(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 738 282**

(51) Int. CI.:

G06F 9/54 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **03.01.2014 PCT/US2014/010134**

(87) Fecha y número de publicación internacional: **10.07.2014 WO14107553**

(96) Fecha de presentación y número de la solicitud europea: **03.01.2014 E 14701842 (8)**

(97) Fecha y número de publicación de la concesión europea: **15.05.2019 EP 2941704**

(54) Título: **Almacenamiento en caché de copia cero**

(30) Prioridad:

04.01.2013 US 201313734785

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
21.01.2020

(73) Titular/es:

MICROSOFT TECHNOLOGY LICENSING, LLC (100.0%)
One Microsoft Way
Redmond, WA 98052, US

(72) Inventor/es:

YU, JINSONG;
GOODSELL, ANDREW, E.;
TEREK, F., SONER;
BRUMME, CHRISTOPHER, WELLINGTON y
MOHAMED, AHMED, HASSAN

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

DESCRIPCIÓN

Almacenamiento en caché de copia cero

Antecedentes

El rendimiento del sistema operativo de un ordenador a menudo se caracteriza por la tasa máxima de operaciones de entrada/salida (E/S) (también denominada "rendimiento de E/S") que el sistema operativo puede mantener durante un intervalo de tiempo determinado. Como resultado, los sistemas operativos emplean una variedad de mecanismos bien conocidos para aumentar el rendimiento de E/S.

Los sistemas operativos se escriben tradicionalmente utilizando lenguajes no administrados (como el lenguaje ensamblador, C o C++), lo que proporciona al programador del sistema un control muy preciso de cómo se manipula la memoria. El uso de punteros no controlados se puede usar para minimizar la sobrecarga del sistema operativo y permitir un mayor rendimiento o una menor latencia. La desventaja del uso de estos punteros no controlados es que son difíciles de crear y razonar, lo que provoca que el software no sea confiable y existan vulnerabilidades de seguridad.

Escribir software en un lenguaje de programación administrado proporciona beneficios de corrección sustanciales y eficiencias en el tiempo de desarrollo. Estos lenguajes administrados evitan que los programadores creen muchos tipos de defectos de software, lo que conduce a una mejor calidad del software y un menor tiempo de desarrollo. La corrección del sistema operativo es un ingrediente crítico para brindar una experiencia de cómputo confiable y segura. Por lo tanto, el uso de lenguajes administrados para crear sistemas operativos es una propuesta convincente, ya que la confiabilidad del sistema operativo puede mejorar y los costos de desarrollo pueden reducirse.

Para lograr estos beneficios, los lenguajes de programación administrados insertan una capa de abstracción entre el código fuente redactado por el programador y los recursos de la máquina en bruto de un sistema informático físico. Esta capa de abstracción generalmente sirve para restringir lo que los programadores pueden escribir, y al hacerlo eliminan categorías enteras de defectos potenciales. Desafortunadamente, esta capa de abstracción introduce una sobrecarga que puede afectar el rendimiento del software que se está creando. Como resultado, un supuesto común es que los lenguajes administrados intercambian defectos de corrección por defectos de rendimiento. Por lo tanto, el software escrito en lenguajes administrados a menudo se considera intrínsecamente más lento que el software escrito en lenguajes no administrados.

El problema particular que afecta a los sistemas operativos de código administrado es la necesidad inherente de copiar datos entre capas a medida que los datos viajan a través del sistema. Esto se debe al hecho de que existen distintos componentes del sistema en diferentes contextos de aislamiento y no existe un mecanismo claro para salir de estos contextos de aislamiento.

El artículo "IO-Lite: A Unified I/O Buffering and Caching System" (IO-Lite: Un sistema unificado de almacenamiento en búfer y en caché de E/S" por VS Pai *et al.* (ACM Transactions on Computer Systems, Vol. 18, No. 1, febrero de 2000, pp. 37 - 66) divulga IO-Lite, un sistema unificado de almacenamiento en búfer y en caché de E/S para sistemas operativos de uso general. I/O-Lite unifica todo el almacenamiento en búfer y en caché en el sistema, en la medida en que lo permita el hardware. IO-Lite almacena datos de E/S en búfer en búferes inmutables, cuyas ubicaciones en la memoria física nunca cambian. Los diversos subsistemas utilizan agregados de búfer modificables para acceder a los datos de acuerdo con sus necesidades.

Sumario

La invención se refiere a un sistema de conformidad con la reivindicación 1 y a un procedimiento de conformidad con la reivindicación 8. Realizaciones preferentes se describen en las reivindicaciones dependientes.

De acuerdo con al menos una realización descrita en la presente memoria, se describe el almacenamiento en caché de un búfer inmutable. El búfer inmutable protege los datos que se rellenan en el mismo para que no cambien durante la vida útil del búfer inmutable. La dirección física del búfer inmutable también está protegida contra cambios durante la vida útil del búfer inmutable. Una primera entidad informática que mantiene una caché del búfer inmutable y tiene una referencia fuerte al búfer inmutable. Siempre que cualquier entidad tenga una referencia fuerte al búfer inmutable, se garantiza que el búfer inmutable continuará existiendo durante al menos la duración de la referencia fuerte para cada entidad que tenga la referencia fuerte. Una segunda entidad informática se comunica con la primera entidad informática para obtener una referencia fuerte al búfer inmutable y luego leer los datos del búfer inmutable. Después de leer los datos de la caché, la segunda entidad informática reduce la referencia fuerte al búfer inmutable a una referencia débil al búfer inmutable. Una referencia débil al búfer inmutable no garantiza que el búfer inmutable continuará existiendo mientras dure la referencia débil. Al recibir una solicitud para leer desde el búfer inmutable mientras aún existe la referencia débil a la caché, la segunda entidad informática

determina si el búfer inmutable aún existe: si el búfer inmutable aún existe, la segunda entidad informática convierte la referencia débil al búfer inmutable a una referencia fuerte al búfer inmutable y lee los datos del búfer inmutable sin realizar una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática; y si el búfer inmutable aún no existe, la segunda entidad informática realiza una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática para que la primera entidad informática vuelva a adquirir los datos y vuelva a crear un nuevo búfer inmutable y permita que la segunda entidad informática obtenga una referencia fuerte al nuevo búfer inmutable y lea del nuevo búfer inmutable.

Esto permite que la primera y la segunda entidades informáticas tengan una caché para el búfer inmutable sin requerir una comunicación entre las dos entidades, a excepción de la primera comunicación para permitir que la segunda entidad informática obtenga acceso inicial a la referencia fuerte.

Este sumario no tiene por objeto identificar las características clave o las características esenciales de la materia u objeto que se reivindica, ni está destinada a ser utilizada como ayuda para determinar el ámbito de la materia reivindicada.

Breve descripción de los dibujos

Con el fin de describir la manera en que se pueden obtener las ventajas y características antes mencionadas, además de otras posibles, se proporcionará una descripción más particular de varias realizaciones por referencia a las figuras adjuntas. Al comprender que estas figuras representan solamente realizaciones de muestra y, por lo tanto, no deben considerarse como que limitan el ámbito de la invención, las realizaciones se describirán y explicarán con mayor especificidad y detalle mediante el uso de las figuras adjuntas, en las que:

La Figura 1 ilustra de manera abstracta un sistema informático en el que se pueden emplear algunas realizaciones descritas en la presente memoria;

La Figura 2 ilustra un diagrama de flujo de un procedimiento para proporcionar un búfer inmutable;

La Figura 3A ilustra un entorno en el que se produce un procedimiento de llenado de un búfer;

La Figura 3B ilustra el entorno en el que el búfer poblado se hace inmutable;

La Figura 4 ilustra un diagrama de flujo de un procedimiento para usar el búfer inmutable;

La Figura 5 ilustra un entorno en el que diferentes entidades informáticas tienen diferentes vistas en un búfer inmutable;

La Figura 6 ilustra un diagrama de flujo de un procedimiento para comunicar los datos inmutables de una entidad informática a la siguiente;

La Figura 7 ilustra un entorno de transmisión en el que se proporciona un flujo de datos desde una fuente de flujo a un búfer de flujo, y luego se proporciona desde el búfer a un consumidor de flujo;

La Figura 8 ilustra un entorno en el que una segunda entidad informática adquiere una caché a través de la caché de una primera entidad informática;

La Figura 9 ilustra un diagrama de flujo de un procedimiento para que una segunda entidad informática lea primero desde una caché soportada por una primera entidad informática;

La Figura 10 ilustra un diagrama de flujo de un procedimiento para que la segunda entidad informática lea posteriormente desde la caché soportada por la primera entidad informática;

La Figura 11 ilustra un diagrama de flujo de un procedimiento para que la primera entidad informática (o la caché de respaldo) realice el desalojo;

La Figura 12 ilustra un ejemplo de sistema de código administrado; y

La Figura 13 muestra una matriz administrada normal de bytes que tiene dos tramos distintos apuntando hacia ella y que permiten que la aplicación vea porciones de la matriz como tipos diferentes.

Descripción detallada

De acuerdo con las realizaciones descritas en la presente memoria, se describen mecanismos que promueven la semántica de entrada/salida (E/S) de copia cero en sistemas operativos administrados. Algunos de estos

mecanismos se pueden utilizar en sistemas operativos de código no administrado, así como en sistemas operativos de código administrado. Los mecanismos no se excluyen mutuamente ya que uno, algunos o incluso todos los mecanismos se pueden combinar para promover aún más la semántica de E/S de copia cero.

"Copia cero" se refiere a una arquitectura diseñada para permitir que los datos ingresen al sistema al escribirse en la memoria y propagarse a través de muchas capas de abstracciones sin la necesidad de copiar los datos. Una arquitectura de copia cero no garantiza que no se realice ninguna copia de datos. Más bien, simplemente pone en marcha un mecanismo para garantizar que la mayoría de las operaciones de E/S se puedan realizar sin copiar. En esta descripción y en las reivindicaciones, "memoria" se define como cualquier memoria de acceso aleatorio, que generalmente es una memoria volátil, pero también puede incluir porciones no volátiles, o tal vez puede ser completamente no volátil. En esta descripción y en las reivindicaciones, "memoria" se define como el medio de almacenamiento principal de un sistema informático, comprometido con ubicaciones individuales accesibles a los microprocesadores del sistema informático y accesibles a dispositivos de hardware como controladores de gráficos o controladores de interfaz de red a través de mecanismos de acceso directo a memoria (DMA, Direct Memory Access).

En primer lugar, se describirán los mecanismos de datos masivos inmutables de copia cero que se pueden compartir y que utilizan un búfer inmutable de datos compartidos. Dichos mecanismos permiten la transferencia de grandes búferes de datos a través del sistema informático sin necesidad de ser copiados. Los mecanismos se extenderán aún más al uso compartido de flujos de datos dentro del sistema informático con control de flujo completo para permitir la utilización eficiente de los recursos, mientras se mantiene la semántica completa de copia cero. Si bien la seguridad de tipos actual de los sistemas de código administrado permite una implementación más inmediata de estos mecanismos, también se puede emplear el uso de estos mecanismos dentro de los sistemas de código no administrado.

En segundo lugar, se describirá un mecanismo para el almacenamiento en caché de copia cero. Dicho almacenamiento en caché de copia cero puede emplearse tanto en sistemas de código no administrado como en sistemas de código administrado. El almacenamiento en caché de copia cero hace posible crear una arquitectura de almacenamiento en caché de propósito general que presenta una semántica de copia cero para los datos que ingresan en la caché, así como los datos que se devuelven desde la caché.

En tercer lugar, se describirán varios mecanismos que mejoran aún más el rendimiento de los sistemas de código administrado, ya sea que esos sistemas empleen el búfer inmutable o los datos compartidos. Dichos mecanismos de código administrado incluyen acceso uniforme a memoria y conversión de tipos en tipo seguro. El acceso uniforme a memoria permite que el código administrado acceda de manera uniforme tanto a la memoria administrada como a la memoria no administrada (utilizada para búferes de E/S) utilizando un enfoque coherente y ajustable. La conversión de tipos en tipo seguro permite que el código administrado realice la conversión de puntero para permitir que una región determinada de la memoria se vea como de tipos distintos mientras se mantiene la seguridad de todo tipo.

Se describirá una discusión introductoria de un sistema informático con respecto a la Figura 1. Luego, los mecanismos enumerados anteriormente se describirán en el orden proporcionado anteriormente con respecto a las Figuras 2 a 13.

Los sistemas informáticos actualmente están tomando una amplia variedad de formas cada vez mayor. Los sistemas informáticos pueden ser, por ejemplo, dispositivos de mano, electrodomésticos, ordenadores portátiles, ordenadores de escritorio, ordenadores centrales, sistemas informáticos distribuidos o incluso dispositivos que no han sido considerados convencionalmente como un sistema informático. En esta descripción y en las reivindicaciones, el término "sistema informático" se define ampliamente como que incluye cualquier dispositivo o sistema (o una combinación de estos) que incluya al menos un procesador físico y tangible, y una memoria física y tangible capaz de tener en ella Instrucciones ejecutables por ordenador que pueden ser ejecutadas por el procesador. La memoria puede tomar cualquier forma y puede depender de la naturaleza y la forma del sistema informático. Un sistema informático puede estar distribuido en un entorno de red y puede incluir sistemas informáticos de múltiples componentes.

Como se ilustra en la Figura 1, en su configuración más básica, un sistema informático 100 incluye al menos una unidad de procesamiento 102 y medios legibles por ordenador 104. El medio legible por ordenador 104 puede pensarse conceptualmente como que incluye una memoria física del sistema, que puede ser volátil, no volátil, o alguna combinación de estas. El medio legible por ordenador 104 también incluye conceptualmente el almacenamiento masivo no volátil. Si se distribuye el sistema informático, también se pueden distribuir las capacidades de procesamiento, memoria y/o almacenamiento.

Como se usa en la presente memoria, el término "módulo ejecutable" o "componente ejecutable" puede referirse a objetos de software, enrutamientos o procedimientos que pueden ejecutarse en el sistema informático. Los diferentes componentes, módulos, motores y servicios descritos en la presente memoria pueden implementarse

como objetos o procesos que se ejecutan en el sistema informático (por ejemplo, como subhilos separados). Dichos módulos ejecutables pueden ser código administrado en el caso de que se ejecuten en un entorno administrado en el que se aplica la seguridad de tipos y en el que los procesos se asignan sus propios y distintos objetos de memoria. Dichos módulos ejecutables también pueden ser código no administrado en el caso de que los módulos ejecutables se realicen en código nativo como C o C++.

En la descripción que sigue, las realizaciones se describen con referencia a actos que son realizados por uno o más sistemas informáticos. Si dichos actos se implementan en software, uno o más procesadores del sistema informático asociado que realiza el acto dirigen el funcionamiento del sistema informático en respuesta a la ejecución de instrucciones ejecutables por ordenador. Por ejemplo, tales instrucciones ejecutables por ordenador pueden incorporarse en uno o más medios legibles por ordenador que forman un producto de programa de ordenador. Un ejemplo de tal operación involucra la manipulación de datos. Las instrucciones ejecutables por ordenador (y los datos manipulados) pueden almacenarse en la memoria 104 del sistema informático 100. El sistema informático 100 también puede contener canales de comunicación 108 que permiten que el sistema informático 100 se comunique con otros procesadores a través de, por ejemplo, la red 110.

Las realizaciones descritas en la presente memoria pueden comprender o utilizar un ordenador de propósito especial o de propósito general que incluye hardware de ordenador, tal como, por ejemplo, uno o más procesadores y memoria del sistema, como se describe con mayor detalle a continuación. Las realizaciones descritas en la presente memoria también incluyen medios físicos y otros medios legibles por ordenador para transportar o almacenar instrucciones ejecutables por ordenador y/o estructuras de datos. Dichos medios legibles por ordenador pueden ser cualquier medio disponible al que se pueda acceder mediante un sistema informático de propósito general o de propósito especial. Los medios legibles por ordenador que almacenan instrucciones ejecutables por ordenador son medios de almacenamiento físico. Los medios legibles por ordenador que llevan instrucciones ejecutables por ordenador son medios de transmisión. Por lo tanto, a modo de ejemplo, y no de forma limitativa, las realizaciones de la invención pueden comprender al menos dos tipos diferentes de medios legibles por ordenador: medios de almacenamiento informáticos y medios de transmisión.

El medio de almacenamiento informático incluye RAM, ROM, EEPROM, CD-ROM u otro almacenamiento de disco óptico, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnético, o cualquier otro medio de almacenamiento tangible que pueda usarse para almacenar los medios de código de programa deseados en la forma de instrucciones o estructuras de datos ejecutables por ordenador, a las que se puede acceder mediante un ordenador de propósito general o de propósito especial.

Una "red" se define como uno o más enlaces de datos que permiten el transporte de datos electrónicos entre sistemas informáticos y/o módulos y/u otros dispositivos electrónicos. Cuando la información se transfiere o proporciona a través de una red u otra conexión de comunicaciones (ya sea cableada, inalámbrica o una combinación de cableada o inalámbrica) a un ordenador, el ordenador ve la conexión como un medio de transmisión. Los medios de transmisión pueden incluir una red y/o enlaces de datos que pueden usarse para transportar o los medios de código de programa deseados en forma de instrucciones ejecutables por ordenador o estructuras de datos y a los que se puede acceder mediante un ordenador de propósito general o de propósito especial. Las combinaciones de los anteriores también deben incluirse dentro del alcance de los medios legibles por ordenador.

Además, al alcanzar diversos componentes del sistema informático, los medios de código de programa en forma de instrucciones ejecutables por ordenador o estructuras de datos pueden transferirse automáticamente desde los medios de transmisión a los medios de almacenamiento informáticos (o viceversa). Por ejemplo, las instrucciones ejecutables por ordenador o las estructuras de datos recibidas a través de una red o enlace de datos pueden almacenarse en la memoria RAM dentro de un controlador de interfaz de red (por ejemplo, una "NIC"), y luego transferirse a la RAM del sistema informático y/o a medios menos volátiles de almacenamiento informático en un sistema informático. Por lo tanto, debe entenderse que los medios de almacenamiento informático pueden incluirse en los componentes del sistema informático que también (o incluso principalmente) utilizan medios de transmisión.

Las instrucciones ejecutables por ordenador comprenden, por ejemplo, instrucciones y datos que, cuando se ejecutan en un procesador, hacen que un ordenador de propósito general, ordenador de propósito especial o dispositivo de procesamiento de propósito especial realice una determinada función o grupo de funciones. Las instrucciones ejecutables por ordenador pueden ser, por ejemplo, archivos binarios, instrucciones de formato intermedio, como lenguaje de ensamblador, o incluso código fuente. Aunque el tema se ha descrito en un lenguaje específico para características estructurales y/o actos metodológicos, debe entenderse que la materia de las reivindicaciones adjuntas no está necesariamente limitada a las características o actos específicos descritos anteriormente. Más bien, las características y los actos descritos se divulgan como formas de ejemplo de implementación de las reivindicaciones.

Los expertos en la técnica apreciarán que la invención puede ponerse en práctica en entornos informáticos de red con muchos tipos de configuraciones de sistemas informáticos, que incluyen ordenadores personales, ordenadores

de escritorio, ordenadores portátiles, procesadores de mensajes, dispositivos de mano, sistemas multiprocesador, dispositivos electrónicos de consumo basados en microprocesador o programables, ordenadores personales de red, miniordenadores, ordenadores centrales, teléfonos móviles, asistentes digitales personales, buscapersonas, enrutadores, conmutadores y similares. La invención también puede ponerse en práctica en entornos de sistemas distribuidos donde los sistemas informáticos locales y remotos, que están vinculados (ya sea mediante enlaces de datos cableados, enlaces de datos inalámbricos o mediante una combinación de enlaces de datos cableados e inalámbricos) a través de una red, ambos realizan tareas. En un entorno de sistema distribuido, los módulos de programa pueden ubicarse en dispositivos de almacenamiento de memoria local y remota.

Datos masivos inmutables y compartibles de copia cero

Un reto importante para soportar la copia cero ha sido las interfaces de E/S en los sistemas tradicionales que se definen como operaciones de copia entre diferentes capas en un sistema. Una interfaz de programa de aplicación (API) de lectura acepta un búfer de aplicación como entrada y lo llena con datos de alguna fuente de datos. De manera similar, una API de escritura toma un búfer de aplicación y escribe su contenido en algún destino de datos. La semántica de las API de lectura/escritura otorga a la aplicación total libertad para la alineación del búfer, el espacio de asignación y la retención. Este modelo simple tiene varias limitaciones inherentes, ya que el modelo no puede expresar búferes no contiguos o reducir el número de operaciones de copia de datos.

Muchos sistemas operativos admiten archivos asignados en memoria como un mecanismo para compartir páginas en la caché del búfer del sistema de archivos con aplicaciones y evitar las operaciones de copia asociadas con la interfaz de lectura/escritura. Se han agregado API especiales a la interfaz de red para enviar directamente datos desde la caché del búfer del sistema de archivos utilizando asignaciones de memoria de archivos para acelerar el tráfico de red.

La abstracción del archivo asignado en memoria carece de soporte para ocultar la alineación subyacente y el diseño disperso de los búferes y requiere que las aplicaciones manejen y administren las asignaciones virtuales y las vistas de datos lógicos directamente. Por ejemplo, una aplicación que accede a un archivo en el desplazamiento 10 necesita aplicar aritmética de puntero en la dirección virtual de la base de mapeo para obtener la dirección correcta. La extensión del archivo mientras se asigna requiere que la aplicación administre vistas adicionales que pueden no ser necesariamente contiguas en su espacio de direcciones y la aplicación debe manejar los accesos de vista cruzada.

Evitar copias de datos a través de archivos asignados en memoria tiene otros inconvenientes. Asegurar la coherencia semántica entre los archivos asignados en memoria y las operaciones de lectura/escritura requiere una coordinación compleja entre los sistemas de E/S, la memoria y los archivos. La E/S asignada en memoria impone la sobrecarga de la finalización sincrónica, ya que una pérdida por falta de página en una dirección virtual que se asigna a una región de archivo detiene el hilo hasta que la página está disponible en la memoria física.

Las técnicas de memoria virtual de copia en escritura también se han utilizado para ocultar el costo de las operaciones de copia. Estas técnicas de copia en escritura imitan a las aplicaciones y las páginas de caché de búfer basándose en el supuesto de que es raro que las aplicaciones modifiquen los búferes de entrada en su lugar. Esto le da al sistema de archivos la oportunidad de almacenar en caché las mismas páginas físicas sin una copia. Para algunas cargas de trabajo, esta optimización puede evitar la operación de copia al precio de una complejidad considerable, especialmente cuando la aplicación y las pilas de almacenamiento están en diferentes dominios de protección. Otras técnicas, como la reasignación de páginas de memoria virtual, han sido útiles para las rutas de recepción de red en ciertas condiciones cuando los búferes de las aplicaciones están alineados correctamente.

La Figura 2 ilustra un diagrama de flujo de un procedimiento 200 para proporcionar un búfer inmutable. La Figura 3A ilustra un entorno 300A en el que se produce un procedimiento de llenado de un búfer. La Figura 3B ilustra el entorno 300B en el que el búfer poblado se hace inmutable. En consecuencia, el procedimiento 200 de la Figura 2 se describirá ahora con referencias frecuentes a las Figuras 3A y 3B. Los entornos 300A y 300B pueden ocurrir dentro del sistema informático 100 de la Figura 1, aunque no son necesarios. Los entornos 300A y 300B pueden distribuirse o ubicarse en un solo sistema informático.

Los datos de origen que se van a utilizar para poblar el búfer son accedidos primero por una entidad informática adquirente (acto 210). Los datos de origen pueden ser cualquier dato, pero en una realización, los datos de origen incluyen grandes cantidades de datos masivos que requieren recursos informáticos significativos para ser generados. En esta descripción y en las reivindicaciones, una "entidad informática" es cualquier componente, módulo, procedimiento, función, proceso, procesador o cualquier combinación de los mismos, que pueda procesar datos en un sistema informático. Dicha entidad informática puede estar distribuida o residir en un solo ordenador.

La entidad informática adquirente puede generar todos o algunos de los datos de origen (acto 211). Alternativamente o, además, la entidad informática adquirente puede adquirir todos o algunos de los datos de origen de una fuente de datos (acto 212). Por ejemplo, con referencia a la Figura 3A, la entidad informática

adquirente 320 adquiere (como se representa por la flecha 301) los datos de origen 311 de una fuente de datos 310. La fuente de datos puede ser, por ejemplo, una red o un dispositivo de almacenamiento no volátil, como un disco.

La entidad informática adquirente también adquiere un búfer (acto 220). Esta adquisición de búfer (220) se muestra en paralelo con la adquisición de datos de origen (acto 210), ya que en el aspecto más amplio de los principios descritos en la presente memoria no se requiere que ocurra primero ninguno de los dos. Sin embargo, en algunos sistemas, uno puede ser requerido antes que el otro y/o sus actos pueden ocurrir al menos parcialmente al mismo tiempo. Haciendo referencia a la Figura 3A, la entidad informática adquirente 320 adquiere el búfer 330 en la medida en que la entidad informática adquirente puede llenar el búfer 330 con datos.

Independientemente de si la entidad informática adquirente genera los datos de origen, o recibe los datos de origen de una fuente de datos, o ambos, la entidad informática adquirente llena el búfer con datos (acto 230). Refiriéndose a la Figura 3A, por ejemplo, la entidad informática adquirente 320 llena (como se representa por la flecha 302) los datos 311 en el búfer 330.

El búfer se clasifica entonces como inmutable (acto 240). La Figura 3B ilustra un entorno 300B que es similar al entorno 300A de la Figura 3A, excepto que los datos 311 se muestran asegurados dentro del búfer 330, que se muestra con un límite sombreado 331 que representa de manera abstracta que el búfer 330 ahora es inmutable. Esta clasificación protege los datos (por ejemplo, los datos 311) que se rellenan dentro del búfer inmutable (por ejemplo, el búfer 330 en la Figura 3B) para que no cambien durante la vida útil del búfer inmutable, y también protege al búfer inmutable para que no se modifique su dirección física durante la vida útil del búfer inmutable. Debido a esta característica inmutable, el acceso a los datos inmutables puede otorgarse a un número arbitrario de entidades informáticas sin riesgo de conflicto, ya que cada una de esas entidades informáticas solo puede observar los datos.

En un entorno de lenguaje nativo (como C o C++), esta inmutabilidad puede lograrse escribiendo en una Unidad de Administración de Memoria (MMU, Memory Management Unit) de un procesador para restringir la escritura del procesador a ciertos intervalos de memoria. Esto puede ser bastante costoso, y la restricción en los accesos a la memoria no es muy granular, y se logra a menudo en el nivel de página relativamente grande. Además, ésta puede ser una operación costosa, y no evita las circunstancias en las que se realiza la copia para ocultar datos de diferentes niveles con granularidades más pequeñas que el nivel de la página.

En un entorno administrado (un entorno que incluye un tiempo de ejecución administrado), el software se utiliza para declarar la memoria como inmutable y para imponer la inmutabilidad. Además, la vida útil de un búfer de memoria se puede mantener a través de un recuento de uso, que se incrementa cuando se asigna un nuevo puntero a la memoria a una entidad, y disminuye cuando la entidad ya no utiliza un puntero a la memoria. Cuando la cuenta de uso vuelve a cero, el búfer es inalcanzable y el administrador de memoria puede reclamarlo. En una realización, el núcleo (kernel) otorga autoridad a diferentes entidades para acceder a la memoria y mantiene un conteo de uso, mientras que un tiempo de ejecución administrado proporciona vistas en la memoria inmutable, impone la inmutabilidad y proporciona restricciones en los datos. A continuación, se describe más sobre los entornos administrados con respecto a la Figura 12.

La Figura 4 ilustra un diagrama de flujo de un procedimiento para usar el búfer inmutable. Primero, un componente de vista ofrece vistas flexibles en el búfer inmutable (acto 401), y luego proporciona los puntos de vista según corresponda con diferentes consumidores del búfer inmutable (acto 402). La entidad informática puede acceder al búfer inmutable solo a través de su vista respectiva (acto 403). Por ejemplo, al hacer referencia al entorno 500 de la Figura 5, una primera entidad informática 501 accede (como se representa por la flecha 521) al búfer inmutable 330 a través de una primera vista 511, y una segunda entidad informática 502 accede (como se representa por la flecha 522) al búfer inmutable 330 a través de una segunda vista 512. Las elipsis 513 muestran que esto puede continuar por más que solo estas dos entidades informáticas 501 y 502. Las vistas 511 y 512 pueden ser vistas diferentes, pero también pueden ser la misma vista. En cualquier caso, el componente 520 de vista es capaz de proporcionar diferentes vistas del búfer inmutable 330 subyacente. En esta descripción, los términos "primero" y "segundo" se usan simplemente para distinguir un elemento de otro, y no implican ningún tipo de secuencia, prioridad, posición o importancia.

En algunas realizaciones, las entidades informáticas que consumen datos del búfer inmutable están en lados diferentes de una protección o límite de proceso. Por ejemplo, la Figura 5 ilustra que las entidades informáticas 501 y 502 que consumen datos a través de sus respectivas vistas 511 y 512 están en realidad separadas por el límite 530. Por ejemplo, las entidades de cálculo 501 y 502 pueden ser procesos distintos, en cuyo caso el límite 530 representa el límite entre procesos. El límite 530 también puede ser un límite de protección, en cuyo caso uno no puede proporcionar datos directamente al otro sin copiar. Por ejemplo, la entidad informática 501 podría ser un componente del núcleo dentro del sistema operativo, mientras que la entidad informática 502 podría ser un componente de modo de usuario tal como un componente de aplicación.

Normalmente, los datos no se comparten a través de los límites de proceso y de la protección, a menos que se copien los datos. Dicha copia puede requerir una cantidad significativa de recursos informáticos, especialmente si la cantidad de datos copiados es muy grande, o si se deben compartir con frecuencia porciones diferentes de los datos a través de dichos límites. Los principios descritos en la presente memoria proporcionan un mecanismo conveniente y flexible para compartir datos a través de los límites de proceso y protección sin copiarlos. Esto mejora de esta forma el rendimiento del sistema operativo.

Las vistas proporcionadas por el proveedor de vista 520 pueden ser de granulación fina. Por ejemplo, suponga que los datos inmutables que se leerán del búfer inmutable son datos de red. Las distintas capas de la pila de protocolos pueden estar interesadas en diferentes porciones de esos datos de red. Los componentes de nivel de red (como un componente de Protocolo de Internet) pueden estar interesados en los encabezados de nivel de red, mientras que el componente de nivel de aplicación puede estar simplemente interesado en la carga útil en bruto. Entre estas dos capas hay diferentes componentes que están interesados en diferentes partes de los datos de red.

Los principios descritos en la presente memoria pueden aplicarse eficazmente al procesamiento de estos datos de red sin requerir que se copien los datos de red. Por ejemplo, el nivel más bajo de la pila de protocolos puede ver todo el paquete de red. Ese nivel más bajo puede procesar el encabezado más externo de ese paquete y devolver una definición de vista al siguiente componente de nivel superior en la pila de protocolos. La definición de la vista define el alcance completo del paquete de red, excepto el paquete más externo. Este segundo componente proporciona la definición de vista al proveedor de vista 520, que proporciona esta vista al segundo componente. Por lo tanto, el componente más bajo ve el paquete completo, mientras que el siguiente componente ve el mismo paquete sin el encabezado más externo. Esto se hizo sin copiar datos en absoluto. En cambio, los datos se mantuvieron dentro del búfer inmutable. Esto puede repetirse hasta que la capa de aplicación más alta reciba una definición de vista que define solo la carga útil del paquete.

La Figura 6 ilustra un diagrama de flujo de un procedimiento 600 para comunicar los datos inmutables de una entidad informática a la siguiente. Una primera entidad informática accede a una definición de vista (acto 601) y proporciona esa definición de vista a un proveedor de vista (acto 602). El proveedor de vista luego proporciona la vista a la primera entidad informática (acto 603). Después de que la primera entidad informática realiza su lógica (acto 604), puede proporcionar otra definición de vista a la siguiente entidad informática (acto 605) que debe procesar los datos del búfer inmutable. La siguiente entidad informática puede entonces repetir este procedimiento 600 y, por lo tanto, el proceso puede continuar a través de múltiples capas del sistema.

Mientras que lo anterior describe el consumo de búferes/flujos en una forma de copia cero, los principios descritos anteriormente también pueden aplicarse a la producción de búferes y flujos por parte de un productor de datos. En el caso de un productor de datos, también hay flexibilidad para que la aplicación envíe sus propios búferes (asignados por separado) o para solicitar al productor de datos que proporcione vistas de escritura (tramas) en su propio búfer interno. Esto potencialmente no solo elimina la copia, sino que también mejora la utilización del búfer al eliminar la necesidad de enviar búferes medio llenos.

Transmisión de datos de copia cero

El movimiento de datos masivos a través de un sistema operativo a menudo se modela utilizando una arquitectura de flujo. Un flujo representa un conducto lógico entre una fuente de datos y un consumidor de datos, lo que permite que los datos producidos por el origen se entreguen a su destino. Los flujos generalmente implementan el almacenamiento en búfer para acomodar las incongruencias de rendimiento entre el productor y el consumidor.

Por ejemplo, la Figura 7 ilustra un entorno de transmisión 700 en el que se proporciona un flujo de datos 711 (como se representa por la flecha 701) desde una fuente de flujo 710 a un búfer de flujo 720, y luego se proporciona (como se representa por la flecha 702) desde el búfer 720 hasta un consumidor de flujo 730. El entorno 700 también incluye un administrador de flujo 740 que realiza el control de flujo. El administrador de flujo 740 hace que las porciones de flujo sean alimentadas al consumidor de flujo 730 desde la memoria intermedia 720 (representada por la flecha 702) a una velocidad satisfactoria para el consumidor de flujo 730. Además, el administrador de flujo 730 realiza una lectura adecuada antes del flujo (como lo representa la flecha 701) para garantizar que la cantidad de porciones de flujo dentro del búfer de flujo 720 no sea tan poca que el consumidor de flujo 730 corra el riesgo de quedarse sin porciones de flujo, y no tantas, que se ocupe una cantidad innecesaria de memoria del búfer de flujo 720. El administrador de flujo 740 también administra el tiempo de vida de las porciones de flujo dentro del búfer de flujo 720, de modo que la memoria ocupada por la porción de flujo se puede reclamar una vez que se consume la porción de flujo.

Los flujos a menudo cruzan lógicamente múltiples límites de proceso y/o de protección. Por ejemplo, cuando una aplicación lee datos de un archivo, los datos a menudo se leen desde el disco físico bajo el control de un controlador de dispositivo en modo protegido. Luego, los datos pasan a través de una capa del sistema de archivos y finalmente se ponen a disposición del código de la aplicación. A menudo, el cruce de capas puede implicar la copia de datos que afecta el rendimiento y el consumo de energía.

Sin embargo, los principios del búfer inmutable de copia cero descritos anteriormente se pueden usar para formular un búfer de flujo (como el búfer de flujo 720) en el que se elimina la necesidad de copiar porciones de flujo a través de límites de procesos o de protección.

Específicamente, supongamos que se establece un búfer inmutable (como el descrito con respecto a las Figuras 2 a 6) para cada una de las múltiples porciones de flujo en el flujo. Además, suponga que el procedimiento de la Figura 2 y los procesos de las Figuras 3A y 3B se realizan para crear un búfer inmutable asociado que contiene una sola porción de flujo, cada vez que se recibe una porción de flujo.

Un búfer inmutable de este tipo permite que cualquier dato, incluidas las partes de flujo, se pase a través de diferentes capas y componentes del sistema, lo que permite que cada uno tenga su propia vista específica sobre los datos, sin necesidad de copiar los datos, como se describe con respecto a datos generales en las Figuras 5 y 6. El búfer de flujo 720 en ese caso sería simplemente una colección de búferes inmutables, cada uno con una porción de flujo correspondiente contenida como datos en el mismo. A medida que se consume cada porción de flujo, se permite reclamar la memoria del búfer inmutable correspondiente. Por lo tanto, la transmisión de datos de copia cero es posible usando los principios descritos aquí.

Almacenamiento en caché de copia cero

El almacenamiento en caché es un aspecto importante del subsistema de E/S de cualquier sistema operativo. La latencia se reduce y el rendimiento efectivo aumenta al aprovechar el hecho de que los patrones de acceso a los datos tienden a agruparse y los mismos datos a menudo se recuperan varias veces. El almacenamiento en caché tradicional se realiza al tener grupos de memoria dedicados en distintas capas del sistema operativo administrados de manera independiente, cada uno con políticas de reemplazo y sustitución ortogonales. El acceso a los datos desde una caché a menudo implica copiar datos de búferes de caché en búferes de aplicaciones.

Los principios descritos anteriormente con respecto a las Figuras 2 a 6 permiten compartir búferes inmutables entre procesos y a lo largo de límites de protección, a través de los cuales no se pueden realizar llamadas a funciones, sino más bien una comunicación entre procesos mucho más costosa o una comunicación de límites de protección cruzada debe ser utilizado para la comunicación a través de límites.

Estos principios pueden usarse para implementar una caché. A medida que los datos fluyen de las operaciones de acceso directo a la memoria (DMA), los datos se introducen en el sistema como búferes inmutables (como el búfer 330 inmutable de las Figuras 3A, 3B y 5). Los búferes inmutables pueden circular alrededor del sistema para comunicar los nuevos datos y al mismo tiempo pueden ser instantáneas en una caché para su posterior reutilización. Cuando surge una solicitud posterior de los datos, el mismo búfer inmutable se puede recuperar de la caché y reutilizarlo, todo sin copiar ni acceder a los datos subyacentes. Esto conduce a ganancias sustanciales de eficiencia.

Por ejemplo, una ganancia de eficiencia sustancial que podría ocurrir en un sistema de código administrado resulta de la mitigación en los costos de recolección de basura en los sistemas de código administrado. La copia de datos requiere asignaciones de memoria de almacenamiento dinámico, lo que aumenta la cantidad de trabajo requerido para los recolectores de basura. Este aumento en el trabajo de recolección de basura es especialmente oneroso para los búferes de E/S, ya que generalmente son mucho más grandes que los objetos de montón normales. A pesar de caracterizar los búferes de E/S como cachés inmutables, la recolección de basura requerida en un sistema de código administrado se reduce significativamente.

De acuerdo con los principios descritos en la presente memoria, cuando una entidad informática tiene una caché que se basa en los datos subyacentes en el búfer inmutable, la entidad informática tiene una referencia "fuerte" a los datos subyacentes en el búfer inmutable, y puede utilizar esa referencia fuerte para acceder a los datos del búfer inmutable. El uso del término "fuerte" para modificar la referencia se usa simplemente para distinguir la referencia de lo que se denominará referencias "suaves" y "débiles" a continuación. Del mismo modo, el uso del término "débil" y "suave" para modificar la referencia se utiliza simplemente para distinguir las referencias entre sí y de una referencia fuerte.

Mientras una entidad tenga una referencia fuerte a un búfer inmutable dentro de la caché, se garantiza que el búfer inmutable y sus datos continúen existiendo durante al menos la duración de la referencia fuerte para cada entidad que tenga la referencia fuerte. No se puede utilizar una referencia "suave" a un búfer inmutable para acceder a los datos del búfer inmutable sin convertir primero la referencia suave en una referencia fuerte. Una referencia fuerte se puede convertir en una referencia suave una vez que se complete el acceso a los datos.

La referencia suave se puede utilizar como una forma de sugerencia de gestión de memoria. Si solo hay referencias suaves a un búfer inmutable dado por cualquier entidad informática y el sistema se está quedando sin memoria, el sistema puede elegir recuperar la memoria que respalda ese búfer inmutable. Si esto ocurre, el siguiente intento de convertir la referencia suave en una referencia segura fallará. Los contenidos del búfer se pierden y la entidad

informática tendría que volver a generar los contenidos de otro búfer inmutable desde la fuente de datos.

Esta referencia suave es una forma valiosa de usar la mayor cantidad de memoria del sistema posible para las memorias caché sin requerir una alta precisión para ajustar los tamaños de las memorias caché en el sistema. Por ejemplo, una caché puede optar por mantener una gran parte de sus datos como referencias suaves en lugar de fuertes. El uso de memoria de otro proceso puede aumentar lo suficiente como para llevar el sistema a un estado de memoria baja. El sistema puede reaccionar rápidamente y liberar memoria de estas referencias suaves sin necesidad de tomar ninguna decisión acerca de cuánta memoria debe proporcionar a qué proceso.

Una entidad informática también puede tener una referencia "débil" a un búfer inmutable dado. Al igual que con una referencia suave, una referencia débil debe convertirse en una referencia "fuerte" para permitir el acceso a los datos dentro del búfer inmutable. Una referencia fuerte también se puede convertir en una referencia débil. La referencia débil proporciona una segunda forma de administración de memoria para estos búferes. Se utiliza para retener el acceso potencial a un búfer inmutable sin hacer que la entidad informática que contiene la referencia débil se cargue con la memoria utilizada por ese búfer. Si solo hay referencias débiles a un búfer inmutable dado por cualquier proceso, entonces el búfer subyacente puede liberarse inmediatamente.

Las referencias débiles a los búferes inmutables se pueden usar para mitigar el costo de las comunicaciones de límites entre procesos y de protección cruzada que se necesitarían para recuperar la referencia fuerte al búfer inmutable de otro proceso que tiene una referencia fuerte al búfer inmutable. Es decir, se podría crear una caché de referencias débiles en una entidad informática (por ejemplo, un proceso) para mitigar los costos de recuperar esos búferes de otra entidad informática (por ejemplo, otro proceso), incluso si ya estaban almacenados en caché por esa otra entidad informática.

La Figura 9 ilustra un diagrama de flujo de un procedimiento 900 para que una segunda entidad informática lea primero desde una caché soportada por una primera entidad informática. La Figura 10 ilustra un diagrama de flujo de un procedimiento 1000 para que la segunda entidad informática lo lea posteriormente de la caché soportada por la primera entidad informática. Juntos, los procedimientos 900 y 1000 permiten que la segunda entidad informática construya una caché local basada en la caché que posee la primera entidad informática. Los procedimientos 900 y 1000 pueden realizarse en el contexto del entorno 800 de la Figura 8, y por lo tanto se describirán con referencia frecuente a la Figura 8.

Con referencia en primer lugar al entorno 800 de la Figura 8, una primera entidad informática 810 tiene una caché 811 de datos soportados por el búfer inmutable 801. Una segunda entidad informática 820 adquiere datos de un búfer inmutable también. La segunda entidad informática 820 también se encuentra para mantener una caché 812 de datos derivados del búfer inmutable 801. Sin embargo, la caché 812 es una caché débil en el sentido de que no puede esperar un comando de liberación de la segunda entidad informática antes de que deje de existir. Por lo tanto, la segunda entidad informática 820 no tiene control sobre cuándo se libera su caché 812.

Un límite 830 (un inter procesador o límite de protección) está entre la primera entidad informática 810 y la segunda entidad informática 820). En un ejemplo, implementación, supongamos que la primera entidad informática es un sistema de archivos, y la segunda entidad informática es un servidor web que sirve y/o procesa los archivos proporcionados por el sistema de archivos.

Cuando la primera entidad informática adquiere la caché (por ejemplo, una caché de archivos en el caso de un sistema de archivos), la primera entidad informática obtiene un acceso más rápido y más local a los datos (de ahí el término "caché"), pero también adquiere una referencia fuerte al búfer inmutable que soporta la caché. La referencia fuerte proporciona la garantía de que el búfer inmutable (y sus datos) continuarán existiendo por lo menos mientras el primer sistema informático continúe sosteniendo la referencia fuerte (y potencialmente más si otras entidades también tienen referencias sólidas al búfer inmutable). En este estado, ingresamos a la descripción de la Figura 9, que ilustra un procedimiento 900 para que la segunda entidad informática (por ejemplo, la segunda entidad informática 820) lea inicialmente desde la caché (por ejemplo, una caché 811) admitida por la primera entidad informática (por ejemplo, una primera entidad informática 810).

La segunda entidad informática se comunica con la primera entidad informática para obtener una referencia fuerte a los datos inmutables (acto 901). Esta es una comunicación entre procesos o de límites de protección cruzada, y por lo tanto es una comunicación costosa. Sin embargo, bien puede ser la única comunicación de límites cruzada requerida mientras el búfer inmutable que soporta la caché continúe existiendo. Por ejemplo, suponga que el servidor web recibió una primera solicitud de un archivo contenido dentro de la caché. Esta solicitud inicial puede hacer que el servidor web realice esta comunicación inicial y obtenga una referencia segura al búfer inmutable del sistema de archivos. Usando esta referencia fuerte, la segunda entidad informática puede leer datos del búfer inmutable (acto 902). Tras o después de leer los datos de la caché, la segunda entidad informática reduce la referencia fuerte al búfer inmutable que a una referencia débil al búfer inmutable (acto 903).

La Figura 10 ilustra un diagrama de flujo de un procedimiento 1000 para que la segunda entidad informática lea

posteriormente de la caché soportada por la primera entidad informática si la caché de la segunda entidad informática no tiene los datos. Al recibir una solicitud para leer desde la caché mientras la referencia débil a la memoria caché aún existe (acto 1001), la segunda entidad informática determina si todavía existe el búfer inmutable (bloque de decisión 1002). Si el búfer inmutable aún existe ("Sí" en el bloque de decisión 1002), la segunda entidad informática convierte su referencia débil en una referencia fuerte al búfer inmutable (acto 1011), lee desde el búfer (acto 1012) (y almacena localmente en caché esos datos en la caché local 812), y de allí en adelante convierte la referencia fuerte en una referencia débil (acto 1013). Esto se hace sin realizar una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática. Más bien, la segunda entidad informática simplemente adquiere una vista en el búfer inmutable, y lee desde el búfer inmutable.

Si el búfer inmutable aún no existe ("No" en el bloque de decisión 1002), se realiza una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática para que la primera entidad informática vuelva a adquirir los datos y vuelva a crear un nuevo búfer inmutable (acto 1021). Luego, volviendo al procedimiento 900, la segunda entidad informática puede obtener una referencia fuerte a un nuevo búfer inmutable (acto 901) y leer desde el búfer (acto 902).

En un sistema de almacenamiento en caché, a menudo ocurre que el servidor (es decir, la primera entidad informática) de la caché reemplaza un elemento en esa caché (por ejemplo, la clave X tiene su valor cambiado de A a B). En este sistema de almacenamiento en caché débil/fuerte descrito aquí, una vez que el valor A para la clave X se almacena en caché como débil, el cliente (es decir, la segunda entidad informática) nunca se comunica con el servidor, por lo que no verá que el valor de X ha sido cambiado La propia copia de X del cliente tiene el valor A debido a algunos usuarios en algún lugar del sistema, manteniendo una referencia fuerte a ese valor. En algunas realizaciones, esto se soluciona invalidando todas las referencias débiles a un valor dado. Este proceso de invalidación marca el búfer inmutable subyacente de modo que ya no es posible una conversión débil a fuerte. Esto obliga a cualquier caché débil a recuperar el valor de X, lo que hará que la caché débil detecte el nuevo valor B.

Se puede considerar que la segunda entidad informática tiene una caché débil (una que puede tener que reconstruirse antes de que la segunda entidad informática se haga usando la caché débil) que se deriva de la caché fuerte de la primera entidad informática (una que permanece en el lugar en el control de la primera entidad informática). La construcción de este segundo caché "débil" encima de otro caché fuerte plantea algunos problemas con la política de reemplazo (o desalojo) en la caché de respaldo. El desalojo se refiere a un mecanismo en el que los datos menos utilizados (es decir, los datos "fríos") se eliminan (o se desalojan) de la caché para dejar espacio para los datos más utilizados (es decir, los datos "calientes"). El desalojo se basa en estadísticas sobre la frecuencia con la que se accede a ciertos elementos de datos. La caché débil 812 y la caché fuerte 811 tienen estadísticas distintas con respecto a la frecuencia de acceso de las partes de la caché, ya que ven diferentes solicitudes de datos.

Específicamente, la caché débil 812 se usará para atender las solicitudes de la segunda entidad informática 820 primero, antes de volver a la caché de respaldo 811. Por lo tanto, esta caché débil absorberá todas, excepto las referencias iniciales a datos en caliente, ocultando su utilidad para la caché de respaldo 811. Por lo tanto, cuando la caché de respaldo 811 recibe solicitudes de nuevos elementos, sin abordar esto, la caché de respaldo puede causar que los datos estén "calientes" de acuerdo con las estadísticas de la caché débil 812, para reemplazar los elementos que aún están siendo retenidos por la caché débil 812. Este reemplazo puede eliminar la última referencia fuerte/suave persistente al búfer subyacente, liberando el búfer correspondiente a la referencia débil en la caché débil. La siguiente solicitud de ese elemento contra la caché débil fallará.

De acuerdo con las realizaciones descritas en la presente memoria, este problema se resuelve comunicando la utilidad de los datos calientes (como se ve por la caché débil 812) a la caché de respaldo 811. El sistema puede proporcionar este mecanismo como un efecto secundario cuando la segunda entidad informática convierte una referencia débil para los datos en una referencia fuerte para los datos. El sistema cuenta la cantidad de veces que esto ocurre por el búfer subyacente y expone este conteo como una propiedad de metadatos en el búfer en sí. La caché de respaldo puede consultar este valor y determinar la cantidad de referencias que se han producido entre dos puntos en el tiempo. El algoritmo de reemplazo de la caché de respaldo puede utilizar esta información para mantener ese elemento vivo en ambas cachés.

La Figura 11 ilustra un diagrama de flujo de un procedimiento 1100 para que la primera entidad informática (o la caché de respaldo) realice el desalojo. Primero, la caché de respaldo usa sus propias estadísticas para identificar a los candidatos para el desalojo (acto 1101). La primera caché de respaldo concede las segundas estadísticas de la caché débil para los candidatos identificados (acto 1102). La decisión de desalojo con respecto a los candidatos identificados se compromete (acto 1103) después de haber consultado con la segunda estadística, de manera que, si la segunda estadística indica un acceso más frecuente al candidato identificado, el candidato identificado puede mantenerse dentro de la caché por el momento.

Esta política de desalojo permite arquitecturas de caché débil arbitrariamente complicadas. Debido a la naturaleza

de la política de desalojo utilizada por la caché fuerte, que cuenta las conversiones de débil a fuerte de un búfer dado, el origen de esa conversión no es relevante. Por lo tanto, es posible tener cualquier número de cachés débiles que tengan muchas referencias débiles independientes al mismo búfer subyacente, mientras que tienen el mismo efecto equivalente en la política de desalojo de la caché fuerte. Además, este efecto es el mismo incluso para arquitecturas que tienen múltiples niveles de cachés débiles anidados basados en una caché fuerte. Por lo tanto, el sistema de almacenamiento en caché se puede utilizar con diseños de sistemas ricos. Por ejemplo, un servidor web puede estar estructurado sobre un sistema de archivos en la parte superior de un almacén estructurado de registros. La caché fuerte estaría en la parte inferior del almacén estructurado de registros. El sistema de archivos tendría el primer nivel de caché débil. El servidor web tendría el segundo nivel de caché débil. El mismo elemento de datos (por ejemplo, una parte de un archivo) se puede mantener en las tres cachés.

Estos tres primeros conceptos (a saber, datos masivos inmutables de copia cero compartidos, transmisión de datos de copia cero y almacenamiento en caché de copia cero) pueden aplicarse en sistemas de código no administrado, así como en sistemas de código administrado. Sin embargo, dado que los puntos de vista proporcionados por los sistemas administrados pueden crearse más rápidamente y ser mucho más detallados que los de los sistemas no administrados, los principios pueden usarse de manera más efectiva con los sistemas administrados.

La Figura 12 ilustra un ejemplo de sistema de código administrado 1200. El sistema administrado 1200 incluye memoria administrada 1201. El sistema administrado 1200 tiene múltiples componentes de código administrado 1230, cada uno con acceso exclusivo a la memoria específica de entidad. Por ejemplo, los componentes del código administrado 1230 en ejecución se ilustran incluyendo siete componentes 1231 a 1237, aunque las elipsis 1238 representan una gran flexibilidad en este número. Por ejemplo, los componentes 1231 a 1237 pueden ser procesos.

Cada uno de los siete componentes en ejecución 1231 a 1237 tiene una memoria específica de entidad correspondiente 1211 a 1217. Un componente administrado no puede acceder a la memoria específica de entidad de otra memoria específica de entidad. Por lo tanto, existe una protección de aislamiento entre la memoria específica de entidad, de modo que solo el componente administrado correspondiente puede acceder a esa memoria específica de entidad. Por ejemplo, el componente 1231 accede a la porción de memoria específica de entidad 1211, pero no a las porciones de memoria específica de entidad 1212 a 1217; el componente 1232 accede a la porción de memoria específica de entidad 1212, pero no a la porción de memoria específica de entidad 1211 o a las porciones de memoria específica de entidad 1213 a 1217, y así sucesivamente.

La memoria de código administrado 1210 también incluye una memoria compartida 1219. Esta memoria compartida 1219 es un ejemplo del búfer inmutable 330 de las Figuras 3A, 3B y 5. Dicho esto, los principios descritos anteriormente no se basan en un sistema de código administrado. Sin embargo, los dos conceptos finales descritos en la presente memoria están limitados a los entornos administrados. Se describirán algunos elementos adicionales de la Figura 12 con respecto a la descripción de estos dos conceptos finales (a saber, Acceso uniforme a memoria y conversión de tipos en tipo seguro).

Acceso uniforme a memoria

La memoria en un entorno de lenguaje administrado es una cosa potencialmente muy dinámica. Los objetos se asignan de un montón y son administrados por un recolector de basura. Haciendo referencia a la Figura 12, el sistema administrado 1200 incluye un recolector de basura 1221. Basándose en heurísticas, el recolector de basura realiza regularmente el mantenimiento del montón al compactar los objetos para recuperar el espacio utilizado anteriormente. Compactar los objetos para que queden juntos implica que la dirección de memoria de un objeto es esencialmente inestable, sujeta a cambios por parte del recolector de basura. El recolector de basura depende de patrones de generación de código particulares y del soporte del sistema operativo para poder mover objetos de manera transparente al código de nivel de aplicación.

Un subsistema de E/S de un sistema operativo es responsable de mezclar grandes cantidades de datos a través de la memoria del sistema. En el momento de lectura, los datos normalmente se adquieren desde un dispositivo externo y se ponen en la memoria a través de una operación de DMA administrada por el propio dispositivo con una interacción mínima con el procesador. De manera similar, al escribir datos, las operaciones de la memoria de DMA pueden leer automáticamente el contenido de la memoria.

Las operaciones de DMA no pueden competir con el contenido de la memoria que se reubica durante la operación. Para hacerlo, se requeriría una coordinación precisa entre el procesador y los dispositivos, lo que impactaría dramáticamente el rendimiento y la eficiencia energética. Como resultado de esta restricción, hay dos opciones amplias para admitir DMA en un sistema operativo administrado.

Se utilizan operaciones de anclaje especiales contra regiones de la memoria para indicar al recolector de basura que no reubique los objetos especificados. Esto permite que las operaciones de DMA vean una instantánea consistente de la memoria afectada mientras se ejecutan.

Las operaciones de DMA ocurren en regiones de memoria especiales que no están sujetas a recolección de basura.

El primer enfoque puede obstaculizar sustancialmente la eficiencia del recolector de basura ya que tratar con las regiones fijas de la memoria complica el proceso de compactación y reduce su eficiencia. El segundo enfoque evita ese problema, pero puede fácilmente conducir a una copia de memoria excesiva, ya que se necesita lógica especializada para transferir datos entre las regiones de memoria normales y las regiones de memoria especiales compatibles con DMA.

Una arquitectura de acceso de memoria unificada proporciona una forma sistemática de referencia a la memoria, ya sea una memoria administrada normal o una región de memoria especial compatible con DMA. Esto hace posible que los programadores manipulen los datos en regiones de memoria compatibles con DMA directamente de una manera completamente segura, evitando la necesidad de fijar objetos y copiar entre la memoria normal y la memoria compatible con DMA.

En un entorno de lenguaje administrado, los datos masivos se mantienen típicamente en matrices. El entorno de lenguaje administrado (por ejemplo, el sistema administrado 1200) comprende directamente las matrices, lo que permite el acceso a elementos individuales de la matriz y garantiza que los programadores no puedan exceder los límites de la matriz. Al ser administrado por el entorno de lenguaje, las matrices están restringidas para ubicarse en el montón administrado.

En el sistema administrado 1200 de la Figura 12, el búfer inmutable 1219 está ubicado fuera del montón administrado y, por lo tanto, en circunstancias normales, no sería directamente accesible desde los programas administrados. La lectura y escritura desde la memoria de E/S se realizaría normalmente utilizando un paquete de E/S clásico con operaciones de lectura y escritura explícitas que inducen el copiado implícito entre la memoria normal en el montón administrado y la memoria de E/S.

El sistema administrado incluye una abstracción (denominada en la presente memoria como un "intervalo") que proporciona un mecanismo para acceder directamente al búfer inmutable 1219 desde un componente de código administrado. Haciendo referencia a la Figura 12, la parte de la memoria administrada 1211 incluye una variedad de objetos, incluida la abstracción de intervalo 1240 representada de manera abstracta. Los tramos se pueden crear de manera segura para proporcionar acceso directo a cualquier región de un búfer de E/S de una manera muy similar a la forma en que funcionan las matrices. Además, los tramos se pueden construir para hacer referencia a la memoria administrada. Por lo tanto, las abstracciones de software creadas sobre tramos pueden ser agnósticas respecto a la ubicación de la memoria subyacente del intervalo. Esto proporciona la mejor historia de composición, permitiendo que las abstracciones se diseñen de manera natural para operar en la memoria administrada (por ejemplo, las partes de memoria 1211 a 1218) o el búfer inmutable 1219.

Los intervalos se crean interactuando con el almacenamiento subyacente para el intervalo. Por ejemplo, el búfer inmutable 1219 puede proporcionar llamadas de procedimiento para devolver intervalos que hacen referencia a los búferes inmutables controlados por el intervalo directamente. De manera similar, las matrices proporcionan procedimientos que devuelven intervalos que apuntan a ellos o a partes de ellos. Una vez que se ha materializado un intervalo, se puede pasar y utilizar en gran medida, ya que las matrices se utilizan en los lenguajes administrados normales.

Una sutileza particular con intervalos se relaciona con la administración de por vida del almacenamiento subyacente. Uno de los principales beneficios de un entorno de programación administrado es que el recolector de basura asume la responsabilidad de detectar cuándo ya no se hace referencia a los objetos y se puede reciclar su almacenamiento. Esto es lo que sucede cuando las matrices ya no son útiles, por ejemplo.

Cuando la memoria subyacente a un intervalo está fuera del montón de basura normal recolectada, entonces la vida útil de esa memoria debe administrarse con cuidado, de modo que los espacios creados que hacen referencia a la memoria no sobrevivan al búfer de memoria en sí. Esto se puede organizar de varias maneras, por ejemplo, utilizando contadores de referencia en la memoria subyacente o limitando la vida útil de los intervalos.

En una realización, un objeto de intervalo contiene un puntero especialmente anotado a la región de memoria que representa. El recolector de basura entiende estos punteros especiales y los trata especialmente. Durante una operación de recolección de basura, si el recolector de basura encuentra un puntero especial, considera la dirección que contiene el puntero. Si el recolector de basura detecta que el puntero apunta afuera del montón administrado, el recolector de basura ignora el puntero completamente desde ese punto en adelante. Si, en cambio, se encuentra que el puntero apunta dentro del montón administrado, el recolector de basura trata al puntero como una referencia a un objeto administrado y, por lo tanto, ajusta automáticamente el valor del puntero en la eventualidad de que el objeto subyacente se reubique.

Los intervalos pueden crearse para representar subregiones de otros intervalos. Esto hace que los intervalos se

conviertan en una forma muy conveniente de extraer una porción de una región de memoria más grande de una manera segura y económica sin hacer copias. El intervalo resultante se parece a cualquier otro intervalo, aunque tenga un alias de un subconjunto de la memoria de otro intervalo.

Conversión de tipos en tipo seguro

Una función principal de los lenguajes de programación administrados es hacer cumplir la seguridad de tipos, lo que evita que un programa tome una dirección arbitraria en la memoria y la manipule como un objeto. Por ejemplo, el sistema administrado 1200 de la Figura 12 incluye un sistema de tipo 1222 que garantiza la seguridad de tipos. Todos los objetos se adquieren explícitamente y la dirección de cada objeto está controlada firmemente por el recolector de basura (por ejemplo, el recolector de basura 1221). En tal sistema, la memoria que no está bajo el control directo del recolector de basura no puede ser utilizada directamente por el código de la aplicación. En su lugar, la memoria termina necesitando ser copiada de la memoria especial a la memoria controlada por el recolector de basura antes de que se pueda usar, lo cual es ineficiente.

Cuando los datos entran y salen de un sistema a través de las operaciones de DMA, los datos manipulados por los dispositivos de DMA típicamente tienen alguna forma inherente. Por ejemplo, cuando se escriben datos a través de DMA, parte de la estructura de datos en el montón de basura recolectada suele contener los datos que se deben escribir. Luego se usa un paso de "serialización" para transcribir los datos en esas estructuras a la forma necesaria para la operación de DMA. Este paso de serialización es tedioso, propenso a errores e ineficiente. La serialización y la deserialización suelen ser parte integrante de los lenguajes de programación administrados.

Al aprovechar la abstracción del intervalo, un modelo de propósito general permite a los programadores interactuar directamente con las regiones de la memoria de DMA usando semántica orientada a objetos. El soporte de conversión de tipos especial hace posible que el programador vea las regiones de memoria de DMA como objetos y, por lo tanto, lea y escriba directamente la memoria de forma natural, maximizando la eficiencia, mejorando la corrección y simplificando la tarea del programador. El modelo se extiende más allá de la mera memoria de DMA y también admite la semántica de conversión de tipos extendido para la memoria normal de recolección de basura.

Para mantener la seguridad de tipos, no es posible ni deseable permitir la conversión de tipos entre tipos arbitrarios. En su lugar, existen reglas específicas que restringen el conjunto de tipos permitidos que pueden estar involucrados en esta conversión de tipos extendida. Sin embargo, las reglas son bastante amplias y terminan funcionando perfectamente para el tipo de datos que generalmente participan en las operaciones de DMA.

En un lenguaje de programación administrado, siempre que se asigna memoria, se le asigna un tipo determinado. El tipo determina el significado de las diferentes partes del bloque de memoria y las operaciones que pueden realizarse contra el bloque de memoria. El tipo no se puede cambiar para el bloque de memoria hasta que la memoria quede inactiva y se recicle a través de un proceso de recolección de basura. En todo momento, el entorno de lenguaje es responsable de asignar, escribir y reciclar bloques de memoria.

La conversión de tipos es la capacidad de tratar alguna memoria como un tipo diferente al que se sabe que es el entorno administrado. La conversión de tipos es común en el lenguaje de programación nativo, pero los lenguajes administrados generalmente no ofrecen conversión de tipos. En su lugar, los entornos administrados proporcionan mecanismos de conversión de tipos que permiten copiar un tipo de valor en otro tipo. Por ejemplo, es posible convertir un valor entero en un valor de punto flotante. Sin embargo, esto siempre se hace a través de la copia: la ubicación de la memoria original permanece sin cambios.

De acuerdo con los principios descritos en la presente memoria, la conversión de tipos se introduce como una instalación de propósito general en lenguajes administrados. Las restricciones se aplican para garantizar que se mantiene la seguridad de tipo, como se explica más adelante.

En un sistema operativo administrado, la memoria utilizada para las operaciones de E/S debe ser objetos fijados o ser regiones de memoria dedicadas a la E/S. Como se mencionó anteriormente, fijar objetos en la memoria para evitar que se muevan es costoso y causa muchos problemas en un entorno de recolección de basura. Los principios descritos en la presente memoria utilizan memoria de E/S dedicada en forma de búferes inmutables (como el búfer 330 inmutable de las Figuras 3A, 3B y 5).

La memoria de E/S está fuera del alcance del subsistema de memoria administrada. El entorno administrado no controla el tipo de esta memoria y, por lo tanto, no es posible acceder directamente a este tipo de memoria desde un programa administrado. En su lugar, un código especial de conexión (es decir, de pegamento) normalmente se usaría para permitir que esta memoria se lea o escriba usando llamadas explícitas. El acceso a cualquier tipo de datos estructurados dentro de estos búferes de E/S implica un código tremendamente ineficiente, o implica copiar datos hacia y desde la memoria de E/S a la memoria administrada normal, que también es ineficiente.

Considérese un búfer inmutable adquirido de un dispositivo de red. En este búfer, hay un encabezado de TCP que

contiene información del protocolo de red. Básicamente, hay dos formas en que los datos en el encabezado de TCP pueden usarse en un lenguaje de programación administrado. La siguiente tabla muestra ambos enfoques y las etapas involucrados en cada uno.

Entorno administrado clásico	**Conversión de tipos**
Determine el desplazamiento del encabezado de TCP en el búfer.	Determine el desplazamiento del encabezado de TCP en el búfer.
Para cada campo en el encabezado de TCP Lea el campo a través de una API como ReadInt32 o ReadInt16. Almacene el campo en el almacenamiento temporal local.	Convierta la sección apropiada del búfer inmutable en un objeto de encabezado de TCP.
Cuando se lean todos los campos, cree un objeto de encabezado de TCP en el montón administrado. Una vez más, esto implica copiar explícitamente cada campo en el encabezado de TCP, esta vez desde el almacenamiento local al almacenamiento de montón	Acceda al objeto de encabezado de TCP para realizar cualquier acción que sea necesaria.
Acceda al objeto de encabezado de TCP para realizar cualquier acción que sea necesaria.	

La obtención de un objeto de encabezado de TCP utilizable es considerablemente más rápida con la conversión de tipos que con el enfoque tradicional. El nuevo enfoque imita lo que sucede en un sistema operativo nativo, donde la matemática de punteros es posible y se aprovecha con frecuencia en este tipo de escenario. El puntero matemático no está disponible en los lenguajes de programación administrados, pero la conversión de tipos en tipo seguro ofrece la misma funcionalidad.

Las variaciones son posibles en el enfoque tradicional, lo que lleva a más o menos gastos generales. Por ejemplo, es posible que el búfer de memoria en cuestión esté directamente disponible para el programador y, por lo tanto, se pueda acceder a él de manera más eficiente que utilizando los procedimientos de lectura/escritura. Sin embargo, en ese caso, el programador aún es responsable de convertir una secuencia de bytes en un objeto de nivel superior que sea tedioso, propenso a errores y que funcione mal.

Lo que hace posible la conversión de tipos y asegura que la seguridad de tipos se conserve es que la conversión de tipos solo es posible con los tipos que están diseñados para permitirlo. Para participar en la conversión de tipos, los tipos: 1) son tipos de valor (a diferencia de los tipos de referencia), 2) están compuestos solo de otros tipos que admiten la conversión de tipos, 3) no están compuestos de referencias, 4) se definen utilizando un diseño de memoria específico, y 5) pueden tolerar cualquier patrón de bits en cualquiera de sus campos.

Estas restricciones significan que, para usarse en la conversión de tipos, un tipo no puede contener referencias a otros objetos. Resulta que estas restricciones describen a la perfección las características de los tipos definidos para representar formatos de datos como encabezados TCP y un amplio conjunto de otras estructuras de datos similares.

Como se describe, la conversión de tipos en tipo seguro se puede usar para leer o escribir en búferes de E/S que se encuentran fuera del alcance del entorno de memoria administrada, y también se puede usar para ver la memoria administrada como un tipo diferente. En particular, esta técnica es útil para ver matrices de bytes como instancias de uno o más tipos más ricos en su lugar.

La Figura 13 muestra una matriz administrada normal de bytes que tiene dos intervalos distintos apuntando hacia ella y que permiten que la aplicación vea partes de la matriz como tipos diferentes. Se puede crear cualquier número de intervalos de esta manera, cada uno con distintos tipos. Los intervalos pueden solaparse libremente, haciendo referencia potencialmente a la misma región de memoria que los diferentes tipos.

La regla que dice que cualquier patrón de bits debe ser tolerable en cualquiera de sus campos es importante para la confiabilidad del modelo. Cuando se utiliza la conversión de tipos, se introducen instancias de objetos de apariencia normal en el entorno sin haber ejecutado el constructor de tipos. Normalmente, un constructor realiza la validación de los argumentos de entrada y sirve para restringir en última instancia el conjunto de valores permitidos que conforman un objeto. Pero con la conversión de tipos, es posible crear un objeto a partir de la nada al ver un espacio de memoria existente como si fuera un tipo diferente.

El enfoque tradicional de copiar datos en un objeto distinto en el montón administrado proporciona una oportunidad para validar los datos a medida que se introducen en el constructor del objeto administrado. Esto significa que, en un sistema del mundo real, las versiones inválidas del objeto administrado nunca existen dentro del sistema, el constructor garantiza que solo se puedan crear versiones válidas. Se puede contrastar esto con la conversión de tipos donde puede aparecer cualquier patrón de bits. Si hay valores que son semánticamente inválidos, no se pueden detectar ya que la construcción del objeto no tiene lugar.

La solución al problema de corrección es introducir una capa de abstracción adicional en el software. En particular, si tomamos nuevamente el ejemplo de leer un encabezado de TCP, puede imaginar que el desarrollador ha definido dos tipos distintos: RawTcpHeader y ValidTcpHeader.

Los datos en el búfer de entrada serían de tipo de conversión a un RawTcpHeader. Dado ese objeto, entonces se puede invocar un método AcquireValidTcpHeader. Este método validaría los campos en RawTcpHeader y devolvería una nueva instancia de ValidTcpHeader que actuaría como un contenedor trivial alrededor de un RawTcpHeader y le diría al titular que tiene un encabezado válido garantizado disponible. Todo esto se hace sin una copia, simplemente la creación de un objeto de paso que es el tipo de valor ValidTcpHeader.

La presente invención puede realizarse en otras formas específicas sin apartarse de sus características esenciales. Las realizaciones descritas deben considerarse en todos los aspectos solo como ilustrativas y no como restrictivas. El alcance de la invención, por lo tanto, está indicado por las reivindicaciones adjuntas en lugar de por la descripción anterior. Todos los cambios que se encuentren dentro del significado y rango de equivalencia de las reivindicaciones deben incluirse dentro de su alcance.

REIVINDICACIONES

1. Un sistema que comprende:

un búfer inmutable (330) que protege datos (311) que se rellenan dentro del búfer inmutable (330) para que no cambien durante la vida útil del búfer inmutable (330), y el búfer inmutable también está protegido para que no se modifique su dirección física durante la vida útil del búfer inmutable;

una primera entidad informática que mantiene una caché del búfer inmutable (330) y tiene una referencia fuerte al búfer inmutable (330), en el que, mientras una entidad cualquiera tenga una referencia fuerte al búfer inmutable (330), el búfer inmutable (330) tiene garantizado que continuará existiendo durante al menos la duración de la referencia fuerte para cada entidad que tenga la referencia fuerte; y

una segunda entidad informática que se comunica con la primera entidad informática para obtener una referencia fuerte al búfer inmutable (330) y luego leer los datos del búfer inmutable (330), en el que, después de leer datos de la caché, la segunda entidad informática degrada la referencia fuerte al búfer inmutable (330) a una referencia débil al búfer inmutable (330), en el que una referencia débil al búfer inmutable (330) no garantiza que el búfer inmutable (330) continuará existiendo mientras dure la referencia débil,

en el que al recibir una solicitud para leer desde el búfer inmutable (330) mientras aún existe la referencia débil a la caché, la segunda entidad informática determina si el búfer inmutable (330) todavía existe,

si el búfer inmutable (330) aún existe, la segunda entidad informática convierte la referencia débil al búfer inmutable (330) en una referencia fuerte al búfer inmutable (330) y lee los datos del búfer inmutable (330) sin realizar una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática; y

si el búfer inmutable (330) todavía no existe, la segunda entidad informática realiza una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática para que la primera entidad informática vuelva a adquirir los datos y vuelva a crear un nuevo búfer inmutable y permita que la segunda entidad informática obtenga una referencia fuerte al nuevo búfer inmutable y lea desde el nuevo búfer inmutable.

2. El sistema de conformidad con la reivindicación 1, en el que las llamadas de función no se pueden colocar entre la primera entidad informática y la segunda entidad informática, sino que se utiliza una comunicación entre procesos o una comunicación de límites de protección cruzada para que la primera entidad informática y la segunda entidad informática se comuniquen.

3. El sistema de conformidad con la reivindicación 1, que además comprende:

un proveedor de vista configurado para proporcionar una vista diferente del búfer inmutable a la primera entidad informática y a la segunda entidad informática.

4. El sistema de conformidad con la reivindicación 1, en el que la primera entidad informática mantiene primeras estadísticas con respecto a la frecuencia de acceso de búferes inmutables dentro de una memoria caché, y en el que la segunda entidad informática mantiene segundas estadísticas con respecto a la frecuencia de acceso de búferes inmutables dentro de la caché, siendo diferentes las primeras y las segundas estadísticas .

5. El sistema de conformidad con la reivindicación 4, en el que la primera entidad informática está configurada para realizar lo siguiente con el fin de realizar un desalojo:

un acto de usar las primeras estadísticas para identificar candidatos para desalojo entre búferes inmutables en la caché;

un acto de consultar con las segundas estadísticas de al menos los candidatos identificados; y

un acto de tomar una decisión de desalojo con respecto a los candidatos identificados con respecto a cada candidato de desalojo identificado después de consultar con las segundas estadísticas, de modo que, si las segundas estadísticas indican un acceso más frecuente al candidato identificado, el candidato identificado puede mantenerse dentro de la caché por el momento.

6. El sistema de conformidad con la reivindicación 1, en el que el sistema es un sistema de código administrado.

7. El sistema de conformidad con la reivindicación 1, en el que el sistema es un sistema no administrado.

8. Un procedimiento para almacenar en caché datos de un búfer inmutable, comprendiendo el procedimiento:

un acto de una primera entidad informática que accede a una primera referencia fuerte a un búfer inmutable (330), en el que el búfer inmutable (330) protege los datos (311) que se encuentran dentro del búfer inmutable (330) para que no cambien durante la vida útil del búfer inmutable (330), y el búfer inmutable (330) también está protegido para que no se modifique su dirección física durante el tiempo de vida del búfer inmutable (330), en el que mientras una entidad cualquiera tenga una referencia fuerte al búfer inmutable (330), el búfer inmutable (330) tiene garantizado que continuará existiendo durante al menos la duración de la referencia fuerte para cada entidad que tenga la referencia fuerte;

un acto de la primera entidad informática que formula una caché utilizando datos del búfer inmutable (330);

un acto de la primera entidad informática que proporciona una segunda referencia fuerte a la segunda entidad informática; y

un acto de la primera entidad informática que libera la primera referencia fuerte cuando la primera entidad informática se completa con la caché,

en el que al recibir una solicitud para leer desde el búfer inmutable (330) mientras aún existe la referencia débil a la caché, la segunda entidad informática determina si el búfer inmutable (330) todavía existe,

si el búfer inmutable (330) aún existe, la segunda entidad informática convierte la referencia débil al búfer inmutable (330) en una referencia fuerte al búfer inmutable (330) y lee los datos del búfer inmutable (330) sin realizar una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática; y

si el búfer inmutable (330) todavía no existe, la segunda entidad informática realiza una comunicación entre procesos o de límites de protección cruzada con la primera entidad informática para que la primera entidad informática vuelva a adquirir los datos y vuelva a crear un nuevo búfer inmutable y permita que la segunda entidad informática obtenga una referencia fuerte al nuevo búfer inmutable y lea desde el nuevo búfer inmutable.

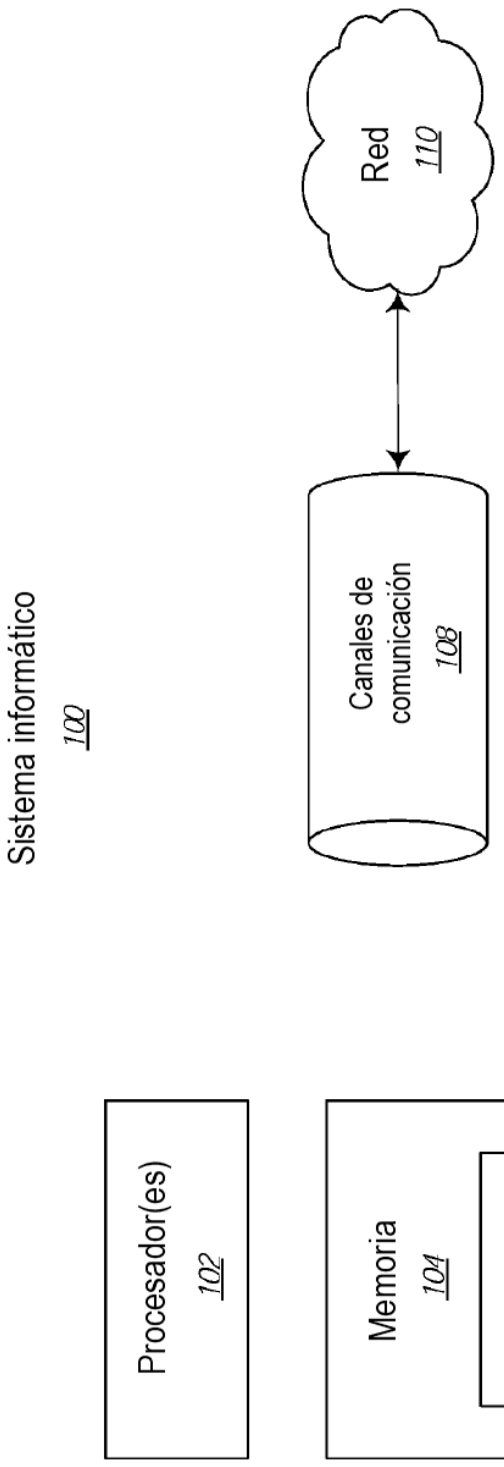
Sistema informático
100
Red
110
Canales de comunicación
108
Procesador(es)
102
Memoria
104
Volátil
No volátil

Figura 1

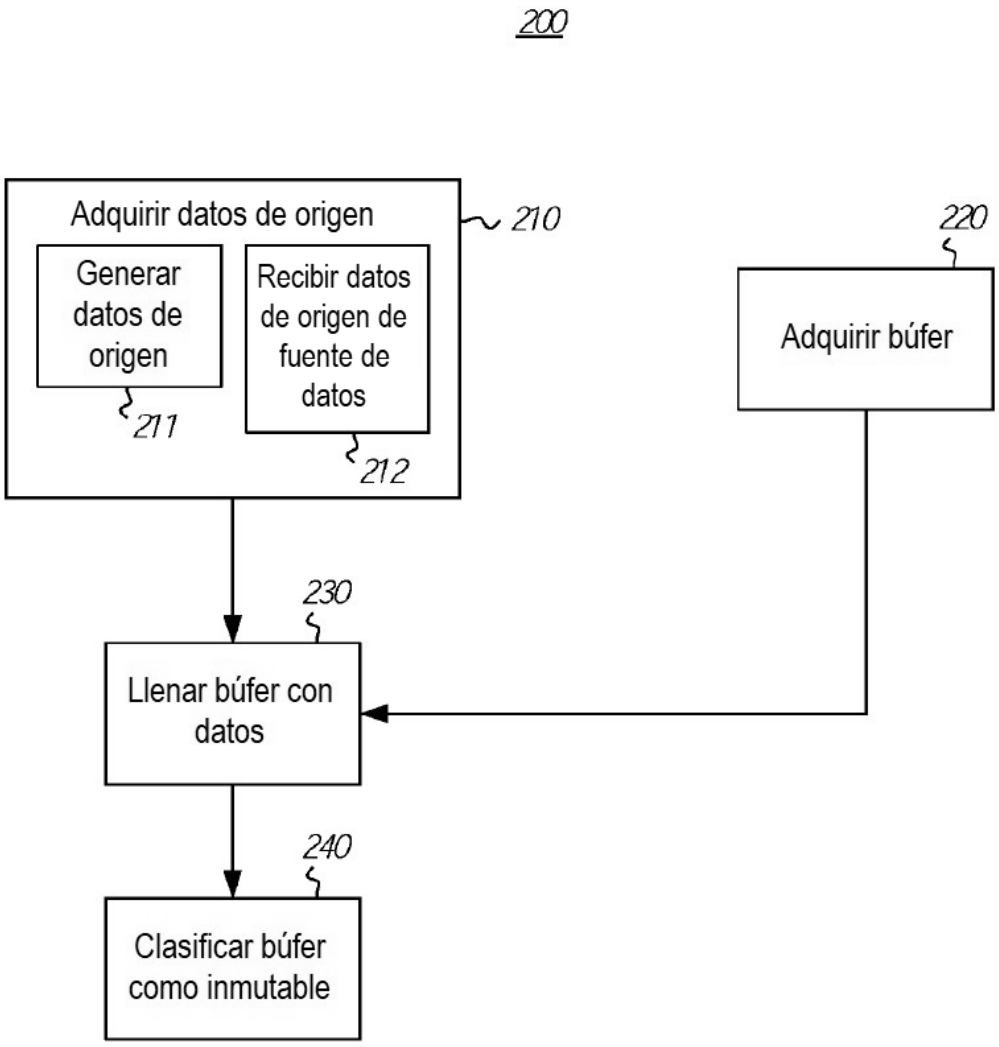
200
Adquirir datos de origen
210
Generar datos de origen
211
Recibir datos de origen de fuente de datos
212
220
Adquirir búfer
230
Llenar búfer con datos
240
Clasificar búfer como inmutable

Figura 2

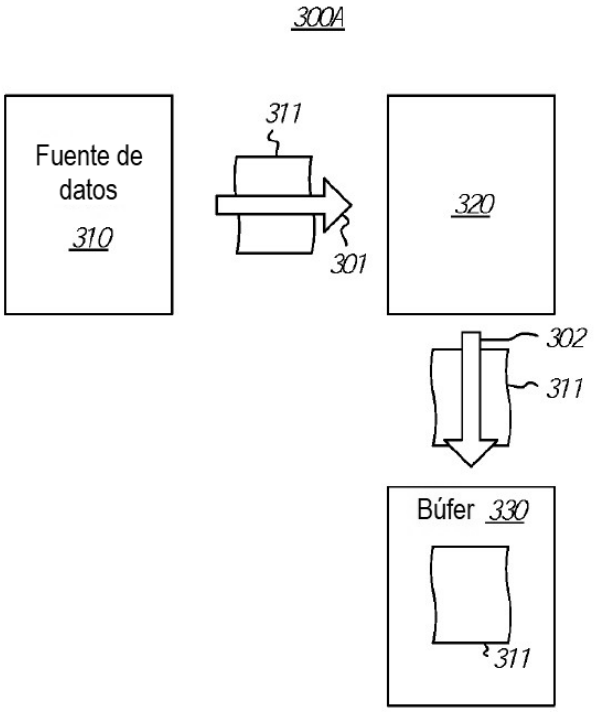

Figura 3A

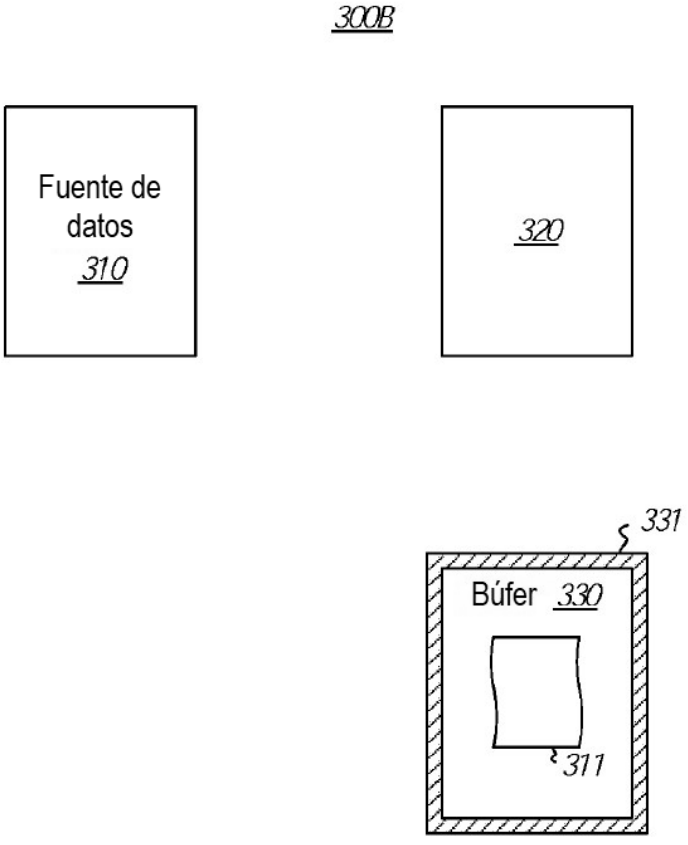

Figura 3B

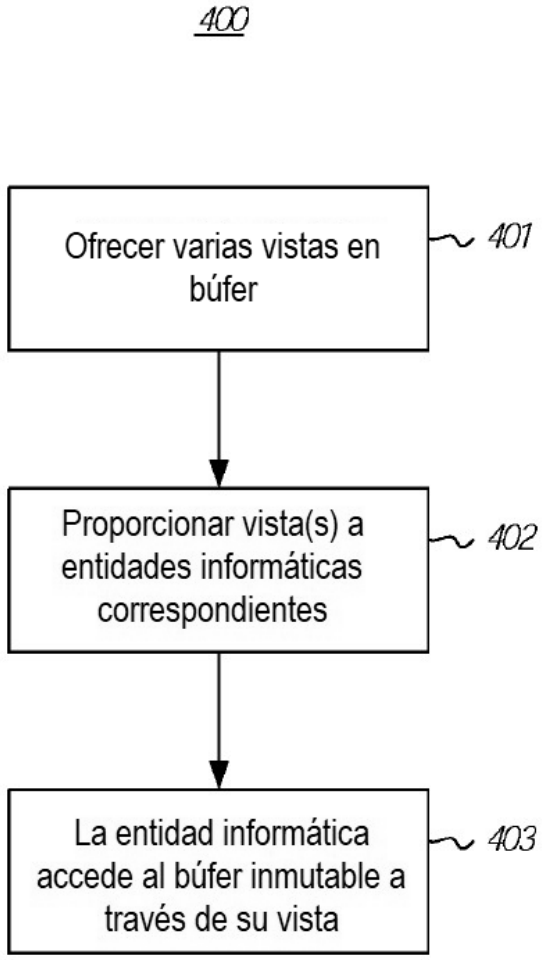
400
Ofrecer varias vistas en búfer
401
Proporcionar vista(s) a entidades informáticas correspondientes
402
La entidad informática accede al búfer inmutable a través de su vista
403

Figura 4

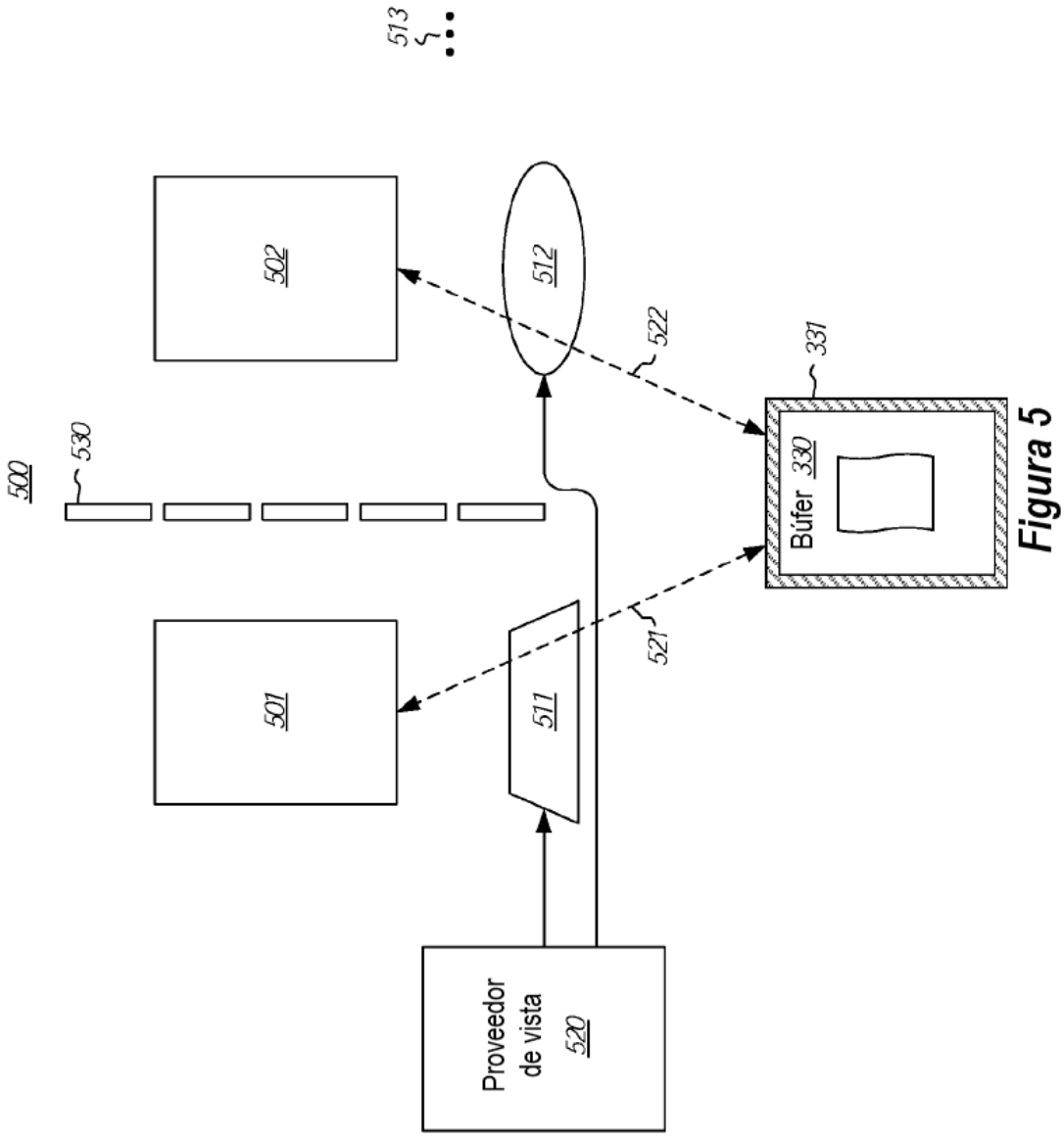
513
502
512
522
331
500
530
Búfer 330
501
511
521
Proveedor de vista 520

Figura 5

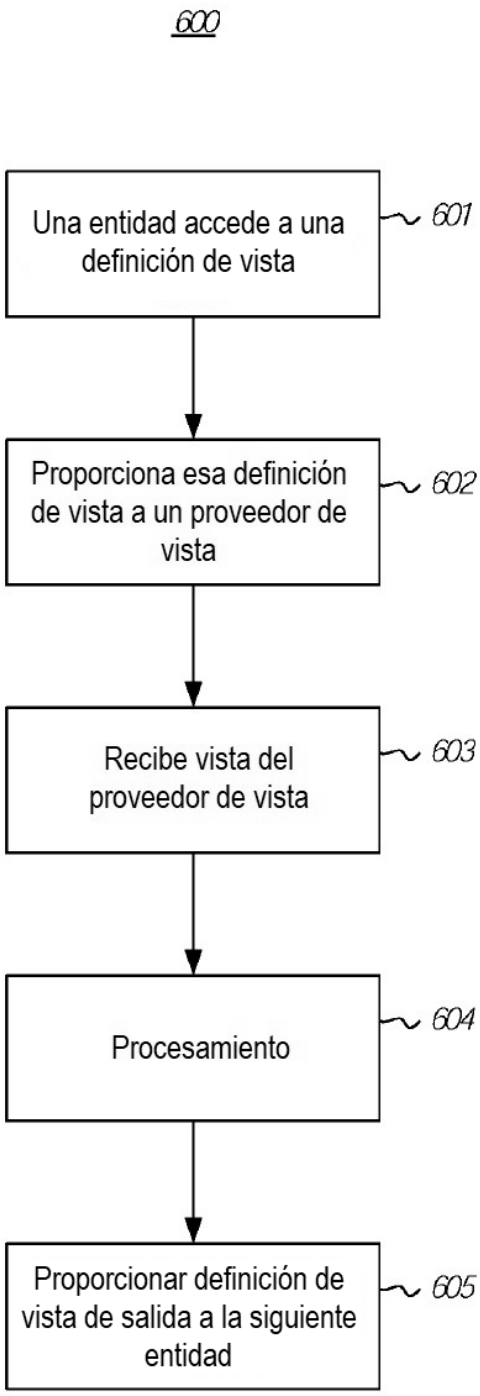
600
Una entidad accede a una definición de vista
601
Proporciona esa definición de vista a un proveedor de vista
602
Recibe vista del proveedor de vista
603
Procesamiento
604
Proporcionar definición de vista de salida a la siguiente entidad
605

Figura 6

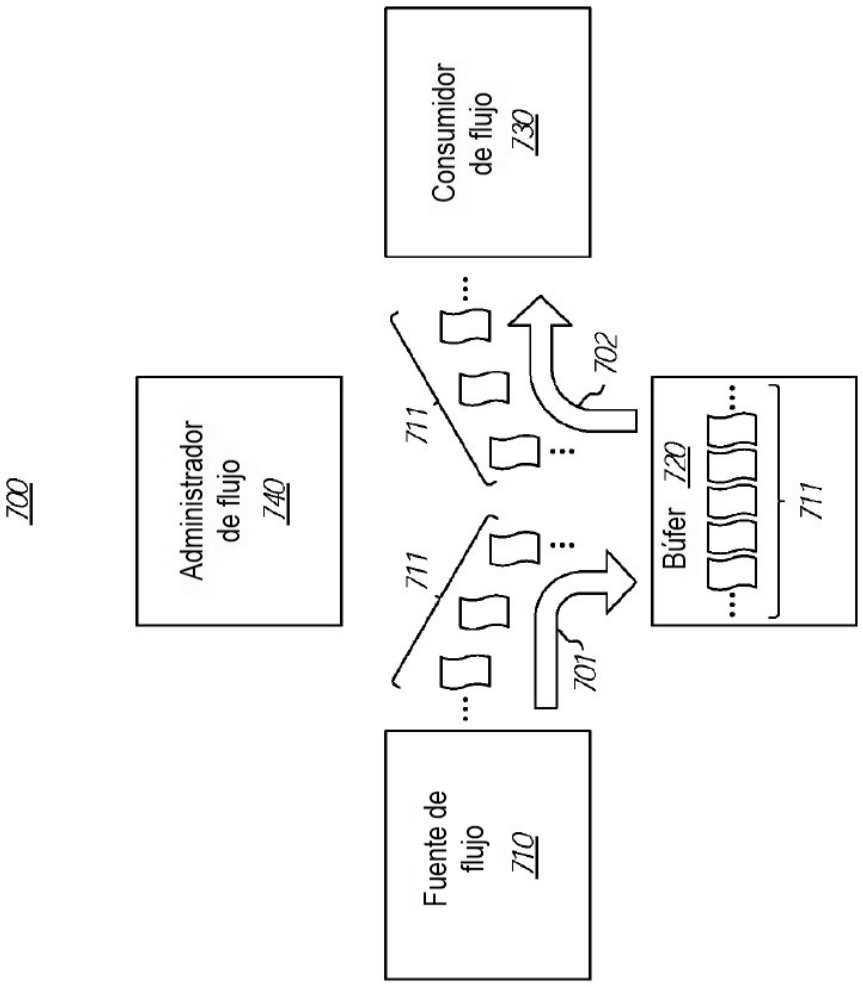
700
Administrador de flujo 740
Fuente de flujo 710
Consumidor de flujo 730
Búfer 720
711
701
702

Figura 7

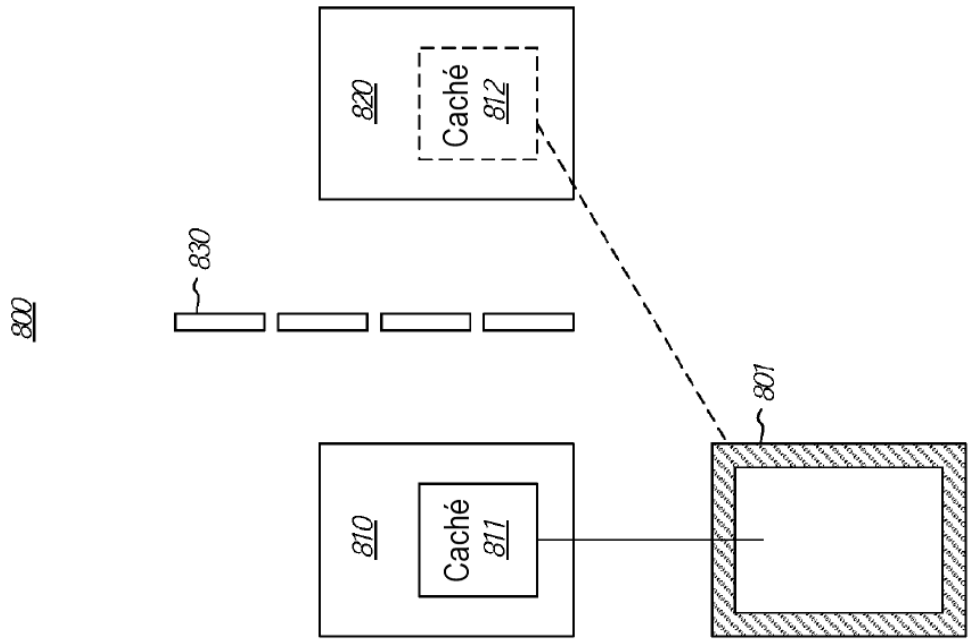

Figura 8

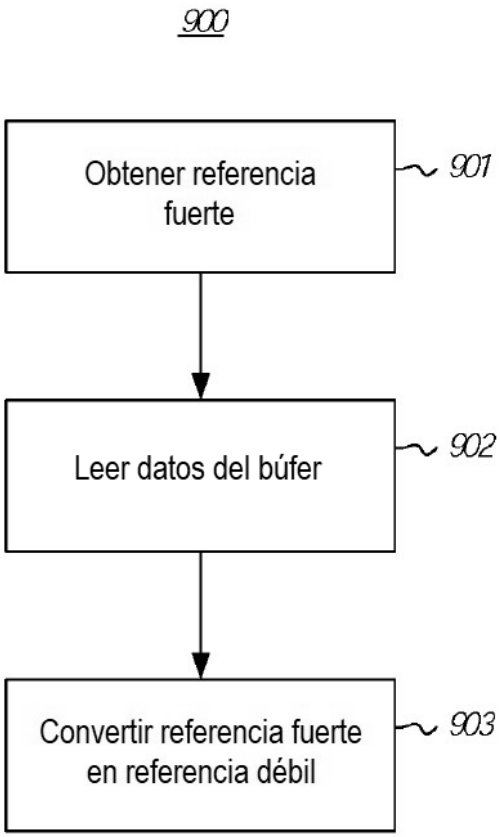
900
Obtener referencia fuerte
901
Leer datos del búfer
902
Convertir referencia fuerte en referencia débil
903

Figura 9

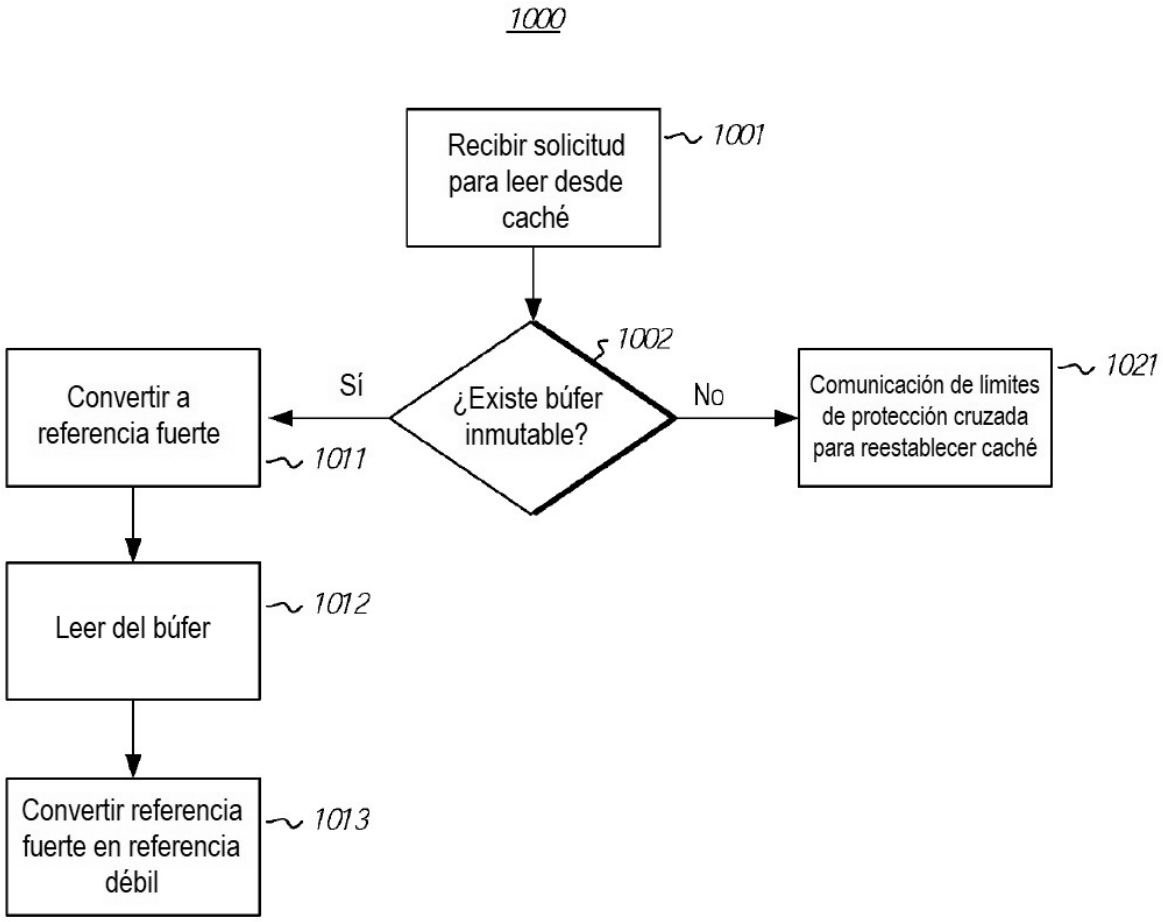
1000
Recibir solicitud para leer desde caché
1001
¿Existe búfer inmutable?
1002
Sí
No
Convertir a referencia fuerte
1011
Comunicación de límites de protección cruzada para reestablecer caché
1021
Leer del búfer
1012
Convertir referencia fuerte en referencia débil
1013

Figura 10

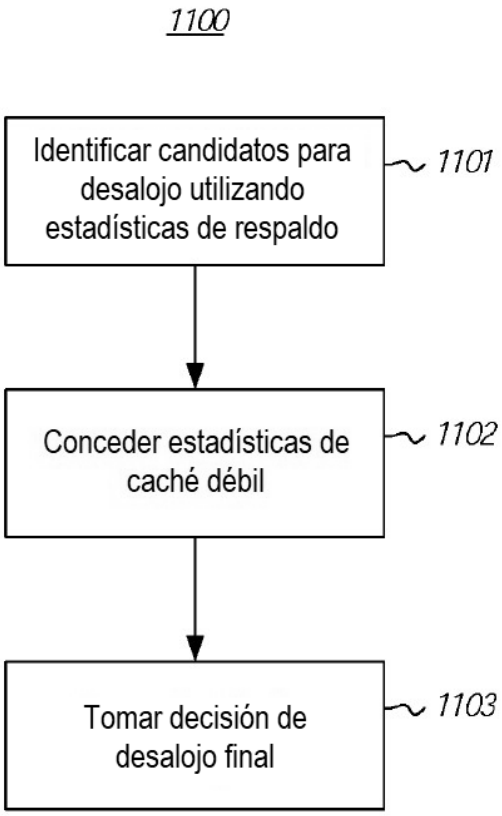
1100
Identificar candidatos para desalojo utilizando estadísticas de respaldo
1101
Conceder estadísticas de caché débil
1102
Tomar decisión de desalojo final
1103

Figura 11

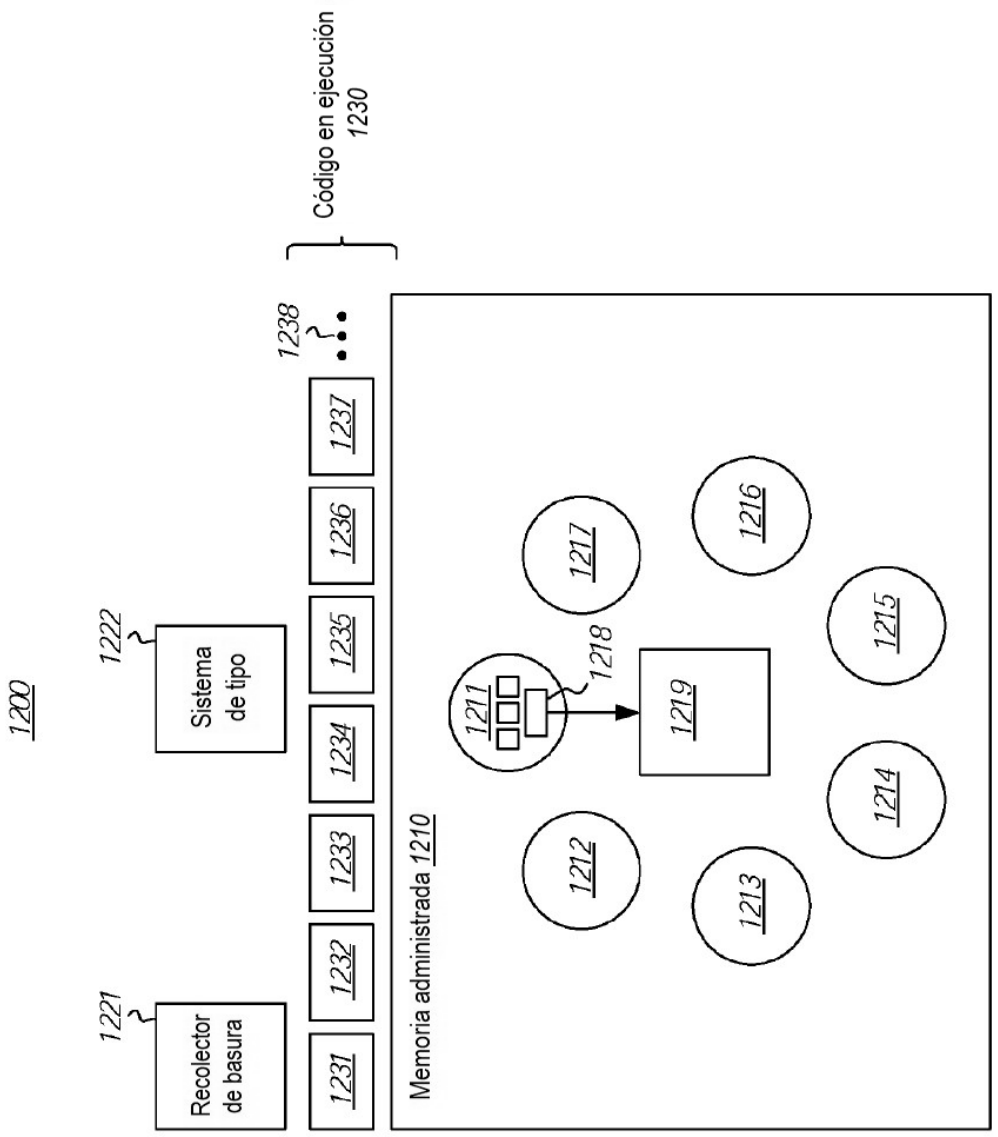
1200
1221
Recolector de basura
1222
Sistema de tipo
1231
1232
1233
1234
1235
1236
1237
1238
Código en ejecución 1230
Memoria administrada 1210
1211
1212
1213
1214
1215
1216
1217
1218
1219

Figura 12

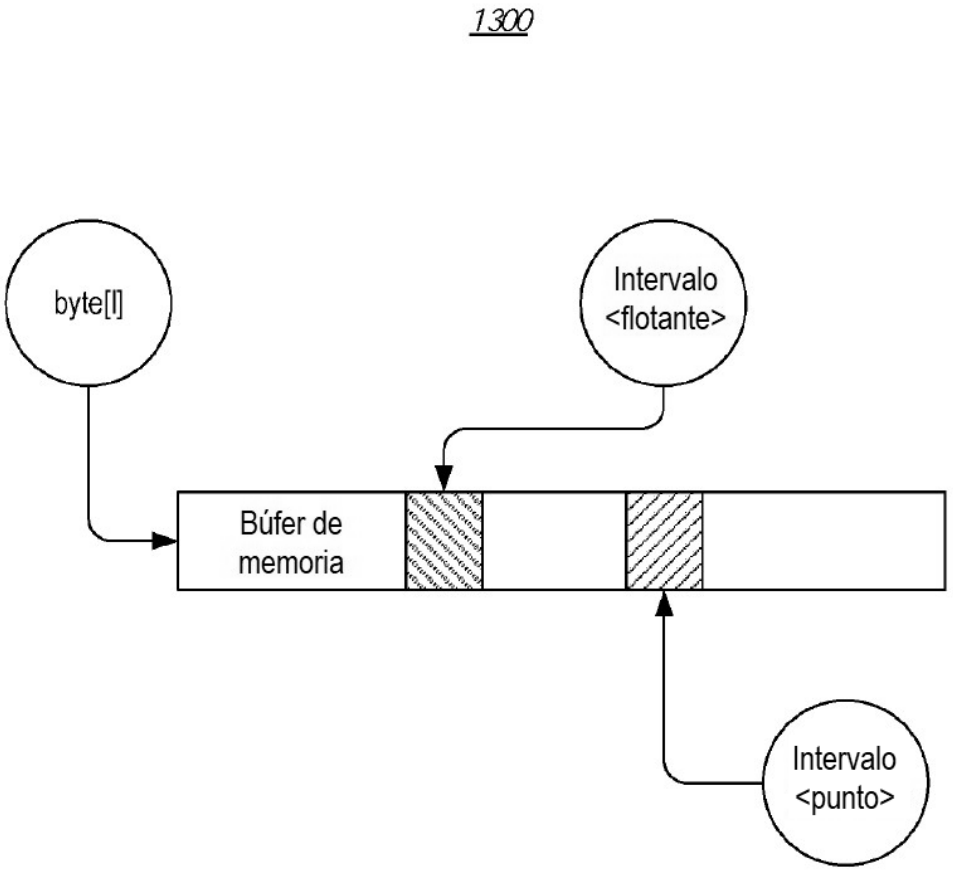
1300
byte[l]
Intervalo
<flotante>
Búfer de
memoria
Intervalo
<punto>

Figura 13