(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 364 400**

(51) Int. Cl.:
G06F 11/14 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Número de solicitud europea: **04029093 .4**
(96) Fecha de presentación : **08.12.2004**
(97) Número de publicación de la solicitud: **1544739**
(97) Fecha de publicación de la solicitud: **22.06.2005**

(54) Título: **Procedimiento y aparato a prueba de fallos para actualizaciones personalizadas de imágenes de soporte lógico a memoria no volátil.**

(30) Prioridad: **16.12.2003 US 530184 P**
01.05.2004 US 837250

(45) Fecha de publicación de la mención BOPI:
01.09.2011

(45) Fecha de la publicación del folleto de la patente:
01.09.2011

(73) Titular/es: **MICROSOFT CORPORATION**
One Microsoft Way
Redmond, Washington 90852, US

(72) Inventor/es: **Rogers, Andrew;**
Glaum, Jeffery;
Plagge, Mark;
Tonkelowitz, Mark;
Markley, Michael;
Patel, Sachin y
Shell, Scott

(74) Agente: **Carpintero López, Mario**

DESCRIPCIÓN

Procedimiento y aparato a prueba de fallos para actualizaciones personalizadas de imágenes de soporte lógico a memoria no volátil

Referencia cruzada a solicitudes relacionadas

La presente invención reivindica prioridad con respecto a la solicitud provisional de patente estadounidense con nº de serie 60/530.184, presentada el 16 de diciembre de 2003.

La presente invención está relacionada con las siguientes solicitudes de patente estadounidense, presentadas de forma concurrente con la presente:

Expediente nº 4281/307.650, "Determining the Maximal Set of Dependent Software Updates Valid for Installation", correspondiente al documento US 2005/132 350;

Expediente nº 4291/307.651, "Ensuring that a Software Update may be Installed or Run only on a Specific Device or Class of Devices".

Expediente nº 4301/307.652, "Self-Describing Software Image Update components", correspondiente al documento EP 154 8587; y

Expediente nº 4311/307.663, "Creating File Systems Within a File In a Storage Technology-Abstracted Manner", correspondiente al documento EP-15 44 732.

Campo de la invención

La invención versa en general acerca de dispositivo de cálculo y, más en particular, acerca de la actualización de la memoria no volátil de los dispositivos de cálculo.

Antecedentes

Los dispositivos móviles de cálculo como las agendas electrónicas, los teléfonos móviles contemporáneos y los ordenadores de mano y con tamaño de bolsillo se están convirtiendo en instrumentos de usuario importantes y populares. En general, se han vuelto lo bastante pequeños como para resultar sumamente convenientes, a la vez que consumen menos batería y, a la vez, se han vuelto capaces de ejecutar aplicaciones más potentes.

Durante el proceso de fabricación de tales dispositivos, típicamente se construyen imágenes de un sistema operativo integradas en un fichero monolítico de imagen y almacenadas en una memoria no volátil (por ejemplo, memoria flash NAND o NOR, un disco duro, etcétera) de cada dispositivo. En consecuencia, actualizar tal dispositivo era, hasta ahora, un problema complejo y que consumía muchos recursos, que generalmente requería una solución a medida.

Por ejemplo, actualizar tal dispositivo implica típicamente descargar un fichero monolítico de imagen complemente nuevo que comprende una sola imagen estática desarrollada y hecha pública para ejecutarse en una colección de dispositivos. Como puede apreciarse fácilmente, se requiere una cantidad significativa de recursos del sistema (por ejemplo, memoria para guardar temporalmente la imagen de la actualización, ancho de banda de la red para recibir el fichero completo de imagen, etcétera), con independencia de lo que haya cambiado y, así, el procedimiento de actualización del dispositivo normalmente necesita una solución excepcional a medida.

Se necesita una manera mejor de actualizar la memoria no volátil de los dispositivos de cálculo que sea más flexible, dinámica y eficiente que los mecanismos de actualización actuales y que, pese a todo, sea a prueba de fallos.

El documento EP 1 351 137 A2 da a conocer un cargador de arranque y un procedimiento de arranque con prestaciones de Internet para un dispositivo que comprende una pila IP que soporta protocolos de Internet y un motor de ejecución de secuencias de órdenes para ejecutar una secuencia de órdenes por defecto.

El documento US 6 591 376 B1 da a conocer un procedimiento y un sistema para la recuperación y la actualización a prueba de fallos de un sistema operativo integrado.

El documento EP 0 936 548 A1 da a conocer un procedimiento y un sistema para la pronta recuperación automática de un sistema. Un sistema de ordenador incluye una memoria del sistema, que contiene instrucciones de BIOS, que tiene múltiples particiones susceptibles de arranque y la capacidad de habilitar una protección de Recuperación Automática del Sistema (ASR) durante una fase inicial de procedimiento de arranque.

El documento WO 0152065 A2 da a conocer un procedimiento y un aparato para realizar una copia de seguridad de un código de aplicación tras un fallo durante una actualización de código.

Resumen

El objeto de la presente invención es proporcionar un procedimiento a prueba de fallos y rápido para actualizar particiones y un sistema correspondiente.

Este objeto se solventa por la materia de las reivindicaciones independientes.

Las realizaciones preferentes están definidas por las reivindicaciones dependientes.

Brevemente, la presente invención se dirige hacia un sistema y un procedimiento que aplican actualizaciones del soporte lógico en forma de entidades autocontenidas y seguras a la memoria no volátil integrada de un dispositivo de manera a prueba de fallos. Pueden aplicarse diversos tipos de actualizaciones del soporte lógico, incluyendo actualizaciones que pueden contener únicamente cambios a una actualización previa. Además, las actualizaciones del soporte lógico pueden contener tanto código ejecutable como datos.

En una implementación, tras un rearranque, un cargador inicial de programas determina cuándo se ha solicitado una actualización o si está en curso y, si es así, arranque a un cargador de actualizaciones en vez de al código normal del sistema operativo. El cargador de actualizaciones opera para validar cualquier actualización pendiente para aplicar las actualizaciones a la memoria flash (o a otros medios de memoria) según sea apropiado. Por razones de seguridad y de control, el cargador de actualizaciones es la única entidad del sistema que tiene acceso de escritura a la memoria protegida (por ejemplo, una partición NK y una partición de sistema). Debería hacerse notar que la aplicación de actualización también tiene acceso de escritura a las regiones reservadas de la memoria flash.

Para actualizar, se descargan al sistema paquetes de actualización (que pueden estar en diversas formas) y se validan por medio de un procedimiento de validación de paquetes, que incluye la comprobación, por razones de seguridad, de que cada paquete está debidamente firmado y de que los paquetes, en general, están debidamente construidos. Si son válidas, las actualizaciones son puestas en cola y se pone una bandera para que el cargador inicial de programas detecte que se desea una actualización. Acto seguido, se hace que el dispositivo rearranque.

Tras el rearranque, el cargador inicial de programas ve que está puesta la bandera de actualizaciones y hace que se ejecute el cargador de actualizaciones, por ejemplo descomprimiendo el cargador de actualizaciones (si está comprimido) en la RAM y saltando al código del cargador de actualizaciones. Para cada paquete, empezando con el paquete de núcleo (si existe tal cosa), el cargador de actualizaciones procesa el paquete, lo revalida, corrige la dirección de los ficheros según sea necesario en la sección de la RAM en base a consideraciones de la memoria flash y escribe los ficheros a su correspondiente partición. Obsérvese que el cargador inicial de programas deja la memoria flash sin bloquear para que el cargador de actualizaciones pueda escribir en ella.

Para interpretar el contenido del paquete, cada paquete contiene un fichero de manifiesto de dispositivos que tiene información detallada sobre el paquete, y el cargador de actualizaciones lee el contenido, incluyendo una ID, globalmente única, para la identificación del paquete, una versión del paquete, información de dependencias relativa a otros paquetes, diversas configuraciones y una lista de ficheros y de versiones de ficheros incluidos en el paquete. El fichero de manifiesto está incluido en el paquete, haciendo con ello que el paquete se describa a sí mismo, y se almacena en último término en el dispositivo después de la instalación. Una colección de ficheros de manifiesto en el dispositivo comprende una base de datos del estado de instalación de los paquetes en el dispositivo, que puede ser enumerado.

El cargador de actualizaciones gestiona las imágenes comprimidas y no comprimidas recuperando de la partición o la imagen información diversa en cuanto a la imagen y descomprimiendo la imagen en la RAM. Si no está comprimida, la información, sencillamente, se lee, por ejemplo, mediante el conocimiento previo del lugar en el que se encuentra el registro maestro de arranque, lo que proporciona la ubicación de la partición activa que se está actualizando.

En una implementación, se aplica primero cualquier actualización del núcleo de una manera a prueba de fallos, esencialmente realizando una copia de seguridad del núcleo leyéndolo a la RAM y comprimiéndolo en una memoria de usuario. La actualización es para la partición en su conjunto, y, si tiene éxito, el núcleo objeto de la copia de seguridad es borrado, mientras que si no tiene éxito el núcleo objeto de la copia de seguridad es descomprimido y restaurado.

La partición del sistema es generalmente mayor que la partición del núcleo y, como tal, es típicamente demasiado grande como para realizar una copia de seguridad por razones relacionadas con la seguridad. En vez de ello, se usa un sistema de ficheros IMGFS para aplicar actualizaciones a módulos y ficheros individuales, uno a uno. Los módulos y los ficheros individuales pueden ser demasiado grandes de realizar como un todo y, así, las actualizaciones pueden ser aplicadas por medio de bloques más pequeños.

El IMGFS puede, en primer lugar, simular el procedimiento de actualización, requiriendo dos pases en el procedimiento, concretamente un primer pase en el que el IMGFS y la aplicación de actualizaciones no escribirán nada definitivamente en la memoria flash. Si la simulación tiene éxito, se ejecuta entonces un segundo pase para

escribir realmente los cambios de manera definitiva. Si la simulación falla, puede reintentarse la simulación después de volver a ejecutar el validador de paquetes y de dar paso a una lista que paquetes excluyendo el paquete que falló. La simulación volverá a ejecutarse entonces con la nueva lista que devuelve el validador de paquetes. Se reintentan las simulaciones hasta que o bien la simulación tiene éxito o no hay ningún paquete que pueda ser aplicado. Esto previene la corrupción de los datos que puede haber ocurrido en el paquete que fue firmado pese a la corrupción.

Cuando se iteran los módulos en IMGFS usando el fichero existente de manifiesto de dispositivos, el procedimiento verifica si el fichero aparece en el nuevo fichero de manifiesto de dispositivos para el paquete. Si no, se borra el fichero de la partición IMGFS y el procedimiento continúa con el fichero siguiente. En caso afirmativo, se obtienen del nuevo fichero de manifiesto de dispositivos nuevas banderas para los módulos y se cargan cabeceras de módulos en la estructura de los módulos.

Para llevar a cabo la actualización, el procedimiento de aplicación de actualizaciones entra en un bucle con cada uno de los paquetes en un bucle anidado para procesar cada fichero del paquete individualmente. Si se produce un corte de corriente durante este procedimiento, entonces un fichero de registro dirá exactamente en qué paquete y en qué fichero del paquete se detuvo el procedimiento, lugar en el que puede reanudarse la actualización cuando vuelva la corriente. Parte del procesamiento anterior incluye (para el código ejecutable) actualizar la asignación de direcciones virtuales para los módulos de actualización.

Las actualizaciones pueden ser de forma canónica o de forma de diferencia binaria (delta). Los ficheros que no son módulos no requieren correcciones de direcciones virtuales y pueden ser escritos una vez que se lleve a cabo la actualización binaria. Si el fichero es un módulo, entonces cada sección es procesada una por una; cada sección es escrita a un flujo nuevo (siendo “flujo” un concepto del sistema de ficheros que forma parte del IMGFS). Si el fichero es un fichero regular de datos, los datos se escriben en el flujo por defecto.

Si el módulo es una actualización canónica, se procesa cada una de las secciones del módulo, incluyendo la reubicación de una sección completa a la dirección asignada por el asignador virtual y la escritura de toda la sección como un nuevo flujo en el nuevo fichero del IMGFS. El módulo antiguo es actualizado *in situ*, desasignándose los bloques antiguos de memoria a medida que se escriben los bloques nuevos. En el caso canónico, la recuperación de un corte de corriente se lleva a cabo obteniendo el tamaño de cada flujo del nuevo fichero que se ha escrito y comparándolo con el tamaño de esa sección en el fichero canónico. Si el flujo del nuevo fichero es menor que el fichero canónico, entonces ese es el lugar en el que el procedimiento se detuvo y puede reanudarse la copia encima de los datos.

En el caso de las diferencias binarias, se lleva a cabo un procedimiento de parcheado bloque por bloque, en el que un bloque puede ser igual al tamaño de la página o el sector. Esto se debe a que puede no haber suficiente espacio en la memoria flash para mantener juntas tanto la versión antigua como la versión nueva de una sección. A medida que se crean los nuevos bloques, los bloques antiguos pueden ser desasignados, puesto que ya no son necesarios. Para una actualización de diferencias binarias, se crea un flujo poco denso en el fichero del nuevo módulo para esta sección y la sección antigua se carga en RAM. Toda la sección antigua se vuelve a reubicar en la dirección base original. Para cada nuevo bloque del fichero de diferencias binarias, se construye un nuevo bloque basado en las instrucciones de las diferencias binarias y el nuevo bloque es reubicado en la dirección asignada por el asignador de direcciones virtuales y es escrito al flujo poco denso. Cualquier bloque antiguo del flujo del fichero antiguo que ya no se necesite es desasignado.

En el caso de las diferencias binarias, un corte de corriente se gestiona acotando qué sección se interrumpió comparando el tamaño del flujo con el especificado en la cabecera. Una vez que se determina la sección, el parche vuelve al bloque siguiente en la sucesión y se realiza una comprobación en cuanto a si el bloque ha sido escrito definitivamente. Si el bloque ya ha sido escrito definitivamente, se lleva a cabo una verificación para comprobar que los bloques antiguos apropiados ya han sido liberados. Una vez que se encuentra el bloque que se interrumpió (es decir, aquel que aún no ha sido escrito definitivamente), el procedimiento prosigue de la manera normal.

Para actualizar secciones reservadas de la memoria, se logra una actualización de secciones reservadas (por ejemplo, fuera de una partición) de una manera similar que la de las actualizaciones de la partición del núcleo, usando los mismos componentes de actualización. Se procesa cada actualización de paquetes relacionados con la reserva procesando cada fichero del paquete y determinando si el fichero es un módulo de diferencias binarias. Si no, los datos del fichero son, sencillamente, leídos y escritos desde el fichero a la región reservada. Si se trata de un módulo de diferencias binarias, entonces se lee la región existente (por ejemplo, introduciéndola en la RAM) y se aplica la diferencia binaria a la misma antes de volver a escribir los datos actualizados a la región reservada.

Breve descripción de los dibujos

Otras ventajas se harán evidentes a partir de la siguiente descripción detallada cuando se toma en conjunción con los dibujos, en los cuales:

la FIGURA 1 es un diagrama de bloques que representa en general un sistema de ordenador en el cual puede incorporarse la presente invención;

la FIGURA 2 es un diagrama de bloques que representa una imagen dividida de un sistema operativo y componentes del cargador para facilitar actualizaciones a prueba de fallos de los componentes según un aspecto de la presente invención;

la FIGURA 3 es un diagrama de bloques que representa la aplicación de un paquete de actualizaciones a una memoria de un dispositivo según un aspecto de la presente invención;

la FIGURA 4 es un diagrama de bloques que representa el formato de un fichero de manifiesto de dispositivos que describe un paquete según un aspecto de la presente invención;

las FIGURAS 5A y 5B comprenden un diagrama de flujo que representa lógica ejemplar para un procedimiento de arranque cargador inicial de programas que determina un modo de actualización según un aspecto de la presente invención;

la FIGURA 6 es un diagrama de bloques que representa una lógica ejemplar para un procedimiento global de actualización según un aspecto de la presente invención;

las FIGURAS 7A y 7B comprenden un diagrama de flujo que representa lógica ejemplar para la actualización de una partición del núcleo según un aspecto de la presente invención;

las FIGURAS 8A y 8B comprenden un diagrama de flujo que representa lógica ejemplar para la actualización de una partición del sistema operativo según un aspecto de la presente invención;

la FIGURA 9 es un diagrama de bloques que representa la aplicación de una actualización en bloques a un dispositivo por medio de un gráfico de dependencias según un aspecto de la presente invención;

la FIGURA 10 es un diagrama de bloques que representa lógica ejemplar para actualizar una sección reservada de memoria según un aspecto de la presente invención.

Descripción detallada

Entorno operativo ejemplar

La FIG. 1 muestra componentes funcionales de un dispositivo 120 de mano de cálculo de este tipo, que incluye un procesador 122, una memoria 124, una pantalla 126 y un teclado 128 (que puede ser un teclado físico o uno virtual, o puede representar ambos). Puede haber presente un micrófono 129 para recibir una entrada de audio. La memoria 124 incluye generalmente tanto memoria volátil (por ejemplo, RAM) como memoria no volátil (por ejemplo, ROM, tarjetas PCMCIA, etcétera). Un sistema operativo 130, como el sistema operativo Windows®, de Microsoft Corporation, u otro sistema operativo, está residente en la memoria 124 y se ejecuta en el procesador 122.

Uno o más programas 132 de aplicación están cargados en la memoria 124 y se ejecutan en el sistema operativo 130. Ejemplos de aplicaciones incluyen programas, programas de planificación, programas de PIM (gestión de la información personal), programas de tratamiento de textos, programas de hoja de cálculo, programas exploradores de Internet, etcétera. El ordenador personal 120 de mano puede también incluir un gestor 134 de notificaciones cargado en la memoria 124 que se ejecuta en el procesador 122. El gestor 134 de notificaciones gestiona solicitudes de notificaciones, por ejemplo, de los programas 132 de aplicación. Además, tal como se describe más abajo, el ordenador personal 120 de mano incluye un soporte lógico 136 de red (por ejemplo, controladores de soporte físico y similares) y componentes 138 de red (por ejemplo, una radio y una antena) adecuados para conectar el ordenador personal 120 de mano a una red, lo que puede incluir la realización de una llamada telefónica.

El ordenador personal 120 de mano tiene un suministro eléctrico 140, que está implementado como una o más baterías. El suministro eléctrico 140 puede incluir, además, una fuente externa de corriente, como un adaptador de CA o una base de acoplamiento con alimentación eléctrica, que puentea o recarga las baterías incorporadas.

El ordenador personal ejemplar 120 de mano representado en la FIG. 1 se muestra con tres tipos de mecanismos externos de notificación: uno o más diodos 142 emisores de luz (LED) y un generador 144 de audio. Estos dispositivos pueden estar directamente acoplados al suministro eléctrico 140 para que, cuando sean activados, permanezcan encendidos una duración dictada por un mecanismo de notificación aunque el procesador 122 del ordenador personal de mano y otros componentes pudieran apagarse para conservar la energía de las baterías. Preferentemente, el LED 142 sigue encendido indefinidamente hasta que el usuario haga algo. Obsérvese que las versiones contemporáneas del generador 144 de audio usan demasiada energía para las baterías actuales de los ordenadores personales de mano, de modo que está configurado para apagarse cuando lo haga el resto del sistema o tras cierta duración finita después de su activación.

Obsérvese que, aunque se ha mostrado un ordenador personal básico de mano, casi cualquier dispositivo capaz de recibir comunicaciones de datos y de procesar los datos de alguna manera para su uso por un programa, como un teléfono móvil, es equivalente para los fines de la implementación de la presente invención.

Aplicación a prueba de fallos de actualizaciones de imágenes del soporte lógico a la medida

La presente invención está dirigida en general hacia la instalación y/o la actualización de un soporte lógico que se almacena en pequeños dispositivos móviles de cálculo, como los dispositivos portátiles basados en Microsoft Windows® CE .NET, incluyendo aquellos en los que el soporte lógico inicial o la actualización del soporte lógico se escriben en la memoria no volátil del dispositivo, es decir, a la memoria flash. No obstante, la presente invención proporciona beneficios a la computación en general y, así, puede aplicarse a otros dispositivos de cálculo y a otros tipos de memoria, incluyendo diversos tipos de memoria y/o a otros tipos de memoria, como unidades de disco duro. En aras de la sencillez, el término “flash” se usará en este documento en lo sucesivo con referencia a la memoria actualizable de un dispositivo, aunque se entiende que cualquier mecanismo de memoria es equivalente. Además, el término “imagen” incluirá generalmente el concepto de la imagen de la instalación inicial del soporte lógico, así como a las subsiguientes actualizaciones del soporte lógico a una imagen, incluso cuando se actualice solamente parte de una imagen existente.

Según un aspecto de la presente invención, se aplican de manera a prueba de fallos actualizaciones del soporte lógico en forma de entidades seguras autocontenidas a una memoria no volátil integrada del dispositivo. Pueden aplicarse diversos tipos de actualizaciones del soporte lógico, incluyendo actualizaciones que pueden contener únicamente los cambios a una actualización previa. Además, las actualizaciones del soporte lógico pueden contener tanto código ejecutable como datos. Según se entenderá, el código ejecutable está hecho a medida del entorno del espacio de direcciones virtuales del dispositivo integrado en el momento de la instalación. En consonancia con la presente invención, las actualizaciones del soporte lógico se instalan de manera a prueba de fallos y permiten las opciones de recuperación tanto en avance como en retroceso, dependiendo de la fase de la actualización.

Las siguientes tablas proporcionan definiciones generales no limitantes para algunos de los términos y los tipos de ficheros descritos en el presente documento:

Términos

Término	Definición general
Módulo	Un único fichero ejecutable (.EXE, .DLL, etc.)
Configuraciones	Una colección de información de configuración que puede contener configuraciones del Registro, instrucciones de inicialización del sistema de ficheros (fichero .DAT), inicializaciones de las bases de datos y XML de aprovisionamiento
Componente	Una colección de módulos, ficheros (ficheros que no son módulos, incluyendo plantillas, fuentes, etc.) y configuraciones que componen una unidad de características. Típicamente, los componentes están asociados con identificadores *MODULES* del generador del sistema
Paquete	Una colección de componentes que está firmada y empaquetada para su distribución
Manifiesto	Un fichero que describe el contenido de un paquete. En el entorno de construcción, hay un fichero de manifiesto del paquete con la extensión .BIB que contiene entradas que describen los nombres de cada fichero del paquete, pero eso es todo. El manifiesto del lado del dispositivo es un fichero binario que describe toda la información sobre un paquete (véase, más abajo, Manifiesto del lado del dispositivo)
Herramienta de sombra de orden	Herramienta de construcción que procesa el fichero de relaciones de componentes y genera los ficheros sombra del paquete

Tipos de fichero

Extensión	Tipo de fichero	Descripción general
.pkd.xml	Fichero de definición del paquete	Fichero XML en el árbol de construcción que define qué paquetes existen (nombre del paquete, GUID, versión, cálculo de líneas de código)
.cpm.csv	Fichero de correspondencias de componente a paquete	Fichero CSV en el árbol de construcción que establece una correspondencia de los identificadores MODULES y de los ficheros con los paquetes

Extensión	Tipo de fichero	Descripción general
.crf.csv	Fichero de relaciones de componentes	Fichero de texto en el árbol de construcción que define las relaciones (sombra y dependencia) entre identificadores MODULES
.psf	Fichero sombra del paquete	Fichero intermedio en el entorno de construcción (uno por paquete, como en <nombre_de_paquete>.psf) que enumera todos los paquetes que deben ser copiados como sombras por el paquete nombrado
.dsm	Fichero de manifiesto del lado del dispositivo	Fichero en el dispositivo (uno por paquete) que describe el paquete (ficheros del paquete, nombre, GUID, firma, versión, sombras, dependencias, CRC, certificados raíz, etc.)
.pkg.cab	Fichero de paquete canónico	Versión completa de un paquete que contiene el fichero completo para todos los ficheros del paquete
.pku.cab	Fichero de actualización del paquete	Un fichero que facilita la actualización de un dispositivo partiendo de una versión específica de un solo paquete a una versión diferente del mismo paquete. Este fichero puede contener diferencias binarias de ficheros individuales o ficheros completos, cualquiera de las dos cosas que minimice el tamaño del paquete de actualización
.pks.cab	Superpaquete de ficheros	Un fichero que contiene una colección de paquetes de actualización y/o de paquetes canónicos

Tal como también se describe en la solicitud de patente estadounidense relacionada titulada “Creating File Systems Within a File In a Storage Technology-Abstracted Manner”, que ha sido mencionada anteriormente, una imagen inicial puede ser construida a partir de partes componentes (incluyendo paquetes, descritos más abajo) y ser instalada en un dispositivo de una forma a la medida durante el proceso de fabricación. La imagen es construida con particiones separadas, representadas de forma general en la FIG. 2, y puede ser actualizada por partes, como se describe más abajo, en vez de requerir el remplazo de la imagen monolítica.

El modelo de partición facilita la actualización fiable del soporte lógico en futuros dispositivos integrados, así como en aquellos que ya existen. En vez de generar una única imagen estática durante el procedimiento de desarrollo y publicación del soporte lógico que está concebida para ejecutarse en una colección de dispositivos, la presente invención proporciona un mecanismo más dinámico de actualización de imágenes que facilita una mejor componentización de la imagen actualizable, a la vez de ser a prueba de fallos. Con este fin, la presente invención proporciona un mecanismo más flexible para segmentar una imagen del sistema operativo en porciones separadas actualizables que pueden ser actualizadas en aislamiento, mientras que mantiene cualquier dependencia entre componentes. Para dar soporte a este concepto, la imagen del sistema operativo inicial es construida con ciertas características arquitectónicas clave.

La FIG. 2 muestra un modelo de partición ejemplar para una imagen 202 del sistema operativo instalada en medios flash y/o en otros medios adecuados de memoria no volátil. Se proporciona una partición 204 del núcleo como partición restringida/protegida con el fin de almacenar una imagen del núcleo. La partición del núcleo/NK proporciona memoria para los componentes nucleares de la imagen del sistema operativo (núcleo, sistema de ficheros, etcétera), y contiene código que debe ejecutarse durante el procedimiento de arranque. Se proporciona una partición 206 restringida/protegida del sistema para almacenar la aplicación de componentes del sistema y otras partes de la imagen del sistema operativo (controladores, aplicaciones, ficheros de datos, etcétera). El contenido de estas particiones 204 y 206 es gestionado por los controladores del sistema de ficheros, que abstraen la correspondencia física de la memoria y también sirven para soportar las actualizaciones de componentes sujetas a transacción en la partición 206 del sistema, permitiendo así una solución de recuperación a prueba de fallos en consonancia con la presente invención, tal como se describe más abajo. El acceso al contenido de la partición protegida del núcleo es controlado por medio de un sistema de ficheros binarios (BINFS), mientras que el acceso al contenido de la partición protegida del sistema es controlado por medio de un sistema de ficheros de imagen (IMGFS).

En la FIG. 2 también se representa (aunque no sea realmente parte de la imagen 202 del sistema operativo) una partición 210 de memoria de usuario, que el sistema/usuario puede usar según se necesite, y que esencialmente puede ser cualquier formato de sistema de ficheros (por ejemplo, TFAT). Tal como se describe más abajo, entre otros usos potenciales, esta partición puede usarse para almacenar temporalmente algunos de los paquetes que han de ser instalados durante un procedimiento de actualización. Obsérvese que se escribe un registro maestro de arranque para definir las particiones; por ejemplo, sus desplazamientos/tamaños.

La FIG. 2 también proporciona una vista funcional del entorno cargador. En general, el cargador es responsable de la carga inicial del sistema integrado hasta el punto en que debe tomarse una decisión (por parte de un cargador 222

inicial de programas, que puede actuar conjuntamente con un cargador preexistente 224 de arranque, si está presente) en cuanto a si ocurrirá un arranque normal del sistema operativo o un arranque de actualización. Obsérvese que un cargador "preexistente" de arranque no es necesario para los dispositivos nuevos; sin embargo, un fabricante de dispositivos puede opcionalmente decidir que debería estar presente.

Para los arranques de actualización, parte del cargador comprende un cargador de actualizaciones (el UL 220), que sirve de portero de cualquier actualización de la memoria flash. El cargador 220 de actualizaciones es responsable de validar cualquier actualización pendiente y, luego, una vez validadas, de aplicar las actualizaciones a la memoria flash, según sea apropiado. Por razones de seguridad y control, el cargador 220 de actualizaciones es la única entidad del sistema que tiene acceso de escritura a la memoria protegida (por ejemplo, la partición NK y la partición del sistema) y, así, proporciona un único punto responsable de proteger la integridad de la memoria del sistema. Obsérvese que el cargador de actualizaciones es relativamente grande y que se usa únicamente cuando se aplican actualizaciones y, así, en una implementación, se almacena de forma comprimida y se descomprime a la RAM cuando es necesario.

La FIG. 3 representa un mecanismo/procedimiento de instalación de actualizaciones. Obsérvese que la FIG. 3 se inicia desde la perspectiva de una actualización después de que el dispositivo haya sido poblado inicialmente (por ejemplo, a través de una interfaz JTAG o de un programador múltiple durante la fabricación) con los componentes del cargador inicial de programas, el registro maestro de arranque (MBR) y el cargador de actualizaciones (UL). También se aplican los paquetes canónicos (descritos más abajo) desde la memoria o la RAM persistentes a las particiones del núcleo y el sistema.

En la FIG. 3, los paquetes 302 de actualizaciones (que pueden ser paquetes canónicos, de actualizaciones delta o superpaquetes, según se describe más abajo) se descargan a la RAM 304 del sistema y/o a la memoria 210 de datos de usuario, posiblemente de forma manual o automática, dándose una notificación al usuario del dispositivo de que hay actualizaciones disponibles. Esto se representa en general en la FIG. 3 mediante la flecha etiquetada con el número uno (1) en un círculo.

Así, una vez que la imagen inicial de fábrica está instalada en un dispositivo, las actualizaciones futuras de la imagen se realizan actualizando partes diferenciadas de la imagen que están encapsuladas en paquetes. En general, un paquete es una colección de ficheros de imagen (código, datos, secuencias de órdenes, etcétera) que se describe a sí misma, y en una implementación comprende una colección de componentes que está firmada y empaquetada para su distribución. Los paquetes proporcionan un medio para que los ficheros del soporte lógico de la imagen del sistema operativo se segmenten en agrupaciones funcionales menores de ficheros. En una implementación, la imagen del sistema operativo complemente comprende uno o más paquetes (excluyendo los componentes del cargador) y puede ser tratada simplemente como una suma de paquete menores que son gestionados, construidos y actualizados individualmente, cada uno de los cuales puede ser actualizado individualmente o en combinación con otros paquetes, dependiendo de los requisitos de cada paquete. Así, en consonancia con la presente invención, las actualizaciones de imágenes ya no tienen por qué hacerse como una región de una imagen de un sistema operativo completo.

Los paquetes pueden ser configurados de diversas maneras, incluyendo las formas "canónica", "delta/diferencia" y "súper", cada una de las cuales sirve diversos propósitos con respecto a las actualizaciones del soporte lógico. Por ejemplo, los paquetes canónicos contienen una copia completa de cada fichero dentro del paquete, mientras que los paquetes delta o de diferencias son típicamente de menor tamaño con respecto a otros paquetes y, así, se usan cuando se intenta optimizar el costo de la descarga y el tiempo de la descarga. Los superpaquetes contienen otros paquetes y se usan como conveniencia cuando se precisa descargar más de un paquete (por ejemplo, como con paquetes interdependientes). En una implementación, los ficheros del paquete son ficheros CAB de Win32 que contienen un fichero de manifiesto (descrito más abajo) y los ficheros que definen el paquete; por ejemplo, los que deben ser instalados.

Los paquetes canónicos se generan durante el proceso de construcción asociando características del sistema operativo y metadatos (por ejemplo, un código ejecutable de una aplicación específica y todos los datos asociados y la información de configuración) con una definición del paquete. Los paquetes delta o de diferencias se generan a partir de los paquetes canónicos aplicando un algoritmo de diferencias binarias al contenido de dos paquetes canónicos y capturando la relación de dependencia que el paquete delta o de diferencias tiene en cuando a la versión de referencia del paquete canónico. Una vez que se establece una correspondencia de las características y los metadatos de la imagen del sistema operativo con los paquetes individuales, es creado cada paquete, como se describe en general en la solicitud de patente estadounidense, "Self-Describing Software Image Update Components", mencionada anteriormente, usando herramientas de empaquetado para enumerar el contenido del paquete y procesando ficheros ejecutables (con una herramienta de reubicación denominada RelMerge) para permitir que sean actualizados en el futuro para las correcciones de dirección relacionadas con la memoria.

Cuando el usuario inicia un procedimiento de actualización, un validador de paquetes verifica el paquete, según se describe en las solicitudes de patente relacionadas, tituladas "Determining the Maximal Set of Dependent Software Updates Valid for Installation" y "Ensuring that a Software Update may only be Installed or Run on a Specific Device

or Class of Devices", mencionadas anteriormente. Si se valida, el sistema pone una bandera de actualización en la memoria flash y rearranca. Obsérvese que, según se describe más abajo, la bandera de actualización es persistente, de modo que, una vez que se inicia el código del cargador de actualizaciones, el dispositivo volverá a entrar en el modo de actualización por si se corta el suministro eléctrico.

Tras el rearranque, el cargador inicial 222 de programas ve que está puesta la bandera de actualización y descomprime el cargador comprimido de actualizaciones a la RAM 304, según se representa en la FIG. 3 por medio del código 308 no comprimido del cargador de actualizaciones, y pasa el control al código 308 del cargador de actualizaciones. Esto se representa en general en la FIG. 3 por medio de la flecha etiquetada con un número dos (2) en un círculo. Obsérvese que, cuando el cargador de actualizaciones lanza la aplicación de actualizaciones, la aplicación de actualizaciones vuelve a ejecutar el procedimiento de validación usando una lista de ficheros, o un conjunto de especificaciones de directorio que fueron establecidas por la aplicación updatebin.exe antes de que el dispositivo volviese a arrancar. El procedimiento de validación vuelve a ejecutarse para prevenir el remplazo malicioso de paquetes.

Comenzando con el paquete del núcleo, si lo hay, para cada paquete, el cargador de actualizaciones procesa el paquete, corrige la dirección de los ficheros según se precise en una sección 310 de RAM en base a consideraciones de la memoria flash y escribe los ficheros a su correspondiente partición usando servicios del BINFS, según se muestra en general en la FIG. 3 por medio de las flechas etiquetadas tres (3) a seis (6). Las correcciones de dirección son descritas en general más abajo, y también en la solicitud de patente relacionada, titulada "Self-Describing Software Image Update Components", mencionada anteriormente.

Cada paquete contiene un fichero 400 de manifiesto del dispositivo que contiene información detallada sobre el paquete, como se representa en general en la FIG. 4. Los ficheros de manifiesto son esencialmente ficheros de especificaciones para cada paquete, y contienen información tal como una ID globalmente única para la identificación del paquete, una versión del paquete, una información de dependencia con respecto a otros paquetes, configuraciones, una lista de ficheros y de versiones de ficheros incluidos en el paquete y otras características comunes compartidas. El fichero 400 de manifiesto se incluye en el paquete, haciendo con ello que el paquete se describa a sí mismo, es repasado durante el procedimiento de instalación y es almacenado en último término en el dispositivo. Una colección de ficheros de manifiesto en el dispositivo comprende una base de datos del estado de la instalación de paquetes en el dispositivo, que puede ser enumerada.

En una implementación correspondiente al formato y la información contenida en el fichero de manifiesto del dispositivo mostrado en la FIG. 4, el fichero de manifiesto del dispositivo es descrito también con esta definición de estructura:

```
typedef struct _DeviceManifestHeader

{

    const DWORD dwStructSize;          // Tamaño de esta estructura (en bytes) para el control de versiones

    const DWORD dwPackageVersion;      // Versión de este paquete

    const DWORD dwPrevPkgVersion;      // Versión del paquete que actualiza este paquete. (0) para canónica

    const DWORD dwPackageFlags;        // Identificadores específicos del paquete

    const DWORD dwProcessorID;         // Qué procesador (coincide con definiciones de winnt.h)

    const DWORD dwOSVersion;           // Para qué versión del sistema operativo se construyó esto

    const DWORD dwPlatformID;          // Cuál fue la plataforma objetivo

    const DWORD dwNameLength;          // Longitud del nombre de fichero en bytes

    const DWORD dwNameOffset;          // Desplazamiento hasta el nombre amigable del paquete

    const DWORD dwDependentCount;      // Cuántas entradas hay en la lista de GUID dependiente

    const DWORD dwDependentOffset;     // A cuántos bytes del comienzo del fichero están las estructuras del GUID

    const DWORD dwShadowCount;         // Cuántas entradas hay en la lista de sombra GUID

    const DWORD dwShadowOffset;        // A cuántos bytes del comienzo del fichero está la matriz de GUID de sombra del

                                       // paquete
```

```
    const DWORD dwFileCount;          // Cuántos ficheros se enumeran en este manifiesto
    const DWORD dwFileListOffset;     // A cuántos bytes del comienzo del fichero está la primera FileEntry
    const DWORD cbCERTData;           // Número de bytes de los datos de certificado digital
    const DWORD dwCERTDataOffset;     // A cuántos bytes del comienzo del fichero están los datos del certificado
    const GUID guidPackage;           // GUID de este paquete
} DeviceManifestHeader, *PDeviceManifestHeader;

typedef struct _DependentEntry {
    const DWORD size;
    const DWORD version;
    const GUID guid;
} DependentEntry, *PDependentEntry;

typedef struct _FileEntry {
    const DWORD dwNameLength;
    const DWORD dwFlags;
    const DWORD dWOffset;
    const DWORD dwBase;               // Dirección base con la que este fichero estaba ligada en su origen
} FILEENTRY, *PFILEENTRY;
```

Cada paquete que se crea es firmado por razones de seguridad y entonces está lista para su descarga/instalación. El procedimiento de generación del paquete o el generador de paquetes, tal como se describe en la solicitud de patente estadounidense relacionada, titulada “Self-Describing Software Image Update Components”, mencionada anteriormente, automatiza la creación de ficheros del paquete que contienen versiones completas de los ficheros que están definidos por los ficheros de descripción del paquete. En general, el generador de paquetes da entrada a un fichero de definición del paquete (por ejemplo, especificado por un argumento de la línea de órdenes); otros ficheros de entrada son determinados en base al contenido del fichero especificado de definiciones del paquete. El generador de paquetes analiza el fichero de definición del paquete (que comprende, por ejemplo, una o más definición del paquete, estando codificado cada paquete como una entrada de un documento XML) y puede analizar un fichero sombra del paquete que contiene datos relacionados con la ordenación de precedencia con respecto a otros paquetes para configuraciones como las configuraciones del registro. Además, el generador de paquetes analiza un fichero de manifiesto de la construcción para encontrar cada fichero especificado en el mismo y determinar después el tipo del fichero. Para tipos de ficheros ejecutables, el generador de paquetes invoca un proceso (RelMerge) que lleva a cabo operaciones relacionadas con la reubicación, al igual que lleva a cabo un formateado (por ejemplo, para convertir un ejecutable a un formato de fichero de Windows® CE).

En una implementación, el generador de paquetes comprende un programa de aplicación que usa bibliotecas de clases .NET denominadas, por ejemplo, PkgCommon. La aplicación generadora de paquetes crea múltiples subdirectorios, siendo el número de subdirectorios que se crean igual al número de paquetes válidos encontrados en el fichero de la colección de paquetes especificado en la línea de órdenes. Se crea para cada paquete un subdirectorio único para él durante el procedimiento de generación del paquete. A demás, el generador de paquetes creará ficheros adicionales en cada subdirectorio usando información encontrada dentro de una definición del paquete. Para cada paquete, se crea un Fichero de Manifiesto del Dispositivo, derivándose el nombre del Fichero de Manifiesto del Dispositivo del GUID del paquete. Para cada definición válida de paquete, se crea un fichero de paquete para contener los ficheros que comprenden el paquete, incluyendo el fichero de manifiesto del dispositivo. Estos ficheros de paquetes se atienen al formato de la versión 1.3 de los ficheros CAB de Microsoft.

El procedimiento de instalación hace un uso intenso del contenido del fichero de manifiesto del dispositivo, tanto para los paquetes ya instalados en el dispositivo como para los paquetes puestos en cola para su instalación

potencial en el dispositivo. La API de información de paquetes fue diseñada para proporcionar un medio abstraído de interrogar la información de los paquetes y se usa tanto en el dispositivo como en el ordenador central de la construcción. La API se detalla en la solicitud de patente relacionada, titulada “Determining the Maximal Set of Dependent Software Updates Valid for Installation”, mencionada anteriormente.

Las actualizaciones a las particiones del núcleo o del sistema son gestionadas y suministradas al dispositivo usando el mecanismo de paquetes. Los paquetes son transportados al dispositivo, guardados en la memoria temporal y puestos en una cola para su instalación mediante cualquier medio adecuado; por ejemplo, mediante componentes de la imagen existente del sistema operativo. Puede usarse cualquier medio de transporte adecuado (físico y/o protocolo de interfaz) para suministrar los paquetes al dispositivo, y variará dependiendo del dispositivo que se está actualizando (por ejemplo, inalámbrico para los teléfonos inteligentes, USB para pantallas inteligentes, por medio de alguna conexión física con alguna aplicación de actualización de sobremesa, etcétera. La experiencia de la interfaz final del usuario variará dependiendo del tipo de dispositivo que se está actualizando, y el procedimiento de actualización según personalice el fabricante del dispositivo. En consonancia con los aspectos de seguridad de la presente invención, cada paquete temporal se guarda en memoria persistente para que, en caso de un corte de corriente, el sistema sea capaz de recuperarse y de proseguir con la instalación.

El procedimiento de actualización se desencadena cuando se invoca la biblioteca del validador de paquetes. En una implementación, un fichero ejecutable denominado UpdateBin.exe está asociado con las extensiones de fichero .pkg.cab, .pku.cab y .pks.cab. Este ejecutable hace uso de la biblioteca del validador de paquetes para determinar el conjunto de paquetes que usar para actualizar el dispositivo.

El validador de paquetes comprueba la(s) firma(s) de los paquetes, valida el contenido de los paquetes, verifica las versiones, etcétera. Una vez que los paquetes de actualización son considerados apropiados, se ponen en una cola para su instalación y el validador pone una bandera de actualización para señalar al dispositivo que hay disponible una actualización y luego rearranca el sistema.

En el rearranque, el cargador inicial 222 de programas es el primer trozo de código que se ejecuta desde el vector de puesta a cero de la CPU (aunque, en algunos dispositivos, el cargador inicial de programas puede ser arrancado por una imagen existente de un cargador de arranque). El cargador inicial 222 de programas es responsable de determinar si el dispositivo está en modo de actualización o si el dispositivo está en un modo normal de arranque. Si el dispositivo está en un modo normal de arranque, el cargador inicial de programas localiza y arranca la imagen del sistema operativo; si no, el cargador inicial de programas localiza, descomprime y arranca la imagen descomprimida 308 del cargador de actualizaciones. Las FIGURAS 5A y 5B describen el procedimiento inicial de arranque del cargador de programas. Según se muestra en la FIG. 5A, tras cierta inicialización en la etapa 502 (por ejemplo, para inicializar la CPU, si se utiliza NAND para inicializar el controlador, inicializar la SDRAM, etcétera), el cargador inicial de programas determina por medio de las etapas 504 y 506 la razón del rearranque. Más en particular, algunas arquitecturas de CPU imitan la secuencia de rearranque cuando reanudan su funcionamiento partiendo de un estado suspendido. Típicamente, el código de arranque diferencia entonces una reanudación de un arranque “normal” en base a un valor de un registro de la CPU. El cargador inicial de programas tiene en cuenta la operación de reanudación, y las etapas 504-508 se muestran en aras de la exhaustividad.

Si no hay reactivación, la etapa 506 se ramifica en las etapas 510-528, en las que, como puede verse, se detecta el modo de actualización y se produce una exploración en la partición del cargador de actualizaciones en busca de una actualización (etapa 520); si no, se explora la partición del sistema operativo en la etapa 518. Como puede verse en la etapa 522, en el caso de que el cargador inicial de programas no pueda localizar una imagen del sistema operativo en el dispositivo, el cargador inicial de programas intentará cargar y ejecutar el cargador de actualizaciones. Si no se encuentra una partición apropiada, se emite un error (etapa 532); obsérvese que, en esta ocasión, el dispositivo no funciona con controladores de visualización y similares normales y que, por ello, cualesquiera imágenes visualizadas en este momento son mapas de bits procedentes de la memoria de datos.

Además, cuando se arranca al sistema operativo, la memoria flash está bloqueada, por medio de la etapa 524, etapa que no se ejecuta arrancando al cargador de actualizaciones. El procedimiento prosigue entonces a la etapa 538 de la FIG. 5B para gestionar imágenes comprimidas en vez de no comprimidas. Obsérvese que la FIG. 5B es o bien para el cargador de actualizaciones o para el sistema operativo. El cargador de actualizaciones se almacena (en los dispositivos típicos) en forma comprimida, mientras que el sistema operativo puede estar comprimido o no, dependiendo de si el dispositivo es un dispositivo que permita que el código sea ejecutado *in situ* (si lo es, no puede ser comprimido).

Si está comprimido, se recupera de la partición/imagen la información diversa en cuanto a la imagen y se descomprime la imagen en la RAM y se inicia la ejecución por medio de las etapas 540-546. Obsérvese que puede usarse cualquier algoritmo de compresión adecuado, con la condición de que el tipo de compresión y el formato de los datos comprimidos sean conocidos por el cargador inicial de programas. Si no está comprimido, la etapa 538 lee la información en cuanto a la imagen del sector de la partición en la etapa 544, localiza en ese sector la tabla de contenido (TOC), el desplazamiento y la firma. Con este fin, el cargador inicial de programas tiene conocimiento previo del lugar en el que vive el registro maestro de arranque (el comienzo del siguiente bloque bueno de la

memoria flash después de la imagen del cargador inicial de programas) y localiza la partición activa en la tabla de particiones que se almacena en el registro maestro de arranque. Así, el cargador inicial de programas depende de que la imagen de la ROM o la imagen del disco tengan la tabla de contenido (TOC) debidamente ubicada, tanto en la región de ejecución *in situ* de la imagen del sistema operativo como en la imagen del cargador de actualizaciones. En una implementación, se proporciona lo siguiente como ubicación:

En el desplazamiento en bytes (Inicio de la imagen + 0x40):

UINT32 ROMSignature;

UINT32 *pTOC;

Esta información es usada por la imagen del sistema operativo para localizar la TOC (tal como ha sido colocada por la imagen de la ROM o la imagen del disco). El punto solo tiene sentido cuando se conecta la unidad de gestión de la memoria (MMU) y las correspondencias son compatibles con las direcciones para las que fue construida la imagen del sistema operativo. Una alternativa es implementar lo siguiente:

En el desplazamiento en bytes (Inicio de la imagen + 0x40):

UINT32 ROMSignature;

UINT32 *pTOC;

UINT32 TOCOffset;

siendo TOCOffset un desplazamiento en bytes desde la ubicación del inicio de la imagen hasta la TOC y pudiendo ser usado por el cargador inicial de programas (o la aplicación de carga de actualizaciones) para localizar la TOC sin saber el lugar para el cual la imagen está construida para ejecutarse.

El cargador inicial de programas también puede verificar la firma, según representa la etapa 548. En la etapa 548, si es inválida, el procedimiento de arranque se detiene en la etapa 550; si no, la tabla de contenido es leída por medio de las etapas 552 y 554 para encontrar el comienzo de la imagen, la longitud de la imagen y la dirección de salto desde la tabla de contenido. La etapa 558 corrige las direcciones según sea necesario en base a una tabla de datos suministrada por un fabricante. Más en particular, el cargador inicial de programas se ejecutará a menudo en el espacio de direcciones físicas del dispositivo (o, a veces, en un espacio de direcciones virtuales que puede ser diferente del de la imagen del sistema operativo o de la imagen del cargador de actualizaciones). Esta rutina es responsable de la traducción de cualquier dirección virtual específica de la imagen (por ejemplo, los valores de la TOC de la imagen del sistema operativo) a direcciones compatibles con el cargador inicial de programas. Si la rutina de arranque del fabricante no permite la unidad de gestión de memoria, ello significa traducir la dirección virtual a una física. Si se permite la unidad de gestión de memoria y las correspondencias son compatibles con la forma en que están construidos el sistema operativo o la imagen del cargador de actualizaciones, entonces no se requiere traducción alguna.

La etapa 558 verifica si la imagen precisa ser cargada en la RAM (lo cual, en caso afirmativo, ocurriría en la etapa 562) o puede ser ejecutada *in situ*. Se salta entonces a la imagen (cargada o *in situ*) para iniciar la ejecución, tal como se representa con la etapa 564. Con la ubicación de la partición activa, el cargador inicial de programas (según se ha diseñado) carga el contenido de la partición activa (la partición del núcleo) en la ROM en el caso de que sea NAND (o, posiblemente, de NOR), o bien salta a la dirección de arranque dentro de la partición si es un núcleo NOR (ejecutar *in situ*). Si se trata de un arranque normal, el núcleo arranca y sigue cargando los componentes del sistema operativo también situados en la partición del núcleo hasta el punto en que el sistema operativo es capaz de leer de la partición del sistema. Cualquier cosa no situada en la partición del núcleo en este punto es buscada (o bien cargada o ejecutada *in situ*) desde la partición del sistema hasta que la imagen del sistema operativo está completamente arrancada.

En una implementación, el cargador inicial de programas es un componente de soporte lógico de tamaño relativamente muy pequeño que es arrancado por el vector de puesta a cero de la CPU, y es responsable de cargar/iniciar condicionalmente la imagen del sistema operativo o la imagen del cargador de actualizaciones. Tal como se ha descrito en lo que antecede, es preciso que el cargador inicial de programas pueda leer diversos registros, incluyendo la bandera de la RAM, una memoria flash/bandera no volátil y un conmutador de soporte físico, para determinar si debe arrancar la imagen normal del sistema operativo o la imagen del cargador de actualizaciones. Más en particular, el cargador inicial de programas necesitará verificar cualquier bandera puesta por la aplicación UpdateBIN (por ejemplo, después de una validación) o por la aplicación de carga de actualizaciones, porque, típicamente, la imagen del sistema operativo se ejecuta con un sistema de ficheros en memoria flash de solo lectura. Para tener en cuenta las condiciones de corte de corriente, la bandera del modo de actualización debería ser no volátil y, así, si se es incapaz de almacenar una bandera no volátil antes de rearrancar, entonces la aplicación de carga de actualizaciones pone la bandera no volátil cuando se la ejecuta por vez primera y quita la bandera tras la terminación de la instalación con éxito, haciendo con ello que la bandera sea persistente entre cortes de corriente.

Además, es preciso que el cargador inicial de programas analice las estructuras de la partición de memoria en diversos tipos de tecnologías de memoria para localizar las imágenes, y que sea capaz de gestionar tipos comprimidos de imágenes, así como XIP en flash (ejecutar *in situ* en la memoria flash) e imágenes en RAM (copiadas a la RAM para su ejecución). El cargador inicial de programas, en general, abstrae los atributos de memoria, como estructuras de partición, bloques reservados, bloques malos y similares, así como los detalles específicos de cualquier tecnología de memoria (por ejemplo, flash NOR, flash NAND, HDD, DOC y similares) para cualquier código proporcionado por el fabricante original del equipo. El cargador inicial de programas puede validar la integridad de la imagen del sistema operativo y/o de la imagen del cargador de actualizaciones (por ejemplo, llevando a cabo sumas de comprobación o verificaciones de firmas) antes de arrancar la imagen para detectar actualizaciones maliciosas y proporcionar con ello un medio para verificar código fiable para la gestión de derechos digitales, que necesita confiar en que un UUID devuelto desde la imagen del núcleo está siendo proporcionado por código que no ha sido alterado maliciosamente.

Así, como puede verse en la FIG. 5 y como se muestra en la FG. 3, cuando, en el modo de actualización, el cargador 222 inicial de programas descomprime el cargador 220 comprimido de actualizaciones en la RAM 304, ya que el cargador 308 descomprimido de actualizaciones deja la memoria flash desbloqueada para dar al cargador 308 descomprimido de actualizaciones acceso de escritura a la memoria flash del sistema y salta al código del cargador no comprimido de actualizaciones para comenzar la ejecución. La aplicación de carga de actualizaciones, cuando es arrancada por vez primera por el cargador inicial de programas, precisará almacenar una bandera en la memoria no volátil que indique que el dispositivo está en modo de actualización. Esto se debe a que la imagen del sistema operativo probablemente tendrá un sistema de ficheros en memoria flash de solo lectura y será incapaz de poner esta bandera, que es importante para la recuperación de cortes de corriente. El cargador inicial de programas también verificará la existencia de una bandera de RAM como esta bandera no volátil cuando se determina si cargar la imagen del cargador de actualizaciones o la imagen del sistema operativo; la bandera de la RAM es puesta (directa o indirectamente) por la aplicación UpdateBIN.

Pasando a una explicación del procedimiento de actualización, según se describe en lo que antecede, el procedimiento para actualizar el contenido del núcleo o de la partición del sistema se ejecuta después de que el cargador de actualización arranque completamente y se cargue en la RAM, y opera para encontrar los paquetes validados (según ha especificado el validador) en la memoria de usuario. Obsérvese que el cargador de actualizaciones contiene el o los necesarios controladores del sistema de ficheros y comienza el procedimiento de actualización paquete por paquete. Se registra la información del paquete para los paquetes actualizados en el sistema de ficheros para su uso por el validador en futuras actualizaciones. Según un aspecto de la presente invención, para proporcionar actualizaciones a prueba de fallos, el cargador inicial de programas es capaz de recuperarse de un corte aleatorio de corriente en cualquier instante durante el procedimiento de carga. Con este fin, se hace un seguimiento del progreso de la actualización en un registro de transacciones para permitir actualizaciones con recuperación en avance en el supuesto caso de que ocurriera un corte de corriente durante la actualización.

En el modo de actualización, el cargador de actualizaciones se ejecuta y arranca una aplicación de actualización, que es un ejecutable que forma parte de la imagen 308 del cargador de actualizaciones (FIG. 3) y es responsable de la aplicación del contenido de los paquetes a la partición 204 del NK/núcleo y a la partición 206 del sistema. En otras palabras la aplicación de actualización que el cargador inicial de programas carga durante un procedimiento de actualización es responsable de la aplicación de las actualizaciones de paquetes a una imagen. Obsérvese que la imagen del cargador de actualizaciones contiene un conjunto mínimo de módulos necesarios para la aplicación de actualización, tales como nk, filesys, controlador de memoria flash, coredll, IMGFS, etcétera.

Las FIGURAS 6-8 describen el procedimiento de actualización en su conjunto, la aplicación de actualizaciones de la partición del NK/núcleo y las actualizaciones de la partición del sistema, respectivamente. La aplicación de la actualización funciona con el validador de paquetes para validar los paquetes y con un módulo ROMIMAGE para gestionar la asignación virtual y física, las correcciones de direcciones y las funciones de ordenación. Como puede verse en la FIG. 6, la aplicación de actualización recupera la lista de instalación de paquetes (por ejemplo, del registro) en la etapa 600, valida los paquetes por medio de validador (etapa 602). En el modo de actualización, la aplicación de actualización es responsable de la aplicación del contenido de los paquetes a la partición del NK/núcleo y a la partición del sistema.

Antes de que la aplicación de actualización aplique el contenido del paquete a la memoria no volátil, los paquetes son validados, lo cual incluye (entre otras cosas) la comprobación de las firmas para verificar que la actualización proviene de una fuente fiable. Como puede apreciarse inmediatamente, saber que una actualización de un paquete proviene de una fuente fiable y garantizar que haya un único portero fiable que tiene acceso de escritura a la memoria flash es importante cuando se intenta proteger la integridad de la memoria flash del dispositivo. Los paquetes de actualización se firman durante el procedimiento de construcción y, siempre que estén firmados por una de posiblemente muchas fuentes fiables diferentes por paquete para una actualización, se les permite proseguir el procedimiento de validación de paquetes descrito más arriba. Si no están firmados o no están firmados por una fuente fiable, falla la validación del paquete y, así, no se actualiza.

Obsérvese que el diseño del sistema es tal que el único componente de soporte lógico al que se permite actualizar el contenido de la memoria flash es también el mismo componente responsable de comprobar la validez (incluyendo la comprobación de la firma) de cualquier paquete puesto en cola para su instalación (el cargador de actualizaciones). Obsérvese también que el cargador de actualizaciones no puede incluir ningún código en el que no se confíe ni de un tercero, como un sistema operativo de uso general. Así, este sistema ejecuta únicamente código en el que se confía y es menos susceptible a las alteraciones. Cuando no se ejecuta el cargador de actualizaciones, el diseño hace uso de un mecanismo de bloqueo mediante soporte físico que prohíbe escribir a la parte flash (a nivel de soporte físico) sin primero poner a cero la parte que está típicamente ligada a la CPU (poniendo así ambas a cero). Además, la imagen del sistema operativo contiene una versión de solo lectura del sistema de ficheros, garantizando adicionalmente que el contenido de la memoria flash no es actualizable, a no ser que el sistema esté dentro del contexto del cargador de actualizaciones, en el que se garantiza que se llevan a cabo validaciones de paquetes y comprobaciones de seguridad.

La verificación también incluye comprobar la corrección del contenido del paquete correlacionando la información del fichero de manifiesto del dispositivo con el contenido del paquete y garantizando que existe un fichero (por nombre) en un paquete y solo uno. La validación también verifica que las versiones necesarias de los paquetes ya están instaladas en el sistema o que están en cola para su instalación. Por ejemplo, si la versión 1.0 del paquete A ya está instalada en el sistema junto con un paquete A delta/de diferencias que lleva la versión 2.0 a la versión 3.0, es preciso que haya en cola un paquete para su instalación que lleve el paquete A hasta la versión 2.0 para que el paquete delta/de diferencias pueda ser instalado. Otra verificación adicional incluye verificar que se satisfacen las necesarias dependencias de los paquetes. Por ejemplo, si la versión 2.0 del paquete A depende del contenido de la versión 3.0 del paquete B, la verificación comprueba que este paquete ya esté instalado o que esté en cola y validado para su instalación. El procedimiento de validación de paquetes también se describe en la solicitud de patente estadounidense, titulada “Determining the Maximal Set of Dependent Software Updates Valid for Installation”, mencionada anteriormente.

El resultado del procedimiento de validación sobre el dispositivo es una lista de paquetes (y de sus ficheros) que pueden ser instalados en el dispositivo porque han satisfecho los requisitos de validación. El procedimiento de validación también genera una lista de paquetes que no pueden ser instalados, junto con datos de por qué el procedimiento de validación falló para cada paquete específico. El resto del procedimiento de instalación hace uso únicamente de los paquetes que pasaron la validación.

Volviendo a la FIG. 6, el cargador de actualizaciones es responsable de la aplicación de cualquier actualización a la partición del núcleo (etapas 604 y 606) y de cualquier actualización del sistema a la partición del sistema (etapas 608 y 610). Si hay otras actualizaciones, como a una sección de memoria de radio reservada (que existe fuera del ámbito de una partición, es decir, situada únicamente en una ubicación especificada en la memoria flash, estando definidas las estructuras de la partición “alrededor” de las áreas reservadas), estas son gestionadas similarmente por medio de las etapas 612 y 614, como se describe también con referencia a la FIG. 10.

Los paquetes pueden ser borrados por medio de la etapa 616. La bandera se pone a un valor (por ejemplo, quitada_ en la etapa 618 para arrancar la imagen del sistema operativo, la cual, si se realizaron actualizaciones del núcleo o actualizaciones del sistema, se actualiza ahora y se rearranca el sistema (etapa 620).

Las FIGURAS 7A y 7B muestran la lógica general de la actualización del núcleo, en la que (suponiendo una ejecución *in situ*) se realiza una copia de seguridad del núcleo existente leyéndolo a la RAM y comprimiéndolo en la memoria de usuario. Esto es representado por medio de la etapa 700, en la que se ha hecho que el identificador obtenido de la partición XIP, expuesta por medio del sistema de ficheros como una carpeta XIP con un fichero por defecto, sea para la lectura de toda la imagen a la memoria local intermedia. Se crea un nuevo fichero en la memoria de usuario, se llama a la rutina de compresión para que comprima la imagen y, a continuación, los datos del fichero comprimido se escriben al nuevo fichero.

La etapa 702 lee cabeceras y otros metadatos. Con este fin, el procedimiento recorre la pToc (estructura de la tabla de contenido) para determinar la ubicación de cada sección, rellenando una lista de módulos antiguos con la información de módulos, y fijado un puntero de datos de sección para apuntar al lugar apropiado en la copia creada en la RAM para la copia de seguridad. Partes de la TOC se mantienen y se copia a la nueva TOC cerca del final de la actualización de la partición NK/XIP. Obsérvese que, aunque lea las cabeceras de un módulo, el procedimiento también lee el fichero de manifiesto antiguo del dispositivo en busca de banderas adicionales (compresión, módulo del núcleo, invalidación de ranura 1 [bandera L] y similares).

Acto seguido, por medio de las etapas 704 y 722, cada paquete es procesado procesando cada módulo del paquete (etapas 706 y 720), determinando si el fichero es un módulo bindiff (de diferencias binarias). Esto implica realizar un bucle en cada módulo de cada paquete y aplicar una actualización canónica, en la que se proporciona la copia completa del nuevo fichero, o aplicar una actualización de diferencias binarias, en la que únicamente se proporcionan las diferencias binarias del fichero. El procedimiento de aplicación de un fichero de diferencias binarias se denomina parcheo binario; se realiza un bindiff separado en cada sección, uno cada vez. Las cabeceras del nuevo módulo se proporcionan en la forma canónica. Puesto que los ficheros se añaden uno por uno, son añadidos

(en forma de objetos de fichero) a una nueva lista de ficheros. Al final de esta etapa, hay una nueva lista de ficheros que contiene versiones no corregidas en dirección de los módulos y los ficheros no actualizados. Si eran ficheros existentes, son entonces eliminados de la lista antigua, de modo que, al final de esta etapa, la lista antigua contiene únicamente ficheros para los que no había actualización alguna. Obsérvese que el procedimiento sabe qué paquete o paquetes están asociados con la partición en base a la presencia del fichero o de los ficheros de manifiesto de paquetes en esa partición. Cualquier paquete nuevo que no se haya instalado en el dispositivo se guarda en la partición IMGFS.

Así, si no se trata de un módulo de diferencias binarias en la etapa 708, el módulo es simplemente añadido a la nueva lista en la etapa 718. Si se trata de un módulo de diferencias binarias, es preciso aplicar las diferencias al código antiguo, lo que llevan a cabo las etapas 710-716 sección por sección.

Más en particular, tal como se ha descrito en lo que antecede, el paquete delta/de diferencias contiene uno o más ficheros de diferencias binarias, correspondiendo cada fichero de diferencias binarias a un fichero/módulo específico. El fichero de diferencias se basa en una revisión de referencia específica a la que se aplica en el momento de la instalación, y el fichero de diferencias es, típicamente, menor que todo el fichero canónico de la versión resultante, por lo que mejora los tiempos/costes de descarga y reduce la cantidad de recursos generales de memoria temporal necesarios durante el proceso de instalación. Los ficheros de diferencias binarias son generados en el momento de la construcción en base a la detección de las diferencias en los ficheros/módulos entre dos paquetes canónicos diferentes. Si los ficheros son módulos ejecutables, han sido procesados por el montador de enlaces y ubicados en una dirección definida por el montador de enlaces. No han sido reubicados en una ubicación final, lo que se realiza en el dispositivo en el momento de la instalación, tal como se ha descrito más arriba.

Para que una diferencia binaria de un módulo ejecutable sea aplicada a una versión de referencia que ya está en el dispositivo, es preciso que esa versión de referencia vuelva a ser desreubicada a la dirección base generada por el montador de enlaces a partir de la cual se calculó la diferencia binaria. Una vez que la diferencia binaria se aplica a la versión desreubicada del módulo, puede volver a ser reubicada en la dirección base virtual apropiada. El procedimiento de desreubicación y de reubicación es el mismo, y está acompañado por el desplazamiento de la dirección base del módulo a un ubicación específica.

Cuando ha sido procesado cada paquete, la actualización del núcleo prosigue a la etapa 724 de la FIG. 7B, en la que se añade a la nueva lista de módulos cualquier módulo que no haya sido modificado. En este punto, la lista antigua de ficheros contiene los módulos y los ficheros que no han sido actualizados; no es preciso que las direcciones de estos sean devueltas a las direcciones base originales para que haya una lista uniforme de módulos y ficheros sin corrección de dirección que pueda ser procesada en su totalidad. Los objetos de la antigua lista de ficheros que aparecen en el nuevo fichero de manifiesto del dispositivo son transferidos a la nueva lista de ficheros. Obsérvese que si un fichero debe ser borrado, no aparecerá en el nuevo fichero de manifiesto del dispositivo y, por lo tanto, no se pondrá en la nueva lista de módulos.

La asignación y la corrección de direcciones de memoria se llevan a cabo en las etapas 726 y 728, y la nueva imagen del núcleo de ejecución *in situ* se vuelve a escribir a la memoria flash en la etapa 730, lo que también se realiza por medio de una herramienta de imágenes de discos y se implementa en romimage.dll (según se describe más abajo). Hay una función en romimage que toma una lista de módulos y construye la imagen asignando una nueva memoria intermedia para construir una nueva imagen, entrando en un bucle con la lista de módulos para copiar los datos de la sección en la ubicación apropiada según especifica el puntero a los datos y las cabeceras de copia y las cadenas de los nombres de los ficheros a las ubicaciones debidas, según se especifica en la tabla de contenido (TOC). Según se describe más abajo, se escribe una nueva TOC al final de la imagen y se actualiza un puntero pTOC para que apunte a la ubicación en la que está situada la nueva TOC. Como también se describe más abajo, se hace un seguimiento de las bandas de secciones de datos en la ranura 0 para la imagen NK/XIP, las cuales se concatenan con las bandas del IMGFS y se les da salida en el fichero IMGFS que contiene una estructura ROMINFO durante la actualización del IMGFS.

Antes de escribir nada a la memoria flash, el procedimiento anota en el fichero de registro que la escritura está en curso. Una vez que se completa la escritura, ello se consigna, con lo que, si se corta la corriente antes de que se borre el fichero de copia de seguridad, se conoce el estado correcto.

Si tiene éxito total, el fichero de copia de seguridad se borra en la etapa 732, que incluye escribir en el fichero de registro que la partición XIP está completa después de borrar el fichero. Si ocurre un error durante cualquier etapa del procedimiento de actualización NK/XIP, como que la imagen sobrepasase el tamaño de la partición, se restaura la versión de la copia de seguridad de la imagen, con una interfaz adecuada de usuario que indique que ocurrió un error durante la actualización y que se restauró la imagen original. Si la versión de la copia de seguridad no pudo ser escrita con éxito (como ante un error de la memoria flash), se proporciona un mensaje diferente de la UI para indicar que ocurrió un error durante la actualización, pero que la imagen original no pudo ser restaurada y puede estar corrupta.

Además, se consigna el estado de la actualización para UpdateBin. Obsérvese que esto es diferente del fichero de registro que la aplicación de actualización usa internamente para el registro de transacciones.

En este punto, hay una única lista de ficheros de los módulos y los ficheros que compondrán la imagen. Esto es igual que en la instalación inicial tal como es realizada por la herramienta de imágenes de discos en conjunto con una romimage.dll, según e describe en la solicitud de patente estadounidense relacionada, titulada “Creating File Systems Within a File In a Storage Technology-Abstracted Manner”, mencionada anteriormente.

A alto nivel, el procedimiento de instalación de un paquete implica la extracción del contenido de un paquete y aplicarlo al dispositivo. Sin embargo, la mecánica incluye varias etapas que giran fundamentalmente en torno al concepto de correcciones de direcciones de código o reubicaciones de código. La presente invención lleva a cabo correcciones de dirección/reubicaciones en el dispositivo en el momento de la instalación, en vez de hacerlo en el sistema de construcción en el momento de la construcción. Un beneficio es la adaptabilidad de una instalación de un paquete en un sistema específico, porque la instalación no requiere que toda la imagen del dispositivo esté disponible en el momento de la construcción. En vez de ello, permite que las actualizaciones del paquete sean tratadas como entidades en gran medida aisladas de la configuración específica de la imagen del sistema operativo instalada en un dispositivo.

Para que un módulo ejecutable sea reubicable, es preciso que el módulo contenga información que dirija a un localizador a las direcciones dentro del módulo que necesitan ser actualizadas, dado que cambia la ubicación de la dirección base para el módulo. El modelo de actualización de la imagen hace uso de un modo de codificación de la información de reubicación para proporcionar esta información, de forma comprimida, dentro del propio módulo ejecutable.

Una herramienta denominada RelMerge, tal como se describe en general en la solicitud de patente estadounidense “Self-Describing Software Image Update Components”, mencionada anteriormente, convierte un módulo (.EXE o .DLL) en un formato que es adecuado para su inclusión en un paquete y, por lo tanto, para su instalación en el dispositivo. Esto incluye la conversión de la información de reubicación a un formato comprimido que es más adecuado para el almacenamiento a largo plazo en el dispositivo, y la conversión de las cabeceras de ficheros a las variantes _rom usadas por el sistema de ficheros ROMFS en el dispositivo. Además, también se elimina cualquier exceso de relleno de todas las secciones del fichero cuando se transmite.

Una vez que está codificado con esta información, la dirección base virtual de un módulo ejecutable puede ser cambiada, y todas las referencias relevantes a direcciones dentro del módulo pueden ser modificadas para dar cuenta del cambio en la dirección base. El procedimiento de reubicación hace uso de una biblioteca de código que es compartida entre el dispositivo y el escritorio. En este caso, se usa para crear la imagen inicial de fabricación, de modo que las reubicaciones propiamente dichas se lleven a cabo en los módulos de código, y también para crear una lista del espacio de direcciones virtuales usado a medida que se instala cada módulo de actualización, permitiendo así que el consumir de la API confíe en que las reubicaciones de módulos no se solaparán. La creación de la imagen inicial de fabricación por medio de una herramienta de imágenes de discos y un componente romimage.dll se describe en la solicitud de patente estadounidense relacionada, titulada “Creating File Systems Within a File In a Storage Technology-Abstracted Manner”, mencionada anteriormente.

En general, la asignación virtual/física y la corrección de direcciones funcionan igual que en la herramienta de imágenes de discos, o sea, ubicando la antigua tabla de contenido (TOC) a partir de la imagen NK/XIP y ubicando la antigua ROMINFO. Se sitúa un puntero a la TOC en una ubicación fijada (por ejemplo, en el desplazamiento 0x44) en la imagen. Se lee la estructura ROMINFO antigua del fichero del IMGFS denominado “.rom”.

Para asignar direcciones virtuales, un asignador de la ranura 0 comienza con la parte superior de la ranura 0 y acaba en base al valor dwSlot_0_DllBase de la estructura ROMINFO del IMGFS. Un asignador de la ranura 1 comienza con la parte superior de la ranura 0 y acaba en base al valor dwSlot_1_DllBase de la estructura ROMINFO del IMGFS. El código y los datos son entonces corregidos en cuanto a dirección a la nueva asignación VA y se vuelve disponible toda la lista de módulos. Se comprimen las secciones marcadas como comprimidas y el tamaño se registra en un objeto de módulo.

Para asignar espacio físico para la imagen, se usa RAMIMAGE para la RAM y ROMIMAGE para la memoria flash. Para la RAM, un asignador físico RAMIMAGE usa la physfirst de la antigua TOC como la dirección de inicio del asignador físico. La terminación del asignador físico es especificada inicialmente como ulRAMEnd de la antigua TOC. Para RAMIMAGE, el asignador físico devuelve RamStart = PhysFirst + PhysLength (la longitud real requerida para la imagen); las ubicaciones de las direcciones virtuales para la sección de copia se determinan empezando con RamStart, y, de ahí, se corrige la dirección del texto o los datos que se refieren a la sección de copia.

Para la ROM, un asignador físico ROMIMAGE usa la physfirst de la antigua TOC como la dirección de inicio del asignador físico, mientras que la terminación del asignador físico se determina obteniendo la longitud de la partición en la memoria flash, usando GetPartitionInfo.

Obsérvese que se genera una TOC actualizada durante el procedimiento de asignación física.

A diferencia de la partición XIP, en la partición IMGFS no se rehará toda la imagen, puesto que esta sería demasiado grande para hacer una copia de seguridad de la misma. En vez de ello, se usa el sistema de ficheros

IMGFS para aplicar actualizaciones a módulos y ficheros individuales, uno por uno. Además, obsérvese que los módulos y los ficheros individuales pueden ser demasiado grandes para realizarlos en su conjunto, y, así, las actualizaciones pueden ser aplicadas por medio de bloques más pequeños, tal como se describe más abajo con referencia a la FIG. 9.

La actualización del IMGFS usa la misma lista de ficheros que la actualización de NT, aunque una lista se actualizará a medida que prosigue el procedimiento. Tal como se describe más abajo, el IMGFS puede, en primer lugar, simular el procedimiento de actualización, requiriéndose dos pases por todo el procedimiento. En un primer pase, tanto el IMGFS como la aplicación de actualización no escriben nada definitivamente a la memoria flash. Si la simulación tiene éxito, se realiza un segundo pase para escribir definitivamente los cambios de forma real. Si la simulación falla, se reintenta la simulación después de volver a ejecutar al validador de paquetes pasándole una lista de paquetes, excluyendo el paquete que falló. La simulación se ejecutará una vez más con la nueva lista que devuelve el validador de paquetes. Las simulaciones se reintentan hasta que la simulación tiene éxito o hasta que no hay ningún paquete que pueda aplicarse. Esto protege contra la corrupción de los datos que puede haber ocurrido en el paquete que se firmó a pesar de la corrupción.

Las FIGURAS 8A y 8B muestran la lógica general para la actualización de la partición del sistema, en la que la aplicación de actualización funciona en conjunto con una versión de lectura/escritura del controlador del sistema de ficheros de imagen (IMGFS) para gestionar la partición del sistema. Obsérvese que en la operación normal (sin actualizaciones) en los sistemas típicos, la partición del sistema es de solo lectura para el controlador del IMGFS (y la memoria flash está protegida). Además, obsérvese que, a diferencia de las actualizaciones del núcleo, con las actualizaciones del sistema no se realiza una copia de seguridad en su conjunto, aunque los registros de las transacciones pueden restaurar el dispositivo a la situación en la que se encontraba en el procedimiento de actualización del sistema antes de un fallo (por ejemplo, un corte de corriente), y las actualizaciones del sistema pueden reanudarse desde ese punto.

En la etapa 800, se exploran los módulos existentes en la partición del sistema (la región del imgfs) y son añadidos a un asignador de direcciones virtuales (VA). Durante el procedimiento de exploración, el procedimiento también detecta casos de borrado. Más en particular, cuando se itera en el IMGFS usando el fichero existente de manifiesto del dispositivo, el procedimiento hace una comprobación para ver si el fichero aparece en el nuevo fichero de manifiesto del dispositivo para el paquete. Si no, el fichero es borrado de la partición IMGFS y el procedimiento prosigue con el siguiente fichero. Si es así, se obtienen nuevas banderas de módulos del nuevo fichero de manifiesto del dispositivo y las cabeceras de los módulos se cargan en la estructura del módulo usando IOCTL_BIN_GET_E32 e IOCTL_BIN_GET_O32. Obsérvese que los datos de los módulos y los ficheros no se leen en este momento, dado que esto no es necesario. Una vez que se ha leído de entrada cada cabecera del módulo, se pasa la lista de módulos a los asignadores para reservar direcciones (incluyendo secciones reservadas que son conocidas a partir de la tabla reservada almacenada en la memoria flash).

Una vez que los módulos existentes han sido explorados y se han definido las direcciones virtuales actualmente asignadas, el procedimiento está listo para iniciar la actualización. Con este fin, el procedimiento entre en un bucle para cada uno de los paquetes (por medio de las etapas 802 y 828) y realiza el bucle (por medio de las etapas 804 y 826) con cada fichero del paquete, procesando cada fichero uno por uno. Los ficheros son procesados en orden de la ganancia neta en tamaño, comenzando con los ficheros que se comprimen al máximo y acabando con los ficheros que aumentan al máximo. La delta en el tamaño del fichero se determina comparando el fichero antiguo de manifiesto del dispositivo con el nuevo.

Si se corta la corriente durante este proceso, un fichero de registro dirá exactamente en qué paquete y en qué fichero del paquete se detuvo el procedimiento. El procedimiento se reanudará en el fichero en el que se detuvo, tratándose los ficheros actualizados que estaban completos en el momento del corte de corriente como ficheros existentes. Para la recuperación, cualquier módulo nuevo que estuviera en vías de actualización no se incluirá en la reserva inicial mientras el módulo antiguo esté en la reserva. Como es habitual, se desasigna el espacio de direcciones virtuales del módulo antiguo. Si ya se ha escrito la nueva cabecera, contendrá la asignación del espacio de direcciones virtuales del nuevo módulo. En este caso, se añade al asignador del espacio de direcciones virtuales usando una función Reservar. Si la nueva cabecera no se ha escrito aún, se asigna el nuevo espacio de direcciones virtuales de la forma normal usando la función Asignar.

Parte del procesamiento anterior incluye (para ficheros ejecutables, no de datos) actualizar la asignación de direcciones virtuales para los módulos de actualizaciones determinar si cambió la dirección virtual y, en caso afirmativo, desasignando el antiguo espacio de direcciones virtuales y asignando el espacio de direcciones virtuales para el nuevo módulo, tal como se representa por medio de la etapa 806. Si el tamaño de las direcciones virtuales cambia, se invoca una función Desasignar para eliminar del asignador el antiguo espacio de direcciones virtuales y se invoca una función Asignar para para asignar nuevo espacio de direcciones virtuales para el nuevo módulo, pasándose requisitos apropiados de alineamiento en dependencia del código o los datos. La desasignación puede corresponder a una orden de borrar el módulo, en cuyo caso el módulo antiguo es borrado por medio de las etapas 808 y 824 (pero no si lo que acaba de instalarse es un nuevo módulo). Si no es una orden de borrado, la etapa 810 invoca una función CreateFile en el IMGFS, por ejemplo usando nuevo_<nombre_del_módulo>.

<extension_del_módulo> para dar nombre al nuevo fichero, para crear un nuevo fichero en el sistema de ficheros de imagen (para la recuperación si se corta la corriente en torno a esta operación, el fichero no vuelve a ser creado si existe).

Para los ficheros ejecutables, la etapa 812 escribe cabeceras apropiadas en los mismos leyendo la nueva cabecera al objeto del módulo y actualizando la cabecera con la nueva asignación del espacio de direcciones virtuales, tal como se representa por medio de la etapa 812, usando IOCTL_ BIN_SET_E32 e IOCTL_BIN_SET_O32. Cada operación es atómica en el IMGFS. En este punto, la cabecera debería estar completa. El puntero de los datos (dataptr) es desechado (puesto a cero), dado que el IMGFS falsifica este valor cuando el núcleo se lo pide a las E32/O32. Las cabeceras están en la forma canónica, incluso en el fichero de diferencias binarias.

Si se corta la corriente con respecto a la escritura de las cabeceras, si ambas cabeceras existen, entonces no es preciso efectuar recuperación alguna. Si únicamente se ha escrito la E32, entonces solo se vuelve a la escribir la cabecera O32. La existencia de cada cabecera es conocida por medio de IOCTL_BIN_GET_E32 e IOCTL_BIN_GET_O32.

Para aplicar actualizaciones, se hace notar que la actualización del fichero puede estar en la forma canónica o en la forma de diferencias binarias. Si el fichero actualizado no es un módulo, los datos para ese fichero son procesados de la misma manera que una sección individual. Si el fichero es un módulo, entonces cada sección es procesada de una en una, en un orden de ganancia neta, empezando con la sección que se comprime al máximo. Se procesa en primer lugar la sección .creloc. Se escribe cada sección a un nuevo flujo. Si el fichero es un fichero regular de datos, los datos se escriben al flujo por defecto.

Si el módulo es una actualización canónica según se evalúa en la etapa 814, las etapas 816-822 procesan cada una de las secciones en el módulo, incluyendo la reubicación de una sección entera en la dirección asignada por el asignador virtual (etapa 818) y escribiendo la sección entera como un nuevo flujo en el nuevo fichero del IMGFS (etapa 820). A continuación, se borra el módulo antiguo en la etapa 824, a no ser de nuevo que el módulo “antiguo” sea en realidad un módulo nuevo.

En el caso canónico, la recuperación de un corte de corriente se lleva a cabo obteniendo el tamaño de cada flujo del nuevo fichero que ha sido escrito y comparándolo con el tamaño de esa sección en el fichero canónico. Si un flujo en el nuevo fichero es menor que el fichero canónico, entonces ese es el lugar en el que se detuvo el procedimiento y puede reanudarse la copia encima de los datos.

En el caso de las diferencias binarias, el procedimiento patchbin se realiza bloque a bloque (tal como se describe más abajo con referencia a la FIG. 9), igualando un bloque el tamaño de la página o el sector. Esto se debe a que puede no haber suficiente espacio en la memoria flash para mantener juntas tanto la versión antigua como la versión nueva de una sección. Los bloques para el nuevo fichero pueden ser creados en cualquier orden especificado por el fichero de diferencias binarias, de modo que el procedimiento patchbin pueda efectuarse con el uso óptico del espacio de la memoria flash. A medida que se crean los nuevos bloques, los bloques antiguos pueden ser desasignados, puesto que ya no son necesarios.

Con este fin, si el módulo no es una actualización canónica en la etapa 814, el procedimiento se ramifica a la FIG. 8B, etapa 832, en la que, en conjunto con la etapa 852, se entra en un bucle con cada sección del módulo para aplicar actualizaciones binarias bloque a bloque. Más en particular, la etapa 834 crea un flujo poco denso en el nuevo fichero del módulo para esta sección, y la sección antigua se lee a la RAM (etapa 836). Toda la sección antigua vuelve a ser reubicada en la dirección base original (etapa 838).

Para cada nuevo bloque del fichero de diferencias binarias (etapas 840 y 850) se construye un nuevo bloque en base a las órdenes de las diferencias binarias (etapa 842). Se reubica el nuevo bloque en la dirección asignada por el asignador de direcciones virtuales en la etapa 844 y se escribe al flujo poco denso en la etapa 846. Se desasigna cualquier bloque antiguo procedente del flujo antiguo que ya no se necesite, tal como se representa en la etapa 848. El concepto de actualizaciones de bloques también se describe más abajo con referencia a la FIG. 9.

En el caso de las diferencias binarias, un corte de corriente se gestiona acotando qué sección se interrumpió comparando el tamaño del flujo con el especificado en la cabecera. Una vez que se determina la sección, se crea una instancia del patchbin, según es habitual.

Cuando vuelve al bloque siguiente en la sucesión (la salida del patchbin precisa ser reproducible/determinista), se realiza una comprobación en cuanto a si el bloque ha sido escrito definitivamente. Dado que las escrituras en bloques son atómicas, los bloques parciales no pueden ser escritos definitivamente. Si el bloque ya ha sido escrito definitivamente, se descartan los datos devueltos del patchbin y se lleva a cabo una verificación para comprobar que los bloques antiguos apropiados ya han sido liberados. Una vez que se encuentra el bloque que se interrumpió (es decir, aquel que aún no ha sido escrito definitivamente), el procedimiento prosigue de la manera normal.

Para finalizar, si este era un fichero o un módulo existentes y, en el caso de las diferencias binarias, entonces se borra el módulo antiguo (en el caso canónico, el fichero ya se borró) y el fichero de registro consigna que el módulo

ha sido finalizado. Si un paquete ha sido finalizado, se registra esto en su lugar. Cuando se termina con todos los paquetes, se actualiza el fichero .rom con una estructura ROMINFO actualizada, que contiene la base Ranura0, la base Ranura1 y las bandas de los datos de Ranura0 para las imágenes tanto del NK/XIP como del IMGFS.

Si ocurre un error durante cualquier etapa del procedimiento de actualización del IMGFS, como que el fichero de diferencias binarias no tuviera suficientes bloques de memoria flash disponibles, se cancela el procedimiento de actualización y no se intenta una recuperación, dado que es probable que sea irrecuperable, y se presenta un mensaje apropiado. Obsérvese que realizar una simulación de antemano evita tal situación.

Como puede verse, las actualizaciones a la partición del núcleo se tratan de un modo diferente al de las actualizaciones a la partición del sistema, aunque, desde el punto de vista de la construcción y el empaquetado, son iguales. Más en particular, cuando se actualiza la partición del núcleo, se busca en la memoria flash y se pone en la RAM cierta cantidad de un fichero NK.NB0 (la imagen firmada de la partición del núcleo), los componentes constituyentes se actualizan y se corrigen sus direcciones según resulte apropiado, y luego se vuelve a escribir en bloques contiguos a la memoria flash el contenido .NBO modificado. Esto permite saltarse cualquier bloque malo según resulte necesario.

Según otro aspecto de la presente invención, tal como se describe más arriba, los mecanismos consideran el concepto de orden optimizado para la forma en la que se aplican los ficheros de diferencias binarias. Según se entiende, se requiere cierta cantidad de memoria temporal para aplicar un fichero de diferencias binarias a la imagen de referencia y generar así el módulo/fichero actualizado resultante. En el momento de la construcción, cuando se genera el fichero de diferencias binarias, se ejecuta un procedimiento de ordenación en la imagen resultante para ordenar la manera en la que se aplican los bloques dentro del fichero de diferencias binarias, de modo que la necesaria memoria temporal disponible se mantenga en cierto máximo especificado. Llevando a cabo esta operación de optimización en el sistema de construcción, la presente invención garantiza que si el dispositivo tiene suficientemente memoria para el aumento (o compactación) global de una imagen y tiene el espacio temporal necesario, entonces es seguro que el dispositivo tiene suficiente memoria para completar el procedimiento de actualización. La verificación del tamaño se lleva a cabo durante el procedimiento de validación, antes de comenzar la actualización.

La FIG. 9 representa en general el concepto de actualizaciones de bloque. En general, los ficheros del sistema pueden ser grandes y, por ello, podrían en otro caso consumir gran cantidad de memoria temporal (más de la que podría haber disponible) si se crease un nuevo fichero como un todo y el fichero de diferencias binarias se aplicase al fichero antiguo en conjunto. En vez de ello, las actualizaciones de diferencias se aplican a bloques existentes de un fichero que se está actualizando, dando como resultado nuevos bloques actualizados, copiándose los bloques actualizados al flujo del fichero actualizado y desasignándose los bloques antiguos cuando ya no hacen falta.

Para lograr las actualizaciones de bloques, se construye un gráfico de dependencias entre los bloques antiguos y los bloques nuevos, tal como se representa en general en el gráfico 900 en la FIG. 9. Según se entiende, los ficheros 902 de diferencias se aplicarán a los bloques reales de datos representados por los nodos de este gráfico. En una implementación, los bloques tienen cuatro kilobytes de tamaño y se permiten treinta y dos en un momento dado, lo que significa que solo es preciso que haya 128 kilobytes disponibles para garantizar que un sistema pueda ser actualizado. Obsérvese que, en lo anterior, los tamaños ejemplares de los bloques y el límite global son valores arbitrarios, pero es preciso que haya acuerdo entre el proveedor de una actualización del sistema y la aplicación de aplicaciones en el dispositivo. En el caso de una actualización particular sea incapaz de satisfacer los límites convenidos, durante la construcción de la actualización en las instalaciones del vendedor, será preciso que el vendedor rompa los enlaces de dependencia (se requieren actualizaciones no diferenciales para tales bloques).

Tal como se representa en la FIG. 9, un único bloque antiguo de actualizaciones puede proporcionar los datos dependientes (a los cuales se aplica el fichero de diferencias) para uno o más bloques nuevos de actualizaciones, y el nuevo bloque de actualización puede ser dependiente de uno o más bloques antiguos de actualizaciones. En general, las actualizaciones prosiguen aplicando el fichero o los ficheros apropiados de diferencias a un bloque antiguo, hasta que se haya aplicado cada fichero de diferencias para ese bloque antiguo. En este momento, puede romperse el enlace de dependencia (por ejemplo, según se representa en la FIG. 9 por la ruptura en el enlace entre el antiguo bloque X y el nuevo bloque Y), porque el bloque antiguo ya no es necesitado por el nuevo bloque. Dado que este era el único enlace asociado con el antiguo bloque X, el antiguo bloque X puede ser desasignando para liberar espacio para otro bloque. Así mismo, a medida que se rompen los enlaces de dependencia tras la aplicación del fichero de diferencias a un bloque antiguo, cuando un bloque nuevo no tiene ningún enlace de dependencia con un bloque antiguo, o sea, cuando se ha escrito completamente un nuevo bloque por medio del procedimiento de actualizaciones de diferencias, ese nuevo bloque puede ser copiado a la memoria flash según sea apropiado y desasignado de la RAM para liberar espacio para otro bloque. Obsérvese que múltiples bloques pueden ser desasignados a la vez.

Como puede apreciarse fácilmente, el orden de aplicación de las actualizaciones de diferencias puede contribuir a la liberación de la memoria. En general, el orden es aplicar en primer lugar las actualizaciones de diferencias a un bloque antiguo que tenga el menor número de enlaces de dependencia con un nuevo bloque que tenga el mayor

número de enlaces de dependencia. Un algoritmo lleva a cabo una búsqueda a dos niveles (por ejemplo, de un conjunto contador de enlaces) de bloques antiguos de menor recuento que tienen bloques nuevos de mayor recuento.

Con respecto a la realización de una simulación previa a la escritura en memoria flash con actualizaciones de bloques, dado que puede no existir el fichero entero en ningún momento concreto en la simulación, se calculan sumas de comprobación en base a cada bloque que ha de ser escrito, y la suma de comprobación es verificada con una suma de verificación para cada actualización. Si esta suma de verificación generada por la simulación se da por buena, entonces puede llevarse a cabo una actualización real en la memoria flash.

Pasando a las secciones de reserva de la actualización, la FIG. 10 muestra que una actualización de secciones de reserva (por ejemplo, fuera de una partición) se logra de una manera similar a la de las actualizaciones de la partición NK, usando los mismos componentes de actualización. Por medio de las etapas 1000 y 1016, se procesa cada actualización de paquetes relacionados con la reserva procesado cada fichero del paquete (etapas 1002 y 1014), determinando si el fichero es un módulo bindiff (de diferencias binarias). Si no, simplemente se leen los datos del fichero y se escriben desde el fichero a la región reservada en las etapas 1010 y 1012. Si se trata de un módulo de diferencias binarias, se lee la región existente (por ejemplo, a la RAM) y se aplican al mismo las diferencias binarias antes de volver a escribir los datos actualizados a la región reservada, tal como se representa en general por medio de las etapas 1006, 1008 y 1012.

Cuando se ha aplicado el contenido del paquete en el orden apropiado (por versión de paquete) para cada paquete puesto en cola y están completas las actualizaciones, los ficheros del paquete pueden ser eliminados opcionalmente de la memoria de usuario o pueden ser marcados de otra manera como completados. En este momento, una vez que se ha instalado en las particiones apropiadas de la memoria flash el contenido de los paquetes validados, la aplicación de actualización completa su trabajo inhabilitando el modo de actualización (se quita la bandera de actualización de la memoria flash) y se rearranca el dispositivo. Como antes, el cargador inicial de programas detecta el modo actual del sistema, pero, esta vez, a causa de la bandera quitada, se arranca la imagen actualizada del sistema operativo con la memoria flash bloqueada, como se ha descrito antes, y la instalación está completa.

Según un aspecto de la presente invención, se proporcionan procedimientos de actualización a prueba de fallos/con recuperación. Con este fin, una parte del diseño de la actualización de imágenes proporciona una recuperación a prueba de fallos en casos en los que se interrumpe el procedimiento de actualización, por ejemplo debido a un corte inesperado de corriente. El medio mediante el cual se implementa la recuperación a prueba de fallos incluye hacer que el cargador de actualizaciones y la aplicación de actualización sean reentrantes y capaces de decidir el lugar en el que la operación de actualización se detuvo, por medio del registro y de vestigios del sistema de ficheros (es decir, recuperación en avance). También se provee un sistema transaccional de ficheros que da soporte a escribir una actualización a un fichero sin liberar completamente la copia antigua hasta que la nueva actualización haya sido escrita definitivamente a la memoria (esto se realiza en incrementos de subfichero, por ejemplo, bloques, en la partición del sistema). Pueden llevarse a cabo simulaciones de todo el procedimiento de instalación hasta el punto en el que se escribe definitivamente en la memoria para garantizar que se ejecuten casi todas las trayectorias relevantes del código, mientras se reduce el modo de fallo en una actualización real a fallos en el soporte físico (ejemplo, fallo de la memoria flash) o a un posible fallo en las rutinas de soporte lógico de bajo nivel de la memoria flash. Se proporcionan copias de seguridad del NK/núcleo y de las regiones reservadas (por ejemplo, radio), para que, en el caso de que falle una actualización, después de cierto número especificable de reintentos, pueda restaurarse el contenido original de la partición de la imagen y abortarse la instalación (es decir, una recuperación en retroceso).

La aplicación de actualización hace un seguimiento del progreso de la instalación, recuperar el lugar en el que detuvo en el caso de una interrupción inesperada y realiza una copia de seguridad (y potencialmente restaura) el NK/núcleo y las regiones reservadas. La aplicación de actualización actualiza RAMIMAGE, ROMIMAGE, IMGFS y las imágenes reservadas. Las particiones RAMIMAGE y ROMIMAGE se generan de la misma manera en que la herramienta de escritorio de imágenes de discos genera las particiones, es decir, la partición IMGFS se actualiza trabajando con la distribución existente y realizando llamadas al IMGFS y al asignador para disponer debidamente los módulos actualizados, tanto virtual como físicamente. Una actualización reservada, descrita más arriba con referencia a la FIG. 10, se realiza sobreescribiendo toda la región.

En una implementación, cuando se inicia la aplicación de actualización, se asume que los paquetes que deben instalarse se sitúan en un directorio contenedor temporal, según se especifica en un fichero de entrada de la aplicación de actualización en la memoria de usuario, que contiene la entrada para la aplicación de actualización. La ruta del fichero de entrada de la aplicación de actualización está especificada en el registro. A la aplicación de actualización le resulta indiferente el lugar en el que se guarden los paquetes, ya sea la memoria interna de datos o una tarjeta externa de memoria, con la condición de que se proporcione una ruta completa para el directorio contenedor. También se proporciona una ruta a ficheros de mapas de bits que se usan para la actualización; obsérvese que no se está ejecutando el código normal del sistema operativo y, así, se proporcionan mapas de bits para fines de la interfaz del usuario (por ejemplo, para mostrar una barra de avance, qué ficheros están siendo actualizados, mensajes de error y similares).

La aplicación de actualización empieza pasando la ruta del directorio contenedor al validador de paquetes, que devuelve una lista que especifica el orden en el que instalar los paquetes, tal como se describe en la solicitud de patente estadounidense relacionada, titulada “Determining the Maximal Set of Dependent Software Updates Valid for Installation”, mencionada anteriormente. A continuación, el procedimiento del cargador de actualizaciones itera cada paquete y aplica la actualización apropiada, ya sea XIP, IMGFS y/o Reservada, tal como se ha descrito más arriba con referencia a la FIG. 6.

Puede considerarse que la aplicación de actualización tiene varios procedimientos/componentes, incluyendo un procedimiento de actualización NK/XIP que es responsable de las actualizaciones de la partición NK/XIP. La imagen en esta partición puede ser una ROMIMAGE o una RAMIMAGE (siendo una ROMIMAGE una imagen que se ejecuta directamente desde la memoria flash y que requiere una flash NOR, mientras que una RAMIMAGE es una imagen que se cargó en la RAM y que puede ser almacenado en una flash NOR o NAND). Con independencia del tipo de imagen, las actualizaciones de región funcionan en conjunto directamente con el controlador de bloques cuando se lee y se escribe.

Otro procedimiento/componente de la actualización de imágenes es la actualización del IMGFS, que es responsable de las actualizaciones a la partición del sistema operativo, según son gestionadas por el sistema de ficheros de imagen (IMGFS). El procedimiento de actualización de reserva es responsable de las actualizaciones a las regiones de radio y a otras regiones reservadas. Las actualizaciones de regiones reservadas funcionan en conjunto directamente con el controlador de bloques cuando se lee y se escribe.

La romimage es un componente compartido (con la Herramienta de escritorio de Imágenes de Disco que proporciona la imagen inicial de la instalación cuando se fabrica un dispositivo y es responsable de la asignación virtual y física (memoria) y de las correcciones de dirección de los módulos. Romimage.dll contiene la jerarquía y la funcionalidad de clases del asignador para crear y gestionar múltiples asignadores, la jerarquía de clases de fichero (usada para almacenar metadatos sobre un fichero o un módulo) y funcionalidad para crear y gestionar listas de ficheros, y funciona para dar soporte al procedimiento de actualización y de construcción. Un componente Patchbin proporciona el procedimiento que aplica una actualización de diferencias binarias para generar un nuevo fichero. Se proporcionan los datos del módulo antiguo y los datos de las diferencias binarias como entradas a este componente, y la salida son los datos para el nuevo módulo. Puede proporcionarse un componente de UI para visualizar los datos apropiados de la interfaz de usuario durante el procedimiento de actualización. Obsérvese que el contenido visual puede ser generado previamente en base a configuraciones del ámbito local específico del sistema operativo.

El procedimiento de actualización NK/XIP puede ser una función llamada por la función principal de la aplicación de actualización, que tomará una lista de los paquetes NK/XIP que deben aplicarse. Una actualización de la partición NK/XIP requiere que la aplicación de actualización rehaga la imagen por completo (efectivamente el procedimiento de la herramienta de imágenes de disco en el dispositivo). Durante una actualización NK/XIP, se mantienen una lista antigua de ficheros y una lista nueva de ficheros. La lista antigua de ficheros se inicializa con los módulos actuales de la partición NK/XIP, y esa información se usa en combinación con los paquetes para crear una nueva lista como resultado final. La nueva lista de ficheros contendrá toda la información necesaria para crear una imagen (cabeceras, datos de secciones, banderas, etcétera) y se pasa esa lista al asignador virtual y físico para rehacer el procedimiento de asignación.

Las FIGURAS 7A y 7B, descritas más arriba, muestran la manera en la que se actualiza la región del núcleo. Resumiendo las etapas de las FIGURAS 7A y 7B, el procedimiento lee toda la región de XIP (u otra) a la RAM y hace de ella una copia de seguridad como un fichero no comprimido en la memoria de usuario, explora los módulos existentes en la región XIP para leer cabeceras y otros metadatos y construye versiones no corregidas en cuanto a dirección de los módulos y los ficheros actualizados. Acto seguido, el procedimiento añade el resto de los módulos y los ficheros que no se modifican a una nueva lista de módulos, lleva a cabo la asignación virtual/física y la corrección de dirección de los módulos y vuelve a escribir la nueva imagen XIP a la memoria flash.

Los procedimientos a prueba de fallos en esta etapa son muy sencillos, dado que no se escribe nada definitivamente a la memoria flash hasta el final del procedimiento. Por lo tanto, si fuera a ocurrir un corte de corriente antes de que se escribiera la nueva imagen, únicamente es preciso rehacer el procedimiento. Si ocurre un corte de corriente escribiendo la nueva imagen, existe la copia de la copia de seguridad de la imagen antigua, y se usa en la recuperación de la imagen (si el fichero de registro especifica que la nueva imagen estaba en vías de ser escrita, se usa la copia de seguridad comprimida para restaurar la copia de la imagen antigua). Un fichero de registro consigna las etapas de la transacción, haciendo sencillo conocer el lugar en el que falló el procedimiento.

REIVINDICACIONES

1. Un procedimiento de actualización a prueba de fallos en un entorno de cálculo, que comprende:

segmentar una imagen (202) de un sistema operativo en porciones separadas actualizables que comprenden una partición (204) del núcleo y una partición (206) del sistema, en el que la partición del sistema comprende una pluralidad de componentes; y

actualizar al menos una porción en aislamiento con respecto a otra partición, en el que la actualización de la partición del núcleo comprende

leer una imagen existente del núcleo e introducirla en una memoria intermedia,

realizar una copia de seguridad de la imagen existente del núcleo comprimiéndola en una memoria (210) de usuario,

añadir nuevos módulos (718) y módulos que no se modifican (724) a una nueva lista de módulos, en la que un módulo designa un único fichero ejecutable,

construir una imagen de remplazo para la partición del núcleo y

escribir la imagen de remplazo encima de la imagen existente del núcleo en la partición del núcleo; y

actualizar la porción del sistema comprende actualizar por separado únicamente componentes de la partición del sistema, en el que un componente designa una colección de módulos.

2. El procedimiento de la reivindicación 1 en el que la actualización de al menos una partición en aislamiento con respecto a otra partición comprende construir una imagen de remplazo para una primera partición y escribir la imagen de remplazo encima de una imagen existente en la primera partición y actualizar una segunda partición actualizando por separado al menos dos componentes en una imagen en la segunda partición.

3. El procedimiento de la reivindicación 2 que, además, comprende realizar una copia de seguridad de la imagen existente en la primera partición a una imagen de copia de seguridad, determinar si se completó con éxito la escritura de la imagen de remplazo y, si no, restaurar la imagen a partir de la imagen de copia de seguridad.

4. El procedimiento de la reivindicación 3 en el que la copia de seguridad de la imagen existente incluye almacenar la imagen de la copia de seguridad de forma comprimida.

5. El procedimiento de la reivindicación 2 en el que actualizar la segunda partición comprende registrar la identidad de cada componente actualizado a un fichero de registro, de modo que, si ocurre un fallo, se puede acceder al fichero de registro para determinar en la actualización de qué componente ocurrió el fallo.

6. El procedimiento de la reivindicación 2 en el que actualizar la segunda partición comprende aplicar (1008) un fichero de diferencias binarias a un componente existente.

7. El procedimiento de la reivindicación 6 en el que aplicar un fichero de diferencias binarias a un componente existente comprende aplicar un subconjunto del fichero de diferencias binarias a un subconjunto del componente existente.

8. El procedimiento de la reivindicación 7 en el que el subconjunto comprende un bloque de datos que tiene un tamaño definido.

9. El procedimiento de la reivindicación 2 en el que actualizar la segunda partición comprende simular un proceso de actualización y determinar que la simulación tuvo éxito antes de escribir definitivamente las actualizaciones a la segunda partición.

10. El procedimiento de la reivindicación 1 que, además, comprende:

determinar si debe arrancarse el dispositivo en un modo de sistema operativo o en un modo de actualización; y

cuando se arranca el dispositivo (120) de cálculo al modo de actualización, llevar a cabo al menos una actualización de una imagen en la memoria del dispositivo mientras se registra un estado de la actualización para permitir la recuperación de cualquier fallo que pueda ocurrir durante cada actualización.

11. El procedimiento de la reivindicación 10 en el que determinar si debe arrancarse el dispositivo (120) en un modo de sistema operativo o en un modo de actualización comprende comprobar una bandera que se fija antes de un rearranque.

12. El procedimiento de la reivindicación 10 en el que la imagen (202) es dividida en al menos dos particiones y en el que una partición seleccionada es actualizada construyendo una imagen de remplazo para la partición y escribiendo la imagen de remplazo encima de una imagen existente en la partición.

13. El procedimiento de la reivindicación 12 que, además, comprende la realización de una copia de seguridad de la imagen existente en la partición creando una imagen de copia de seguridad, determinar si la escritura de la imagen de remplazo se completó con éxito y, si no, restaurar la imagen existente a partir de la imagen de copia de seguridad.

14. El procedimiento de la reivindicación 13 en el que la copia de seguridad de la imagen existente incluye almacenar la imagen de copia de seguridad de forma comprimida.

15. El procedimiento de la reivindicación 10 en el que la imagen (202) está dividida en al menos dos particiones y en el que se actualiza una partición seleccionada actualizando por separado al menos dos componentes en la imagen.

16. El procedimiento de la reivindicación 15 en el que una de las particiones es una partición de sistema y en el que la actualización por separado de al menos dos componentes en la imagen comprende la escritura de ficheros del sistema operativo en la partición.

17. El procedimiento de la reivindicación 10 en el que la realización de la al menos una actualización comprende la aplicación (1008) de un fichero de diferencias binarias a un componente existente.

18. El procedimiento de la reivindicación 17 en el que la aplicación (1008) de un fichero de diferencias binarias a un componente existente comprende la aplicación de un subconjunto del fichero de diferencias binarias a un subconjunto del componente existente.

19. El procedimiento de la reivindicación 18 en el que el subconjunto comprende un bloque de datos que tiene un tamaño definido.

20. El procedimiento de la reivindicación 18 en el que la realización de al menos una actualización a una imagen (202) comprende simular un proceso de actualización y determinar que la simulación tuvo éxito antes de escribir definitivamente la actualización.

21. El procedimiento de la reivindicación 10 en el que las actualizaciones en un conjunto de actualizaciones se llevan a cabo individualmente en ficheros en paquetes y ficheros, y la información de paquetes y ficheros individuales se registran, de modo que, tras cualquier fallo, el acceso al fichero de registro determina en qué paquete y en qué fichero del paquete ocurrió el fallo.

22. Uno o más medios legibles por ordenador que tienen instrucciones ejecutables por ordenador que, cuando se ejecutan, llevan a cabo el procedimiento según una de las reivindicaciones 1 a 21.

23. Un sistema en un dispositivo (120) de cálculo que comprende:

un mecanismo para segmentar una imagen (202) de un sistema operativo en porciones separadas actualizables que comprenden una partición (204) del núcleo y una partición (206) del sistema, en el que la partición del sistema comprende una pluralidad de componentes; y

un cargador de actualizaciones para actualizar por separado cada partición

leyendo una imagen existente del núcleo e introduciéndola en una memoria intermedia,

realizando una copia de seguridad de la imagen existente del núcleo comprimiéndola en una memoria (210) de usuario,

añadiendo nuevos módulos (718) y módulos que no se modifican (724) a una nueva lista de módulos durante la actualización del núcleo, en la que un módulo designa un único fichero ejecutable,

construyendo una imagen de remplazo para la partición del núcleo y

escribiendo la imagen de remplazo encima de la imagen existente del núcleo en la partición del núcleo; y

actualizar la porción del sistema actualizando por separado únicamente componentes de la partición del sistema, en el que un componente designa una colección de módulos.

24. El sistema de la reivindicación 23 en el que un mecanismo de arranque hace que arranque el cargador de actualizaciones tras detectarse una actualización pendiente, comprendiendo el cargador de actualizaciones la única entidad en el código del dispositivo que tiene acceso de escritura a la memoria protegida del dispositivo,

conteniendo la memoria protegida al menos dos particiones y actualizando el cargador de actualizaciones cada partición por separado.

25. El sistema de la reivindicación 23 que, además, comprende un procedimiento de validación en el que, antes del arranque del cargador de actualizaciones, el procedimiento de validación valida al menos un paquete de actualizaciones y, si al menos un paquete de actualizaciones es válido, el procedimiento de validación pone a cada paquete válido en una cola de espera para la actualización y establece un mecanismo para que el mecanismo de arranque detecte la actualización pendiente.

26. El sistema de la reivindicación 25 en el que el procedimiento de validación valida al menos un paquete de actualización comprobando que cada paquete esté debidamente firmado.

27. El sistema de la reivindicación 25 en el que el procedimiento de validación valida al menos un paquete de actualización comprobando que cada paquete esté debidamente construido.

28. El sistema de la reivindicación 23 en el que un mecanismo de arranque arranca el dispositivo (120) en un modo de actualizaciones en base al valor de una bandera que se estableció antes de un rearranque.

29. El sistema de la reivindicación 23 en el que una memoria protegida del dispositivo está dividida en una pluralidad de particiones y en el que una partición seleccionada es actualizada por medio de un componente de actualizaciones construyendo una imagen de remplazo para la partición y escribiendo la imagen de remplazo encima de una imagen existente en la partición seleccionada.

30. El sistema de la reivindicación 29 en el que el componente de actualizaciones realiza una copia de seguridad de una imagen existente en la partición seleccionada en una imagen de copia de seguridad para restaurarla en la partición seleccionada si la imagen de remplazo no se escribe con éxito a la partición seleccionada.

31. El sistema de la reivindicación 30 en el que el componente de actualizaciones realiza una copia de seguridad de una imagen existente de forma comprimida.

32. El sistema de la reivindicación 29 en el que la memoria protegida del dispositivo está dividida en una pluralidad de particiones y en el que una partición seleccionada es actualizada por el componente de actualizaciones actualizando por separado al menos dos componentes en la partición seleccionada.

33. El sistema de la reivindicación 32 en el que el cargador de actualizaciones actualiza al menos uno de los componentes en la partición seleccionada aplicando (1008) un fichero de diferencias binarias a un componente existente.

34. El sistema de la reivindicación 33 en el que el cargador de actualizaciones aplica el fichero de diferencias binarias aplicando un subconjunto del fichero de diferencias binarias a un subconjunto del componente existente.

35. El sistema de la reivindicación 32 en el que el cargador de actualizaciones simula al menos una actualización antes de escribir definitivamente la actualización.

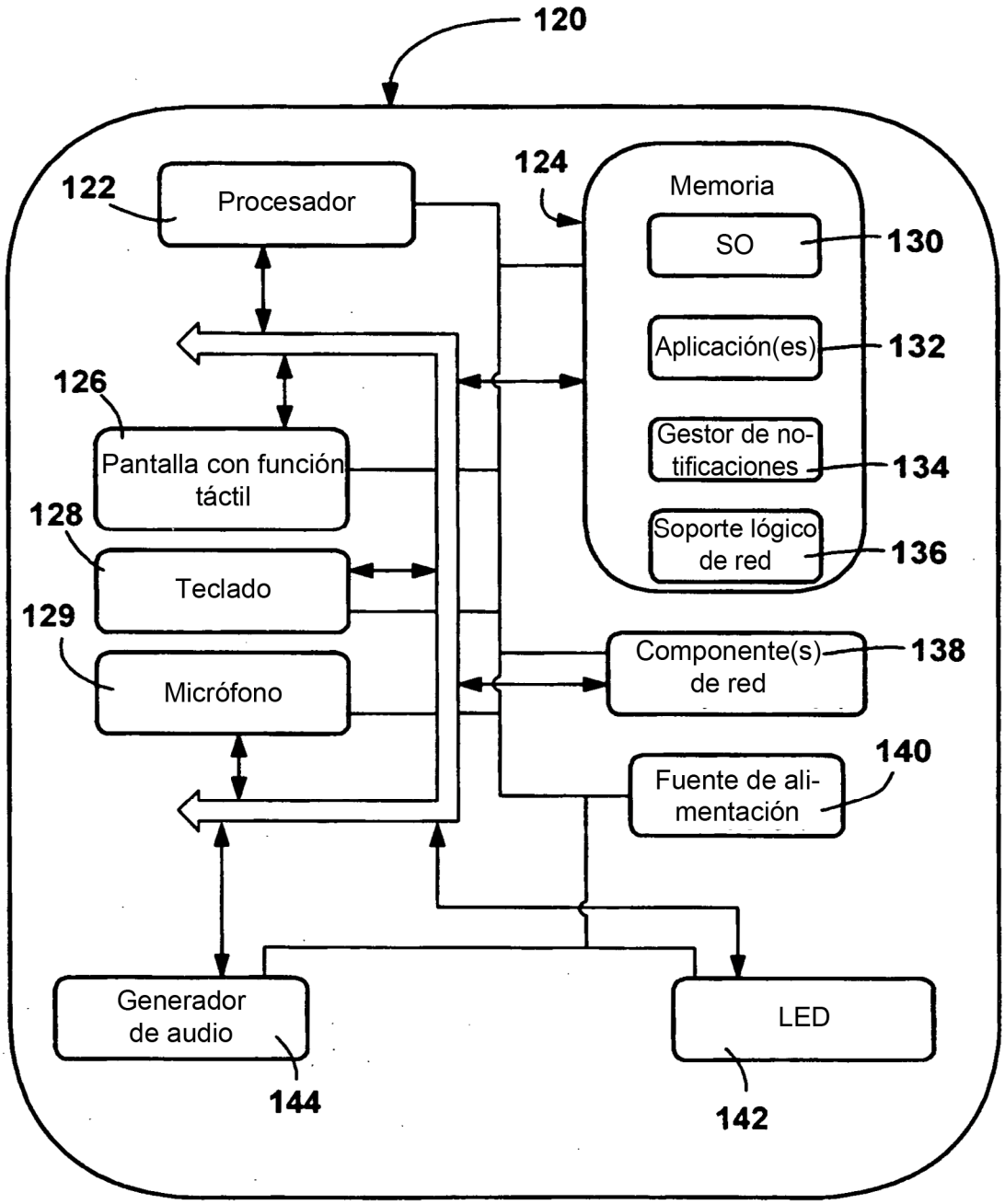

120
122
Procesador
124
Memoria
SO
130
Aplicación(es)
132
126
Pantalla con función táctil
Gestor de notificaciones
134
128
Teclado
Soporte lógico de red
136
129
Micrófono
Componente(s) de red
138
140
Fuente de alimentación
Generador de audio
144
LED
142

FIG. 1

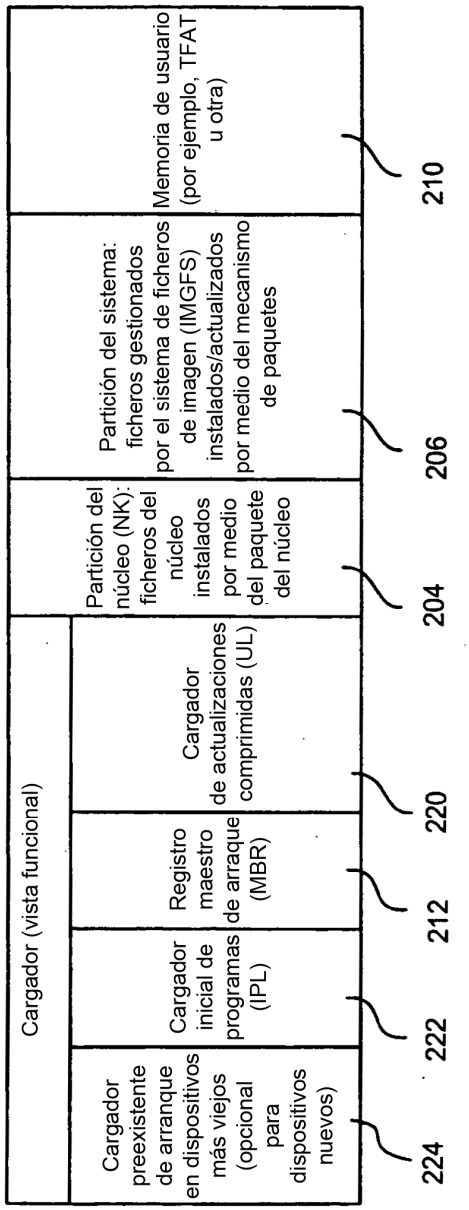

Cargador (vista funcional)
Cargador preexistente de arranque en dispositivos más viejos (opcional para dispositivos nuevos)
Cargador inicial de programas (IPL)
Registro maestro de arraque (MBR)
Cargador de actualizaciones comprimidas (UL)
Partición del núcleo (NK): ficheros del núcleo instalados por medio del paquete del núcleo
Partición del sistema: ficheros gestionados por el sistema de ficheros de imagen (IMGFS) instalados/actualizados por medio del mecanismo de paquetes
Memoria de usuario (por ejemplo, TFAT u otra)
224
222
212
220
204
206
210
202

FIG. 2

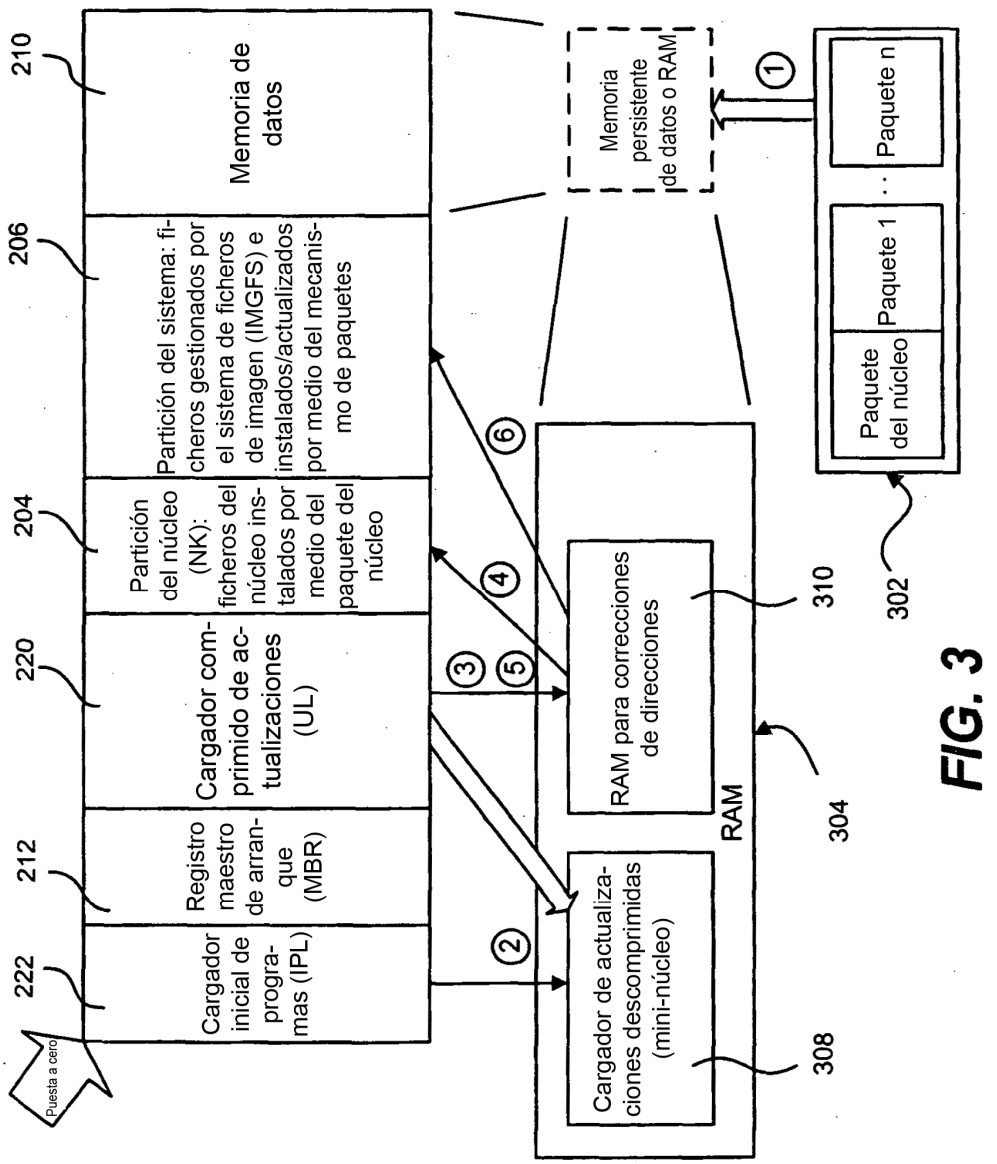

Puesta a cero
Cargador inicial de programas (IPL)
Registro maestro de arranque (MBR)
Cargador comprimido de actualizaciones (UL)
Partición del núcleo (NK): ficheros del núcleo instalados por medio del paquete del núcleo
Partición del sistema: ficheros gestionados por el sistema de ficheros de imagen (IMGFS) e instalados/actualizados por medio del mecanismo de paquetes
Memoria de datos
222
212
220
204
206
210
Cargador de actualizaciones descomprimidas (mini-núcleo)
RAM para correcciones de direcciones
RAM
308
304
310
Memoria persistente de datos o RAM
Paquete del núcleo
Paquete 1
Paquete n
302
1
2
3
4
5
6

FIG. 3

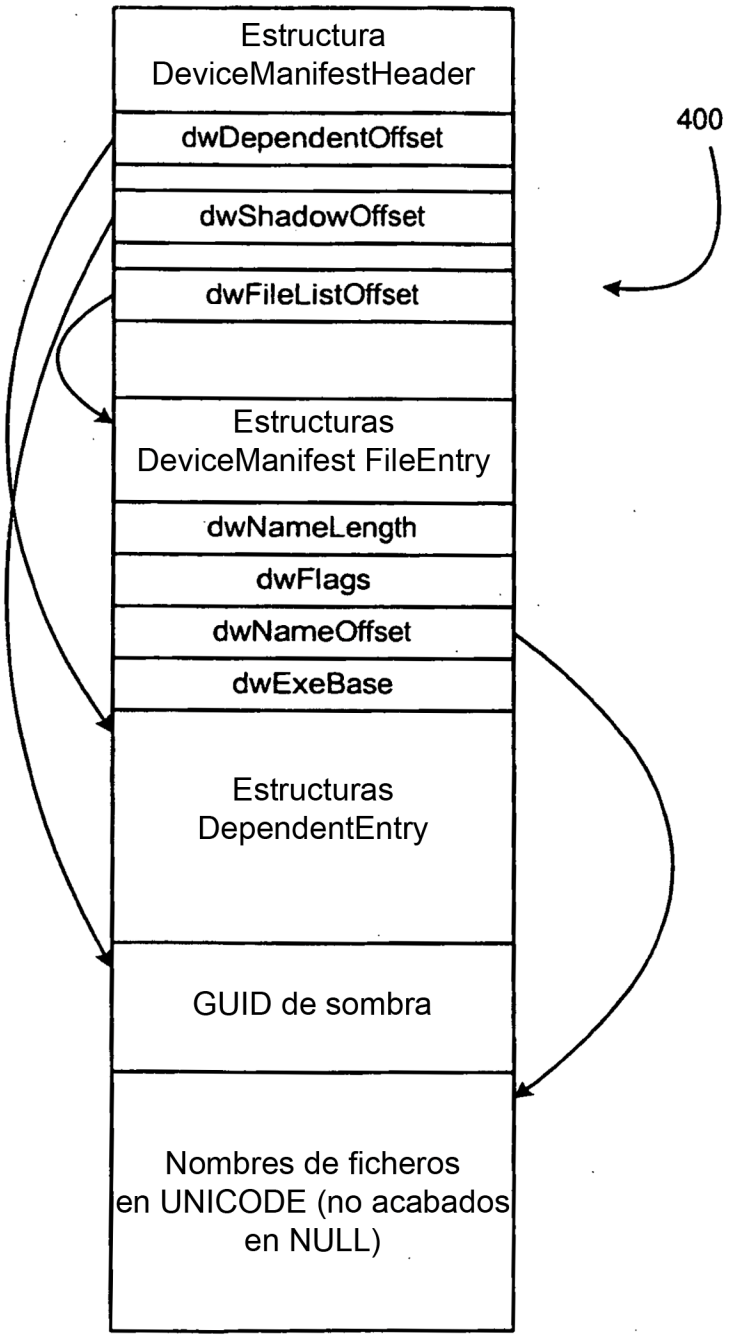

Estructura
DeviceManifestHeader
dwDependentOffset
dwShadowOffset
dwFileListOffset
Estructuras
DeviceManifest FileEntry
dwNameLength
dwFlags
dwNameOffset
dwExeBase
Estructuras
DependentEntry
GUID de sombra
Nombres de ficheros
en UNICODE (no acabados
en NULL)
400

FIG. 4

FIG. 5A

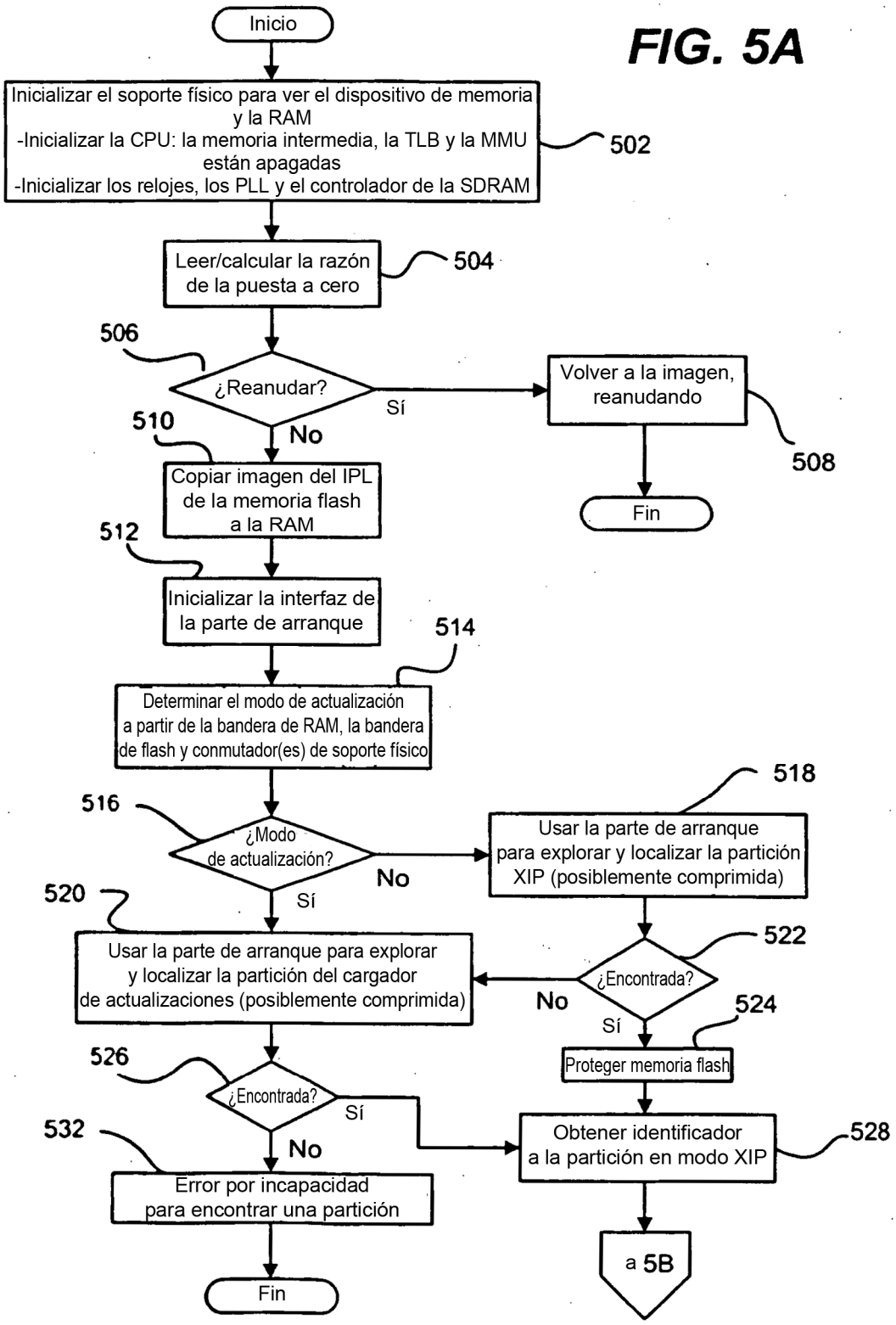
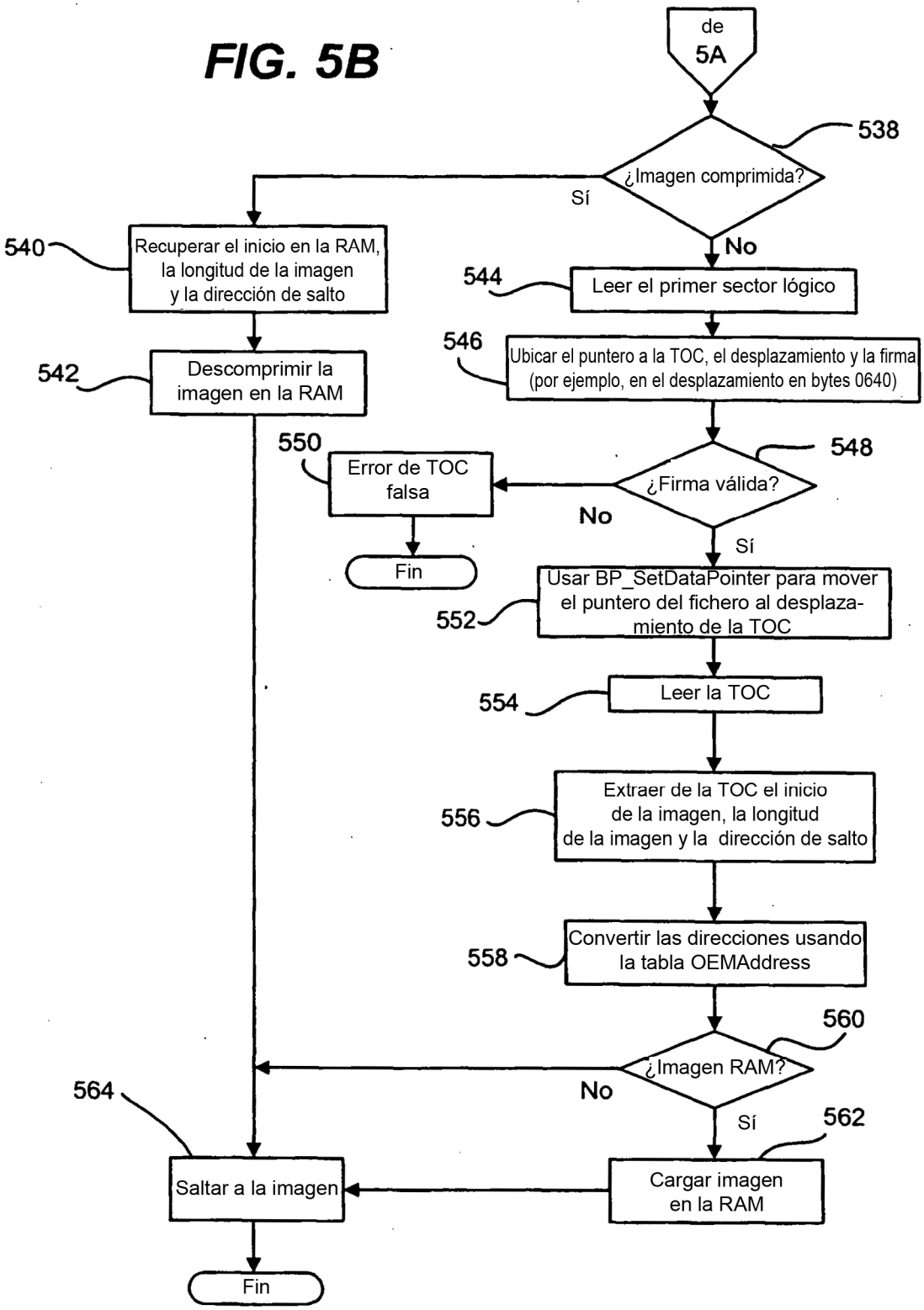

Inicio
Inicializar el soporte físico para ver el dispositivo de memoria y la RAM
-Inicializar la CPU: la memoria intermedia, la TLB y la MMU están apagadas
-Inicializar los relojes, los PLL y el controlador de la SDRAM
502
Leer/calcular la razón de la puesta a cero
504
506
¿Reanudar?
Sí
Volver a la imagen, reanudando
508
Fin
510
No
Copiar imagen del IPL de la memoria flash a la RAM
512
Inicializar la interfaz de la parte de arranque
514
Determinar el modo de actualización a partir de la bandera de RAM, la bandera de flash y conmutador(es) de soporte físico
516
¿Modo de actualización?
No
518
Usar la parte de arranque para explorar y localizar la partición XIP (posiblemente comprimida)
Sí
520
Usar la parte de arranque para explorar y localizar la partición del cargador de actualizaciones (posiblemente comprimida)
522
¿Encontrada?
No
524
Sí
Proteger memoria flash
526
¿Encontrada?
Sí
528
Obtener identificador a la partición en modo XIP
532
No
Error por incapacidad para encontrar una partición
Fin
a 5B

FIG. 5B


de 5A
538
¿Imagen comprimida?
Sí
No
540
Recuperar el inicio en la RAM, la longitud de la imagen y la dirección de salto
544
Leer el primer sector lógico
546
Ubicar el puntero a la TOC, el desplazamiento y la firma (por ejemplo, en el desplazamiento en bytes 0640)
542
Descomprimir la imagen en la RAM
550
Error de TOC falsa
548
¿Firma válida?
No
Sí
Fin
Usar BP_SetDataPointer para mover el puntero del fichero al desplaza-miento de la TOC
552
554
Leer la TOC
556
Extraer de la TOC el inicio de la imagen, la longitud de la imagen y la dirección de salto
558
Convertir las direcciones usando la tabla OEMAddress
560
¿Imagen RAM?
No
Sí
562
564
Saltar a la imagen
Cargar imagen en la RAM
Fin

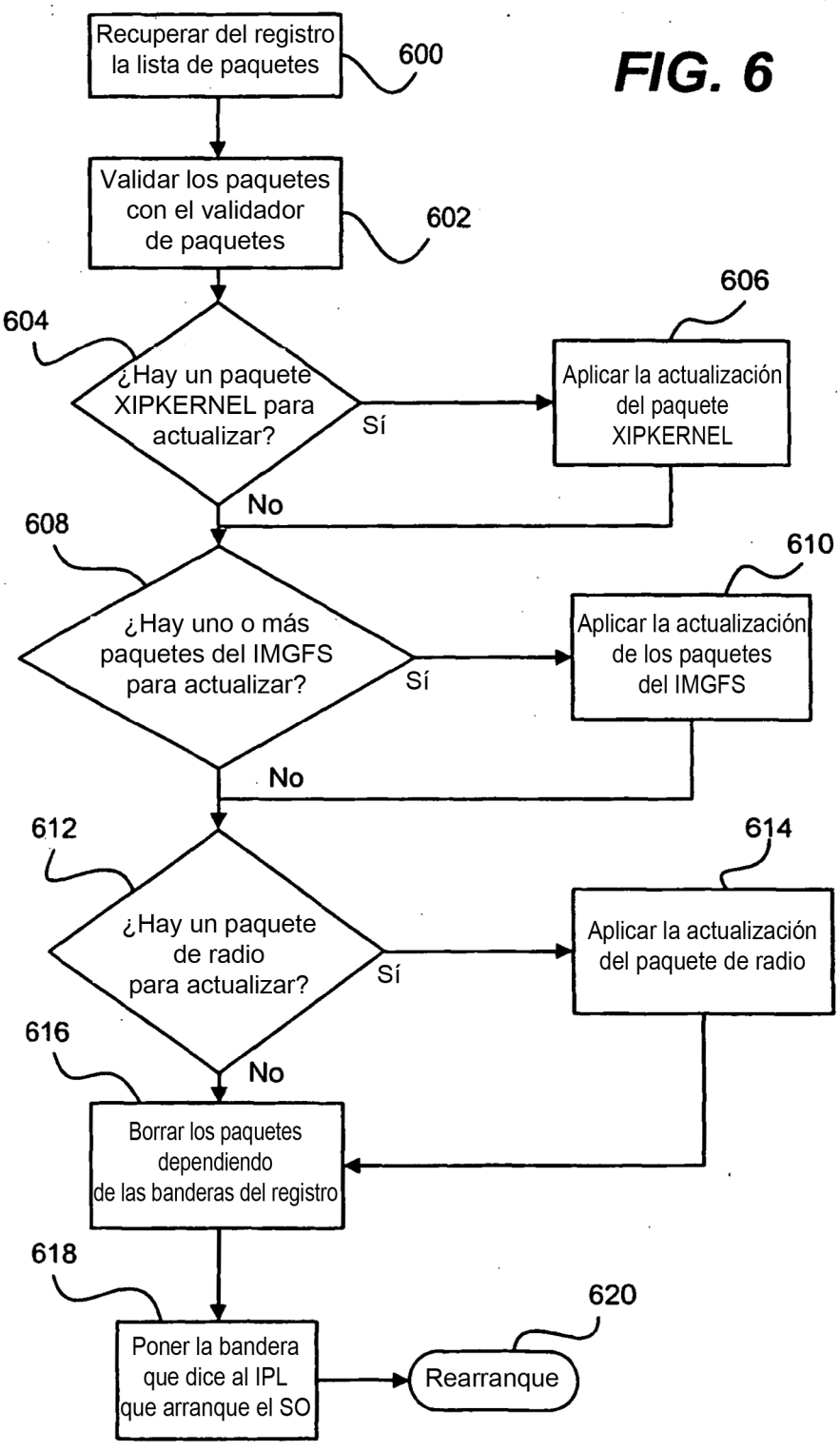

Recuperar del registro la lista de paquetes
600
Validar los paquetes con el validador de paquetes
602
604
¿Hay un paquete XIPKERNEL para actualizar?
Sí
606
Aplicar la actualización del paquete XIPKERNEL
No
608
¿Hay uno o más paquetes del IMGFS para actualizar?
Sí
610
Aplicar la actualización de los paquetes del IMGFS
No
612
¿Hay un paquete de radio para actualizar?
Sí
614
Aplicar la actualización del paquete de radio
No
616
Borrar los paquetes dependiendo de las banderas del registro
618
Poner la bandera que dice al IPL que arranque el SO
620
Rearranque

FIG. 6

FIG. 7A

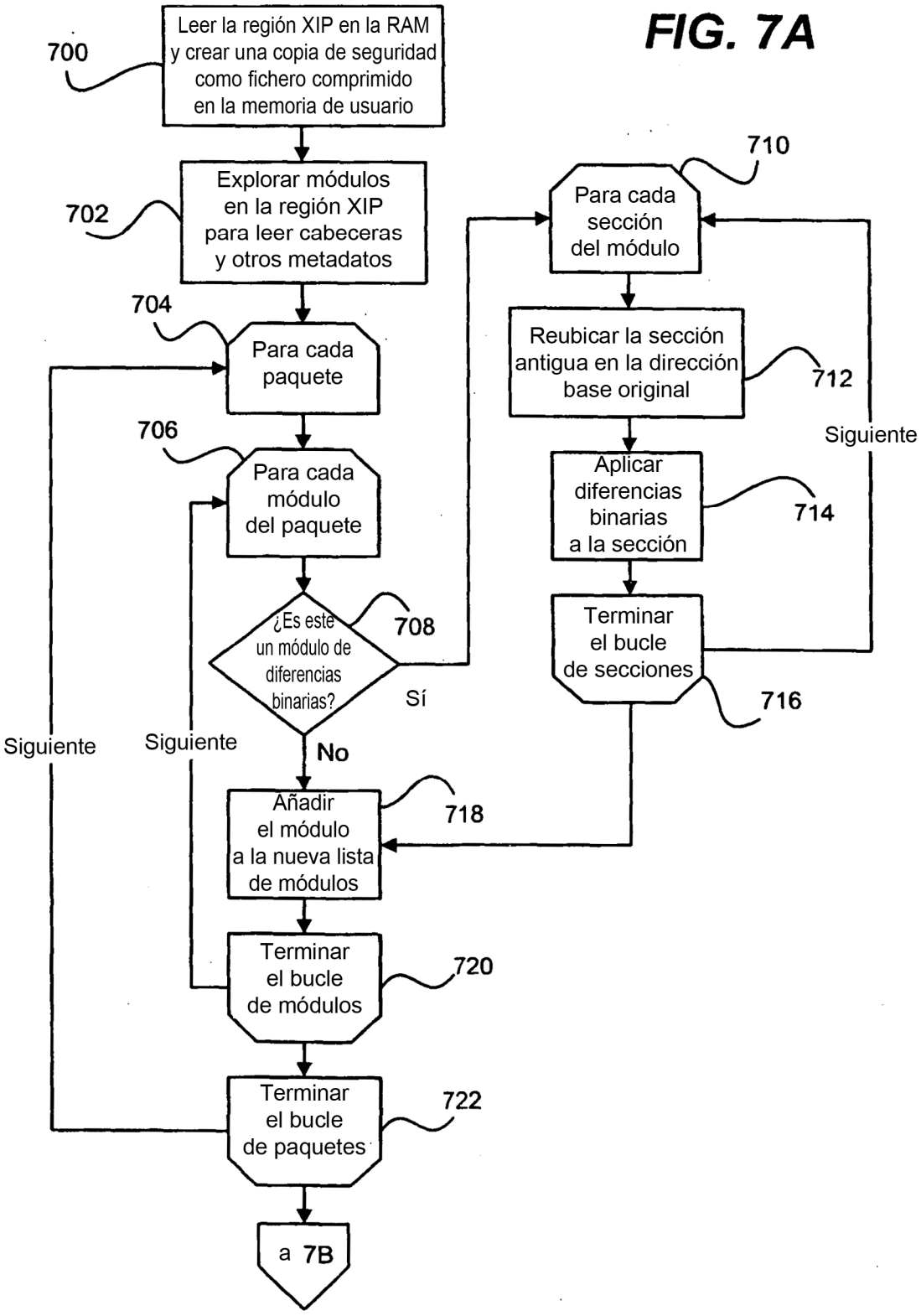
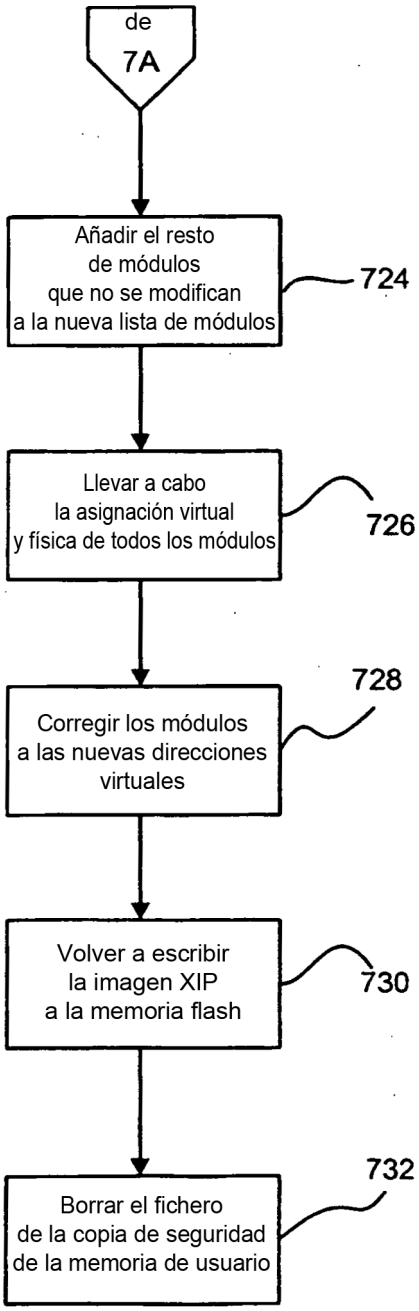

Leer la región XIP en la RAM y crear una copia de seguridad como fichero comprimido en la memoria de usuario
700
Explorar módulos en la región XIP para leer cabeceras y otros metadatos
702
Para cada paquete
704
Para cada módulo del paquete
706
¿Es este un módulo de diferencias binarias?
708
Sí
No
Siguiente
Siguiente
Añadir el módulo a la nueva lista de módulos
718
Terminar el bucle de módulos
720
Terminar el bucle de paquetes
722
a 7B
Para cada sección del módulo
710
Reubicar la sección antigua en la dirección base original
712
Aplicar diferencias binarias a la sección
714
Terminar el bucle de secciones
716
Siguiente

FIG. 7B


de
7A
Añadir el resto
de módulos
que no se modifican
a la nueva lista de módulos
724
Llevar a cabo
la asignación virtual
y física de todos los módulos
726
728
Corregir los módulos
a las nuevas direcciones
virtuales
Volver a escribir
la imagen XIP
a la memoria flash
730
732
Borrar el fichero
de la copia de seguridad
de la memoria de usuario

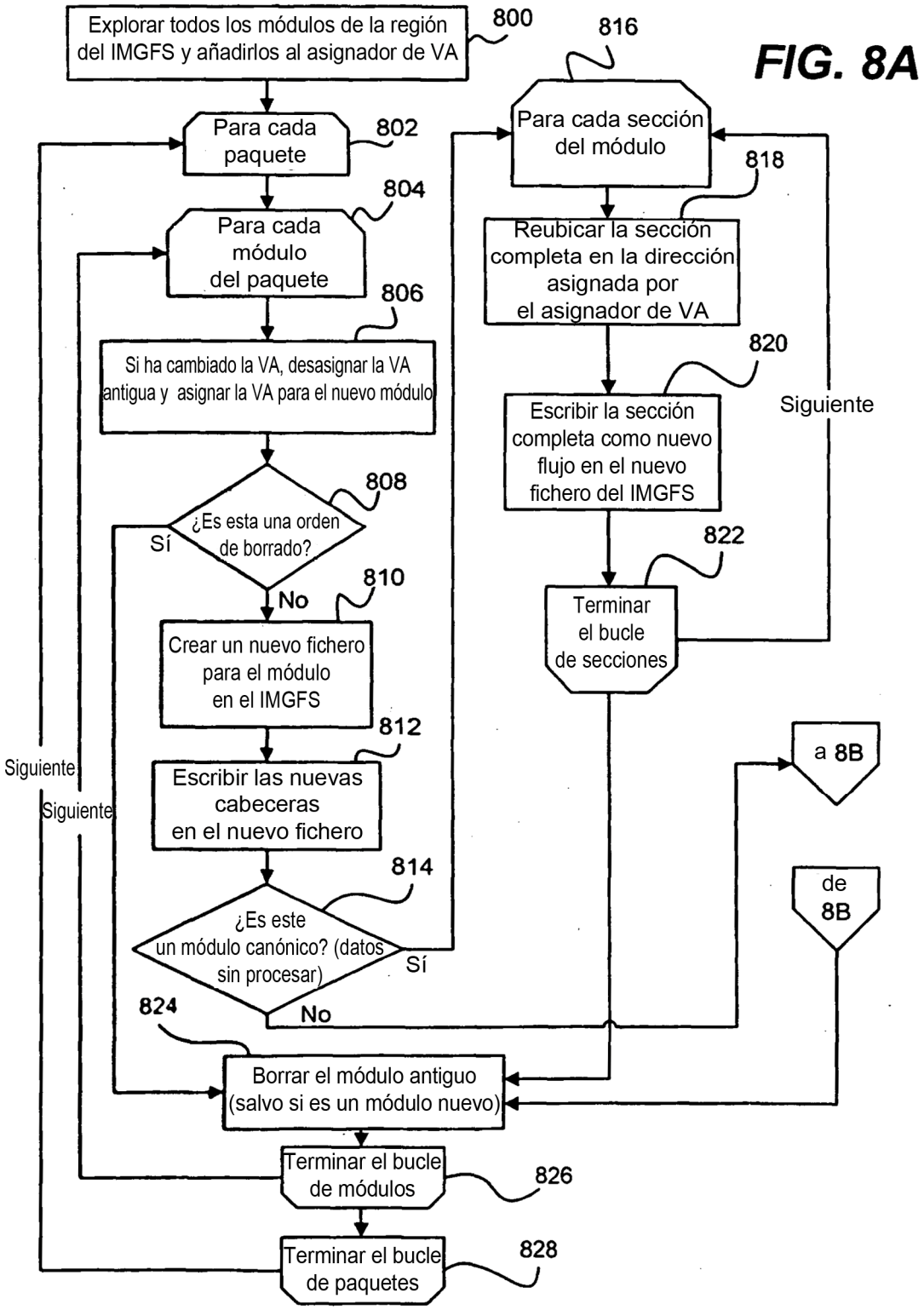

Explorar todos los módulos de la región del IMGFS y añadirlos al asignador de VA
800
Para cada paquete
802
Para cada módulo del paquete
804
Si ha cambiado la VA, desasignar la VA antigua y asignar la VA para el nuevo módulo
806
¿Es esta una orden de borrado?
808
Sí
No
810
Crear un nuevo fichero para el módulo en el IMGFS
812
Escribir las nuevas cabeceras en el nuevo fichero
814
¿Es este un módulo canónico? (datos sin procesar)
Sí
No
824
Borrar el módulo antiguo (salvo si es un módulo nuevo)
Terminar el bucle de módulos
826
Terminar el bucle de paquetes
828
Siguiente
Siguiente
816
Para cada sección del módulo
818
Reubicar la sección completa en la dirección asignada por el asignador de VA
820
Escribir la sección completa como nuevo flujo en el nuevo fichero del IMGFS
822
Terminar el bucle de secciones
Siguiente
a 8B
de 8B

FIG. 8A

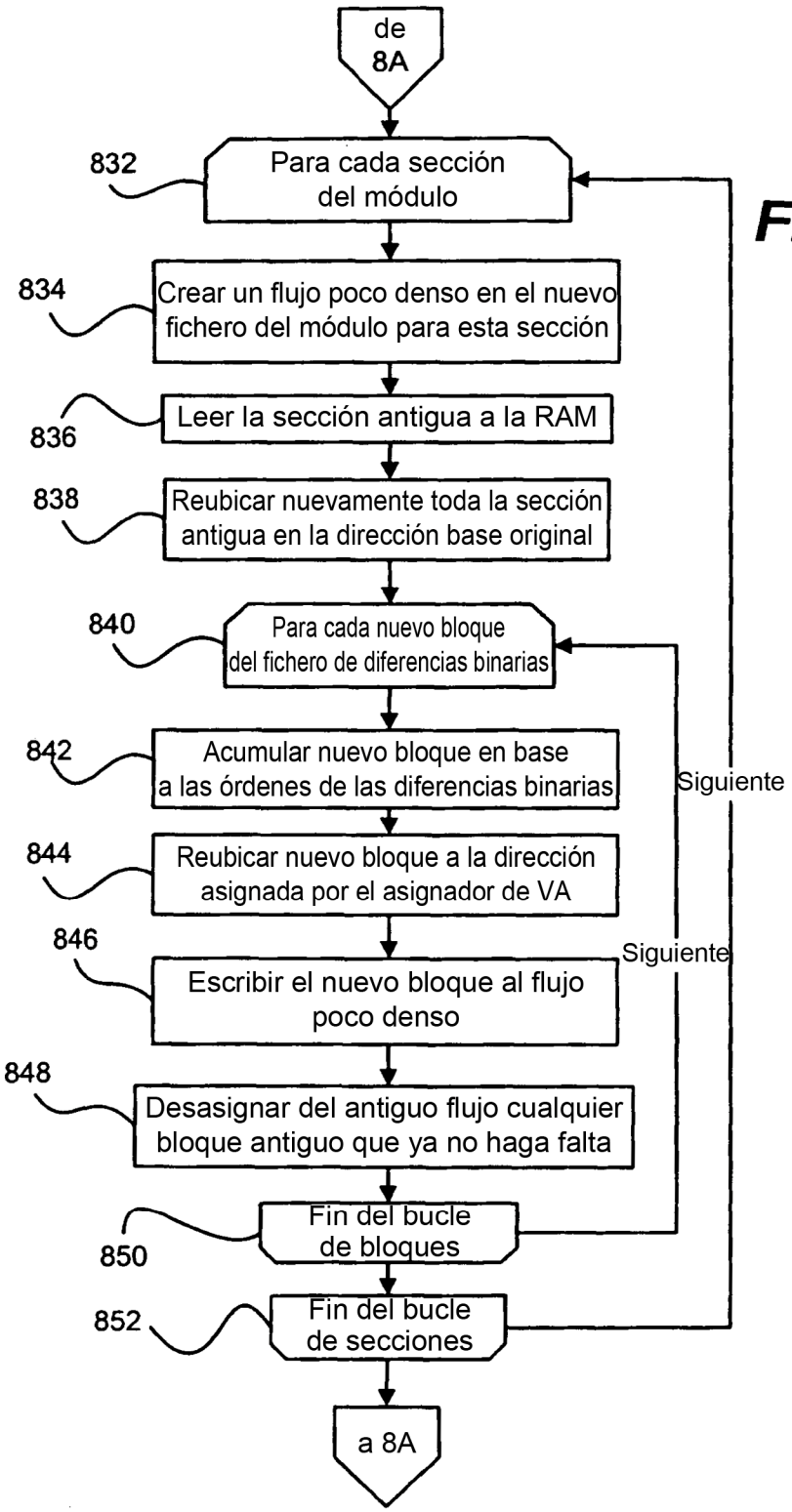

de
8A
832
Para cada sección
del módulo
834
Crear un flujo poco denso en el nuevo
fichero del módulo para esta sección
836
Leer la sección antigua a la RAM
838
Reubicar nuevamente toda la sección
antigua en la dirección base original
840
Para cada nuevo bloque
del fichero de diferencias binarias
842
Acumular nuevo bloque en base
a las órdenes de las diferencias binarias
844
Reubicar nuevo bloque a la dirección
asignada por el asignador de VA
846
Escribir el nuevo bloque al flujo
poco denso
848
Desasignar del antiguo flujo cualquier
bloque antiguo que ya no haga falta
850
Fin del bucle
de bloques
852
Fin del bucle
de secciones
a 8A
Siguiente
Siguiente

FIG. 8B

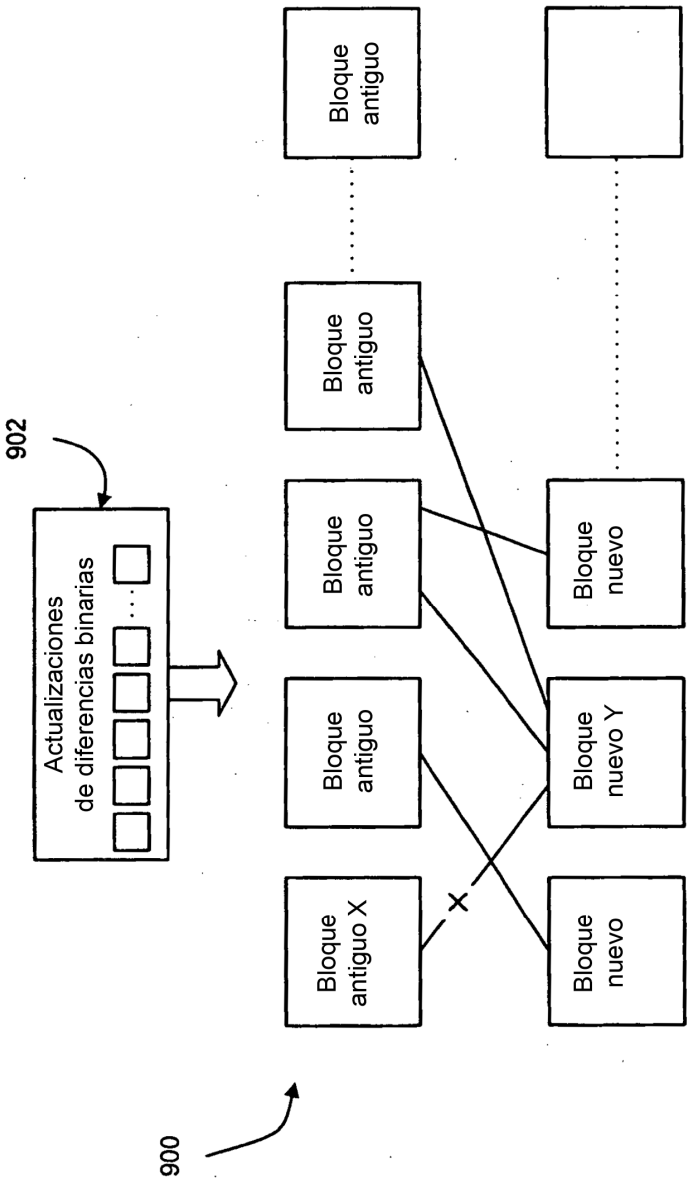
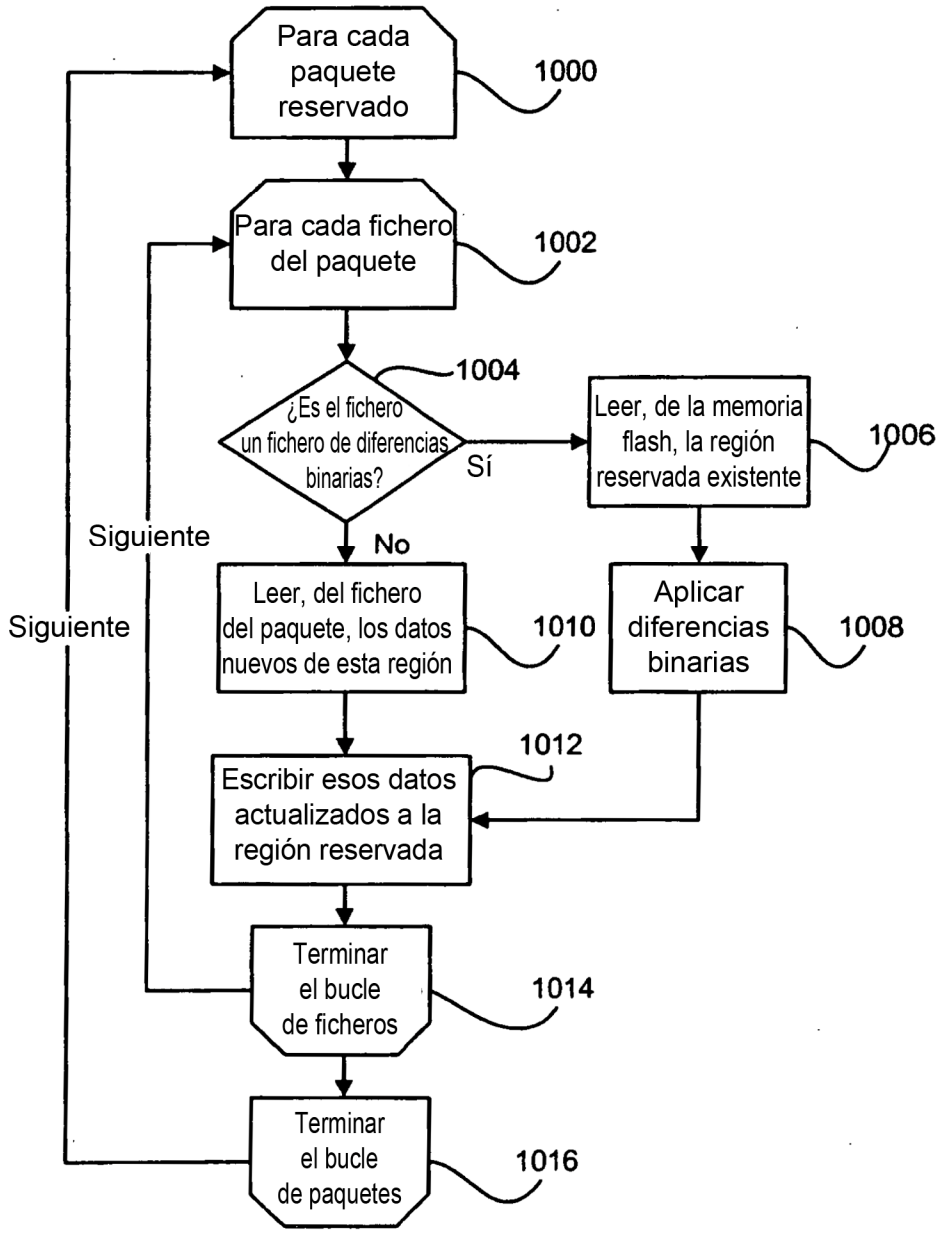

902
Actualizaciones de diferencias binarias
900
Bloque antiguo X
Bloque antiguo
Bloque antiguo
Bloque antiguo
Bloque antiguo
Bloque nuevo
Bloque nuevo Y
Bloque nuevo
X

FIG. 9

FIG. 10


Para cada paquete reservado
1000
Para cada fichero del paquete
1002
1004
¿Es el fichero un fichero de diferencias binarias?
Sí
Leer, de la memoria flash, la región reservada existente
1006
No
Siguiente
Siguiente
Leer, del fichero del paquete, los datos nuevos de esta región
1010
Aplicar diferencias binarias
1008
1012
Escribir esos datos actualizados a la región reservada
Terminar el bucle de ficheros
1014
Terminar el bucle de paquetes
1016