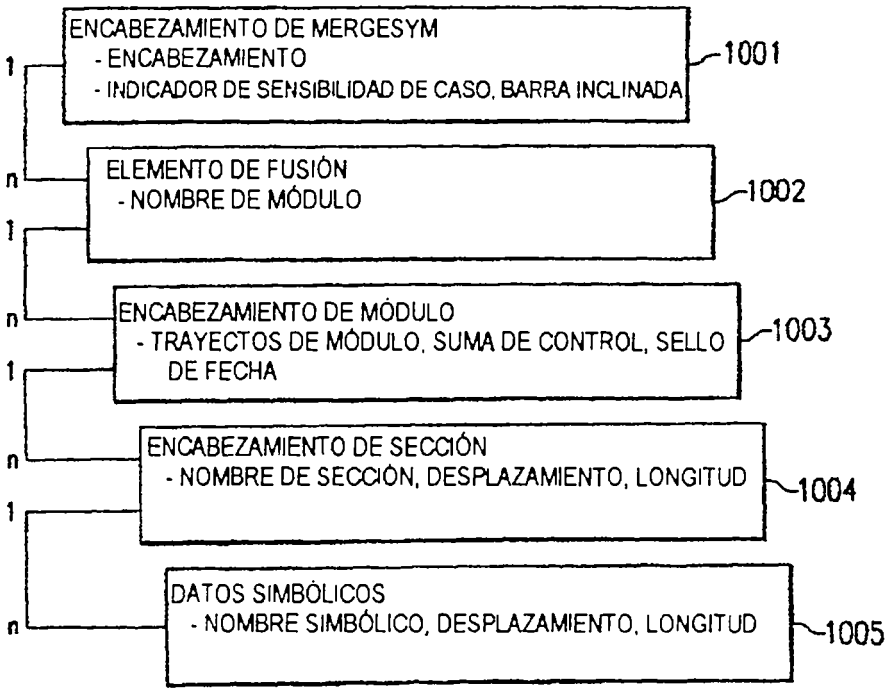
ENCABEZAMIENTO DE MERGESYM
- ENCABEZAMIENTO
- INDICADOR DE SENSIBILIDAD DE CASO, BARRA INCLINADA
1001
ELEMENTO DE FUSIÓN
- NOMBRE DE MÓDULO
1002
ENCABEZAMIENTO DE MÓDULO
- TRAYECTOS DE MÓDULO, SUMA DE CONTROL, SELLO DE FECHA
1003
ENCABEZAMIENTO DE SECCIÓN
- NOMBRE DE SECCIÓN, DESPLAZAMIENTO, LONGITUD
1004
DATOS SIMBÓLICOS
- NOMBRE SIMBÓLICO, DESPLAZAMIENTO, LONGITUD
1005
1
n
1
n
1
n
1
n

FIG. 10A

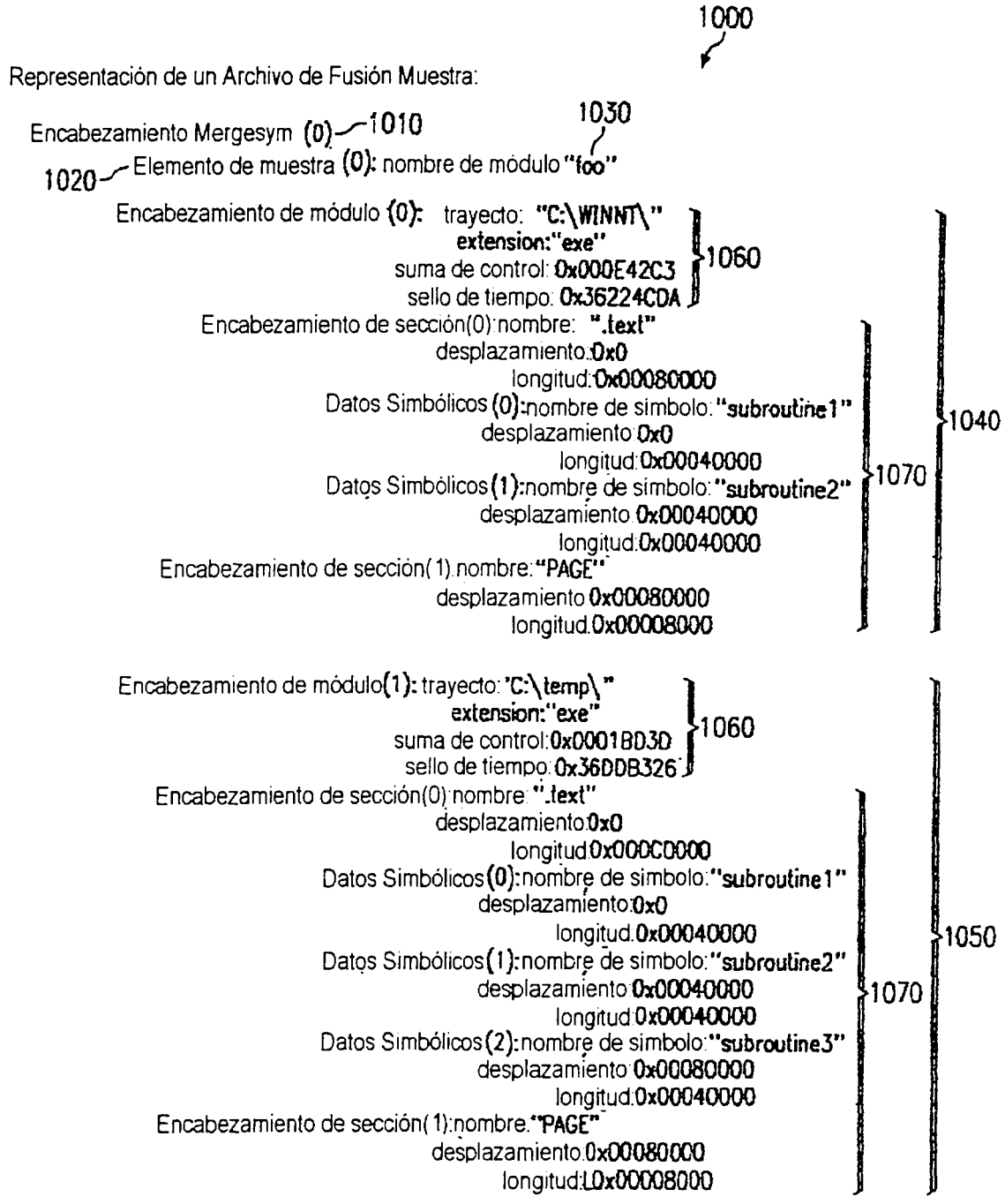
1000
Representación de un Archivo de Fusión Muestra:
Encabezamiento Mergesym (0) 1010
1030
1020 Elemento de muestra (0): nombre de módulo "foo"
Encabezamiento de módulo (0): trayecto: "C:\WINNT\"
extensión:"exe"
suma de control: 0x000E42C3
sello de tiempo: 0x36224CDA
1060
Encabezamiento de sección(0) nombre: ".text"
desplazamiento:0x0
longitud:0x00080000
Datos Simbólicos (0):nombre de símbolo:"subroutine1"
desplazamiento:0x0
longitud:0x00040000
Datos Simbólicos (1):nombre de símbolo:"subroutine2"
desplazamiento:0x00040000
longitud:0x00040000
1070
Encabezamiento de sección(1) nombre:"PAGE"
desplazamiento:0x00080000
longitud:0x00008000
1040
Encabezamiento de módulo(1): trayecto: "C:\temp\"
extensión:"exe"
suma de control:0x0001BD3D
sello de tiempo:0x36DDB326
1060
Encabezamiento de sección(0) nombre: ".text"
desplazamiento:0x0
longitud:0x000C0000
Datos Simbólicos (0):nombre de símbolo:"subroutine1"
desplazamiento:0x0
longitud:0x00040000
Datos Simbólicos (1):nombre de símbolo:"subroutine2"
desplazamiento:0x00040000
longitud:0x00040000
Datos Simbólicos (2):nombre de símbolo:"subroutine3"
desplazamiento:0x00080000
longitud:0x00040000
1070
Encabezamiento de sección(1):nombre:"PAGE"
desplazamiento:0x00080000
longitud:0x00008000
1050

FIG. 10B

1100

Archivo de rastreo de muestra:

```
A8  1  4:33900363  0  00000054
19  38 4:33904248  0  00000002 80140000 00080000 68000020 000E42C3 36224CDA .text C:\WINNT\foo.exe
19  38 4:339045d4  0  00000002 801C0000 00008000 60000020 000E42C3 36224CDA PAGE C:\WINNT\foo.exe
11  B4 4:339007D7  0     3E8
11  B3 4:3390191E  0  00000000 00000000
19  38 4:33b280E7  0  00000002 80240000 000C0000 68000020 0001BD3D 36DDB326 .text C:\temp\foo.exe
19  38 4:33904473  0  00000002 80300000 0000C000 60000020 0001BD3D 36DDB326 PAGE C:\temp\foo.exe
10  03 4:33C35EC5  0  80249000 00000002 00000005 00000000 00000000 00000000 00000000 00000000
19  39 4:33E04248  0  00000002 80140000
19  44 4:55A04368  0  00001F9A 445AF040 00000074 java/lang/String.<init>(([C)V)
19  44 4:55A04368  0  00001F9A 445AF0D8 0000007E java/lang/System.arraycopy ((Ljava/lang/Object;Ljava/lang/Object;II)V
```

FIG. 11

1200

Archivo de MTE de muestra:

```
19  38 4:33904248  0  00000002 80140000 00080000 68000020 000E42C3 36224CDA .text C:\WINNT\foo.exe
19  38 4:339045d4  0  00000002 801C0000 00008000 60000020 000E42C3 36224CDA PAGE C:\WINNT\foo.exe
19  38 4:33b280E7  0  00000002 80240000 000C0000 68000020 0001BD3D 36DDB326 .text C:\temp\foo.exe
19  38 4:33904473  0  00000002 80300000 0000C000 60000020 0001BD3D 36DDB326 PAGE C:\temp\foo.exe
19  39 4:33E04248  0  00000002 80140000
19  44 4:55A04368  0  00001F9A 445AF040 00000074 java/lang/String.<init>(([C)V)
19  44 4:55A04368  0  00001F9A 445AF0D8 0000007E java/lang/System.arraycopy ((Ljava/lang/Object;Ljava/lang/Object;II)V
```

FIG. 12

1300

BASE DE DATOS DE MUESTRA:

Búsqueda de dirección por (pid:dirección) da:

2:80140000 - foo.exe(subroutine1)	C:\WINNT\	
2:80180000 - foo.exe(subroutine2)	C:\WINNT\	1320
2:80240000 -	C:\temp\foo.exe(subroutine1)	
2:80280000 -	C:\temp\foo.exe(subroutine2)	
2:802C0000 -	C:\temp\foo.exe(subroutine3)	

1310

FIG. 13A

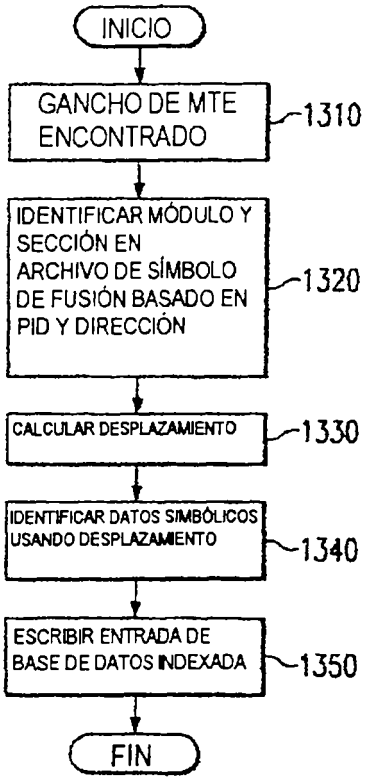

FIG. 13B

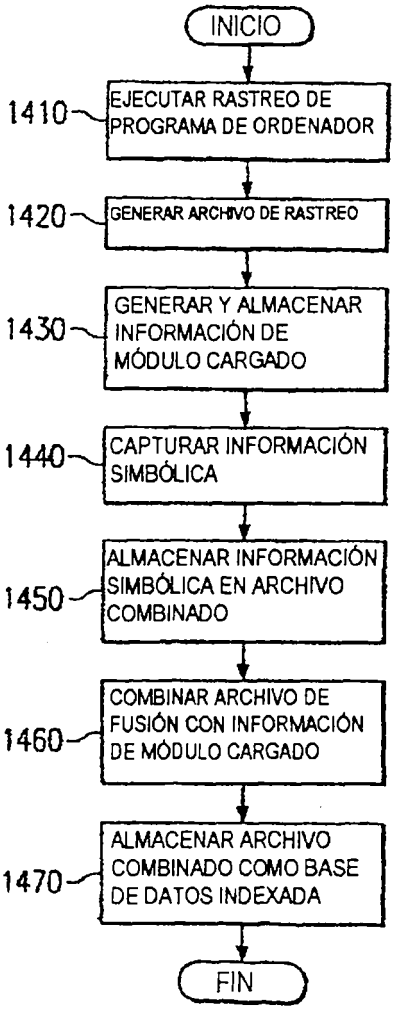

FIG. 14

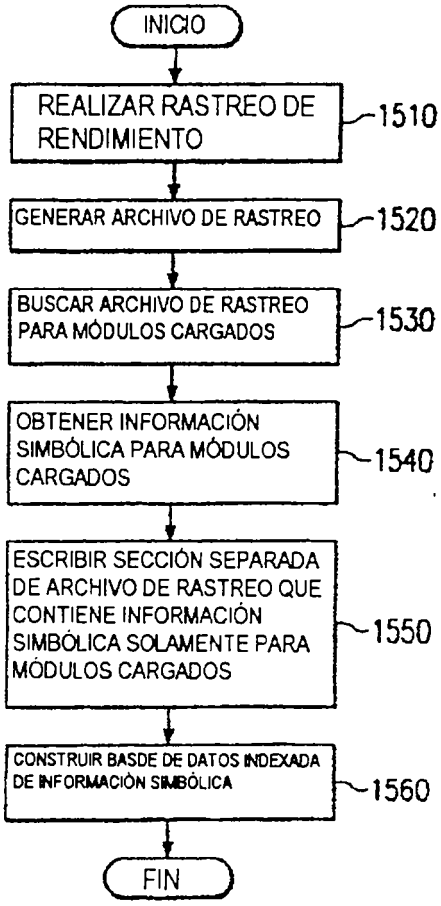
INICIO
REALIZAR RASTREO DE RENDIMIENTO
1510
GENERAR ARCHIVO DE RASTREO
1520
BUSCAR ARCHIVO DE RASTREO PARA MÓDULOS CARGADOS
1530
OBTENER INFORMACIÓN SIMBÓLICA PARA MÓDULOS CARGADOS
1540
ESCRIBIR SECCIÓN SEPARADA DE ARCHIVO DE RASTREO QUE CONTIENE INFORMACIÓN SIMBÓLICA SOLAMENTE PARA MÓDULOS CARGADOS
1550
CONSTRUIR BASDE DE DATOS INDEXADA DE INFORMACIÓN SIMBÓLICA
1560
FIN

FIG. 15

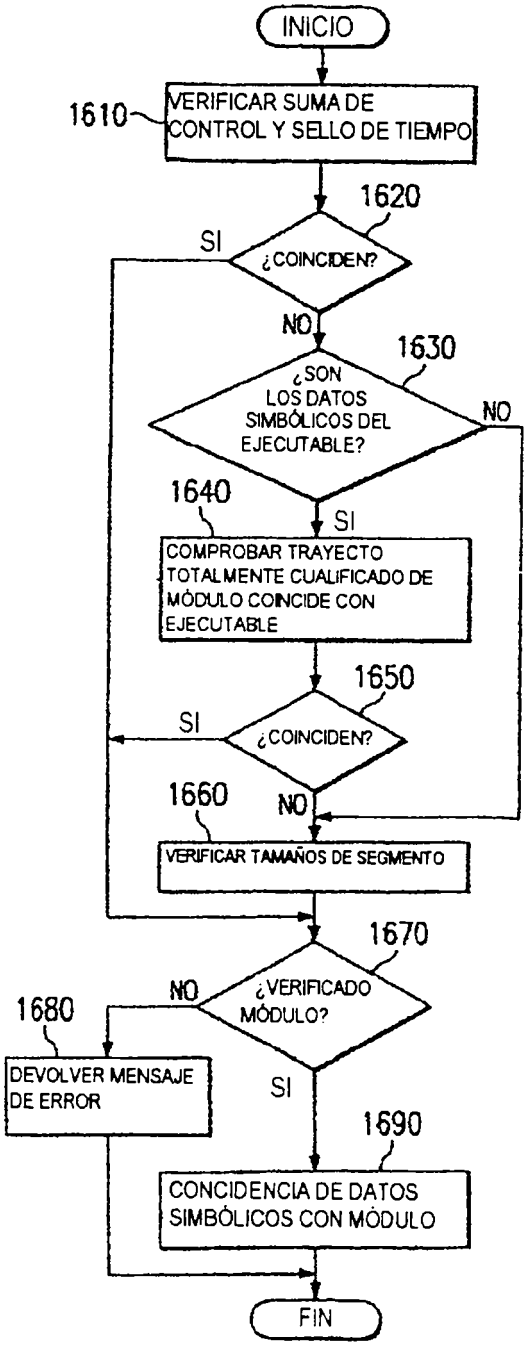
INICIO
1610
VERIFICAR SUMA DE CONTROL Y SELLO DE TIEMPO
1620
¿COINCIDEN?
SI
NO
1630
¿SON LOS DATOS SIMBÓLICOS DEL EJECUTABLE?
NO
1640
SI
COMPROBAR TRAYECTO TOTALMENTE CUALIFICADO DE MÓDULO COINCIDE CON EJECUTABLE
1650
¿COINCIDEN?
SI
1660
NO
VERIFICAR TAMAÑOS DE SEGMENTO
1670
¿VERIFICADO MÓDULO?
NO
1680
DEVOLVER MENSAJE DE ERROR
SI
1690
CONCIDENCIA DE DATOS SIMBÓLICOS CON MÓDULO
FIN

FIG. 16

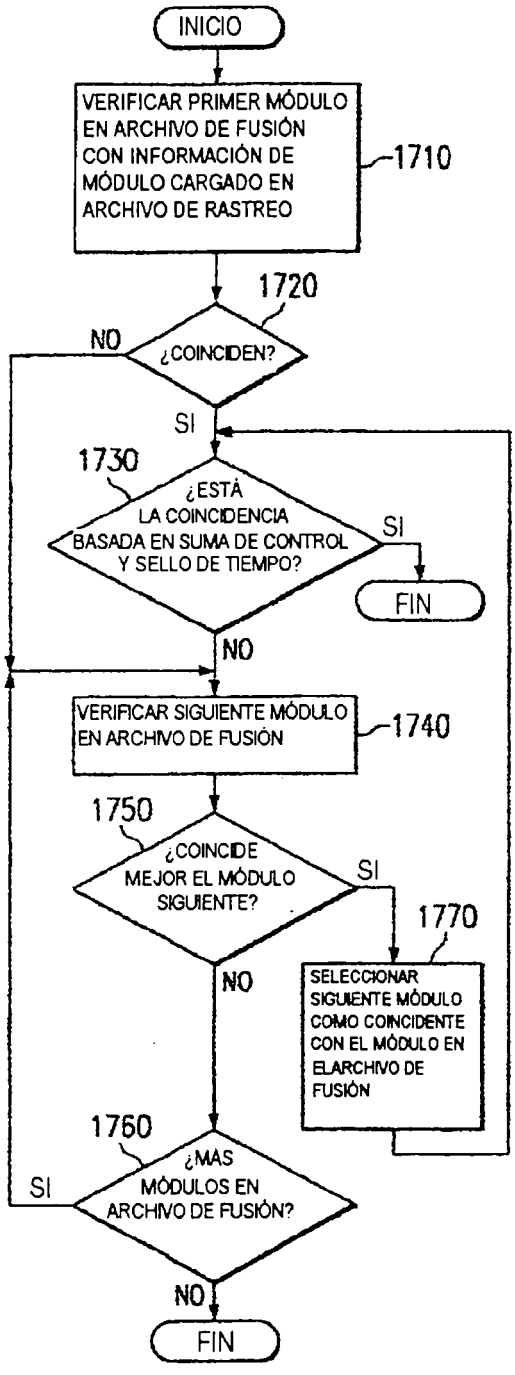
INICIO
VERIFICAR PRIMER MÓDULO EN ARCHIVO DE FUSIÓN CON INFORMACIÓN DE MÓDULO CARGADO EN ARCHIVO DE RASTREO
1710
1720
¿COINCIDEN?
NO
SI
1730
¿ESTÁ LA COINCIDENCIA BASADA EN SUMA DE CONTROL Y SELLO DE TIEMPO?
SI
FIN
NO
VERIFICAR SIGUIENTE MÓDULO EN ARCHIVO DE FUSIÓN
1740
1750
¿COINCIDE MEJOR EL MÓDULO SIGUIENTE?
SI
1770
SELECCIONAR SIGUIENTE MÓDULO COMO COINCIDENTE CON EL MÓDULO EN EL ARCHIVO DE FUSIÓN
NO
1760
¿MAS MÓDULOS EN ARCHIVO DE FUSIÓN?
SI
NO
FIN

FIG. 17

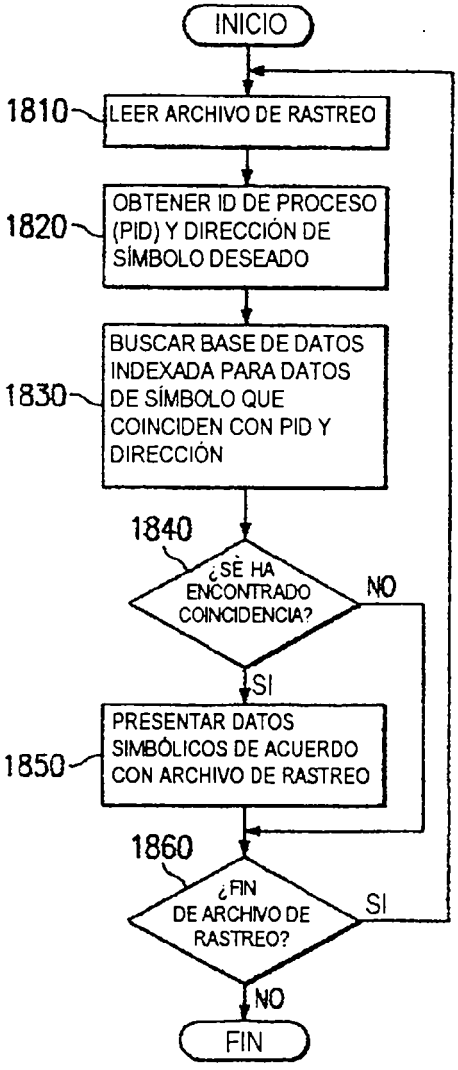
INICIO
1810
LEER ARCHIVO DE RASTREO
1820
OBTENER ID DE PROCESO (PID) Y DIRECCIÓN DE SÍMBOLO DESEADO
1830
BUSCAR BASE DE DATOS INDEXADA PARA DATOS DE SÍMBOLO QUE COINCIDEN CON PID Y DIRECCIÓN
1840
¿SE HA ENCONTRADO COINCIDENCIA?
NO
SI
1850
PRESENTAR DATOS SIMBÓLICOS DE ACUERDO CON ARCHIVO DE RASTREO
1860
¿FIN DE ARCHIVO DE RASTREO?
SI
NO
FIN

FIG. 18

```
FileName \winnt\system32\hal.dll
TaskName system
pid 0
start 80016180
size 00001e00
SegName PAGE SegNo 0x3
Symbols 0
80016180  6604  Start_Segment_#3
80016604  113e  HalGetAdapter
80017742    1c  IoAssignDriveLetters
8001775e    1c  IoReadPartitionTable
8001777a    1c  IoSetPartitionInformation
80017796    f0  IoWritePartitionTable
80017886    34  HalAdjustResourceList
800178ba    44  HalAssignSlotResources
800178fe   682  HalGetInterruptVector
80017f80     0  End_Segment_#3
```

FIG. 19

```
FileName d:\winnt\system32\hal.dll
TaskName system
pid ????
start 0
size 00001e00
SegName PAGE SegNo 0x3
Symbols 0
0          6604  Start_Segment_#3
474        113e  HalGetAdapter
15b2         1c  IoAssignDriveLetters
15ce         1c  IoReadPartitionTable
15ea         1c  IoSetPartitionInformation
1605         f0  IoWritePartitionTable
16f5         34  HalAdjustResourceList
1739         44  HalAssignSlotResources
1dbb        682  HalGetInterruptVector
253d          0  End_Segment_#3
```

FIG. 20

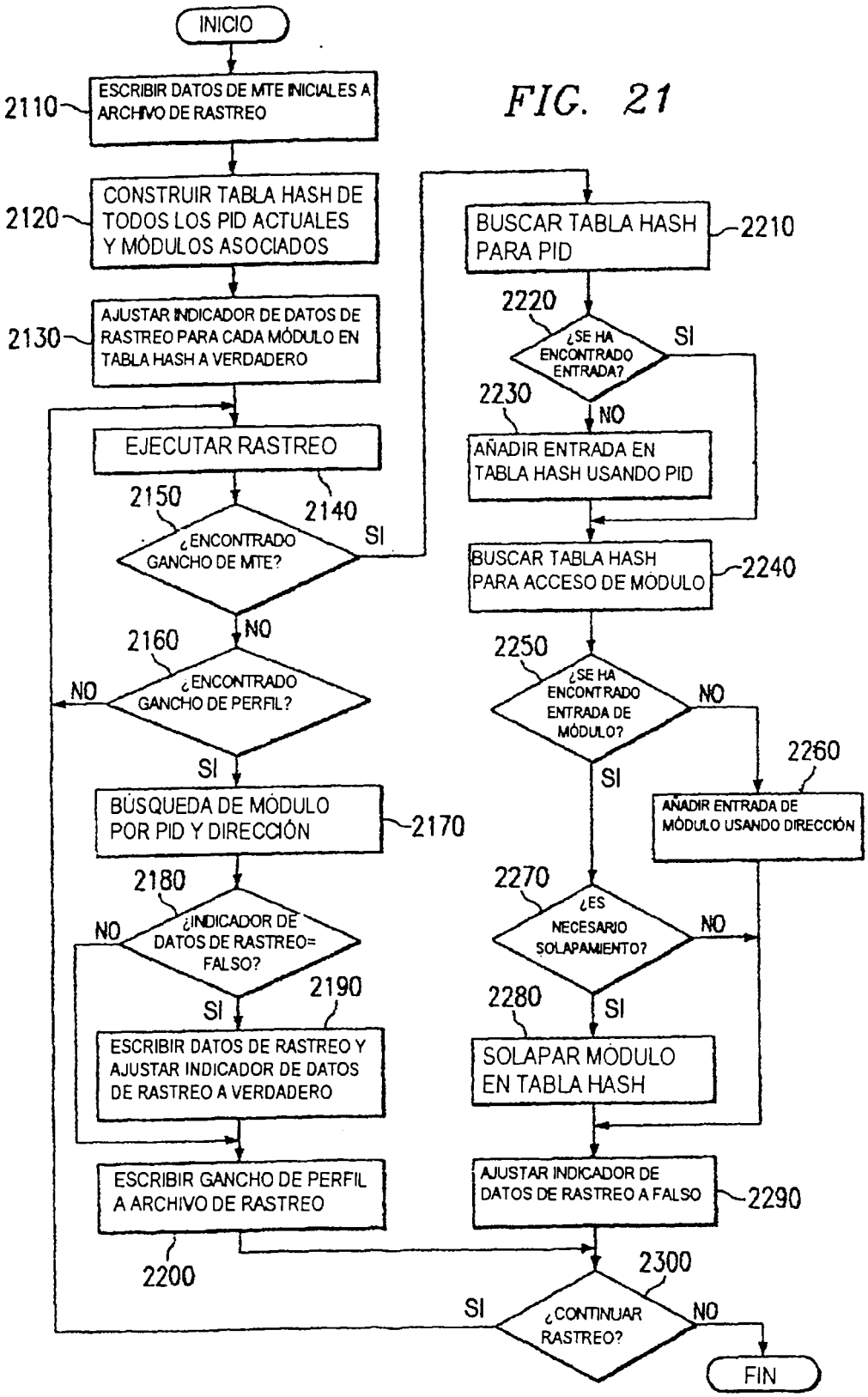
INICIO
ESCRIBIR DATOS DE MTE INICIALES A ARCHIVO DE RASTREO
2110
CONSTRUIR TABLA HASH DE TODOS LOS PID ACTUALES Y MÓDULOS ASOCIADOS
2120
AJUSTAR INDICADOR DE DATOS DE RASTREO PARA CADA MÓDULO EN TABLA HASH A VERDADERO
2130
EJECUTAR RASTREO
2140
2150
¿ENCONTRADO GANCHO DE MTE?
SI
NO
2160
¿ENCONTRADO GANCHO DE PERFIL?
NO
SI
BÚSQUEDA DE MÓDULO POR PID Y DIRECCIÓN
2170
2180
¿INDICADOR DE DATOS DE RASTREO= FALSO?
NO
SI
2190
ESCRIBIR DATOS DE RASTREO Y AJUSTAR INDICADOR DE DATOS DE RASTREO A VERDADERO
ESCRIBIR GANCHO DE PERFIL A ARCHIVO DE RASTREO
2200
BUSCAR TABLA HASH PARA PID
2210
2220
¿SE HA ENCONTRADO ENTRADA?
SI
NO
2230
AÑADIR ENTRADA EN TABLA HASH USANDO PID
BUSCAR TABLA HASH PARA ACCESO DE MÓDULO
2240
2250
¿SE HA ENCONTRADO ENTRADA DE MÓDULO?
NO
SI
2260
AÑADIR ENTRADA DE MÓDULO USANDO DIRECCIÓN
2270
¿ES NECESARIO SOLAPAMIENTO?
NO
SI
2280
SOLAPAR MÓDULO EN TABLA HASH
AJUSTAR INDICADOR DE DATOS DE RASTREO A FALSO
2290
2300
¿CONTINUAR RASTREO?
SI
NO
FIN

FIG. 21