(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 642 320**

(51) Int. CI.:

G06F 9/44 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **17.01.2006 PCT/EP2006/000351**

(87) Fecha y número de publicación internacional: **27.07.2006 WO06077068**

(96) Fecha de presentación y número de la solicitud europea: **17.01.2006 E 06706256 (2)**

(97) Fecha y número de publicación de la concesión europea: **26.07.2017 EP 1844394**

(54) Título: **Cargador de arranque de sistema operativo fácil de usar**

(30) Prioridad:

22.01.2005 US 40798

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
16.11.2017

(73) Titular/es:

TELEFONAKTIEBOLAGET LM ERICSSON (PUBL) (100.0%)
164 83 Stockholm, SE

(72) Inventor/es:

SVENSSON, MATS;
ROSENBERG, MICHAEL;
BAUER, NICLAS y
AULIN, PETER

(74) Agente/Representante:

ELZABURU, S.L.P

Aviso: En el plazo de nueve meses a contar desde la fecha de publicación en el Boletín Europeo de Patentes, de la mención de concesión de la patente europea, cualquier persona podrá oponerse ante la Oficina Europea de Patentes a la patente concedida. La oposición deberá formularse por escrito y estar motivada; sólo se considerará como formulada una vez que se haya realizado el pago de la tasa de oposición (art. 99.1 del Convenio sobre Concesión de Patentes Europeas).

DESCRIPCIÓN

Cargador de arranque de sistema operativo fácil de usar

Esta invención se refiere a la inicialización de sistemas electrónicos que tienen múltiples procesadores programables.

El proceso de iniciar, o arrancar, un sistema electrónico que tiene un procesador programable conectado a uno o más dispositivos de memoria para almacenar instrucciones de programa, o código, y datos no es tan simple como podría parecer a primera vista. Una parte importante de la razón para esto es la necesidad de que el procesador comience la operación en un estado bien definido.

Los modos tradicionales para cargar código de programa y datos en un sistema básico son o bien "empujando" el código y datos a la memoria de acceso aleatorio (RAM) del sistema directamente o bien utilizando un cargador de arranque. El cargador de arranque, que es llamado algunas veces un cargador de arranque o un cargador de inicio, es un conjunto de instrucciones (es decir, código de programa, algunas veces llamado "código de arranque") que puede ser bien "empujado" a la RAM del sistema o bien cargado en la RAM desde una memoria no volátil, tal como la memoria de sólo lectura (ROM). En su ejecución por el procesador, el cargador de arranque entonces "arrastra" el resto del código y los datos e inicia el sistema.

Ejemplos de mecanismos anteriores para iniciar sistemas del procesador, que incluye cargadores de arranque, son las Patentes de los EE.UU. Nº 5.652.886 de Tulpule y col. y Nº 6.490.722 de Barton y col. y la Publicación de Solicitud de Patente de los EE.UU. Nº US 2002/0138156 A1 de Wong y col. Barton y col., por ejemplo, describen un cargador de arranque de dos etapas en el cual la segunda etapa encuentra, verifica, y carga el sistema operativo. En Wong y col., un sistema de multiprocesador utiliza un procesador maestro acoplado a una ROM para transferir el código de arranque a los procesadores esclavos, con controladores de memoria en los procesadores esclavos que rechazan las solicitudes de acceso a la memoria hasta que el código de arranque ha sido transferido a sus RAM.

Como es indicado por Barton y col. y Wong y col., por ejemplo, poner en marcha un sistema multiprocesador, que puede ser considerado generalmente como que tiene un procesador maestro o anfitrión, es decir, el sistema que ordena el arranque, y uno o más procesadores esclavos o clientes, es decir, el sistema que ha de ser arrancado, es incluso más complicado que poner en marcha un sistema de un solo procesador.

La Patente de los EE.UU. Nº 5.155.833 de Cullison y col. describe un multiprocesador maestro-esclavo en el que el procesador esclavo tiene una agrupación de RAM que sirve en el momento de la inicialización como la memoria de arranque del procesador esclavo y durante el funcionamiento normal como la memoria caché del procesador esclavo. El procesador maestro escribe el programa de arranque del procesador esclavo en la agrupación de RAM, por ejemplo, después de un reinicio del sistema.

Las ventajas del método de "empuje" son que no requiere código para ejecutar en el esclavo durante el arranque y que solamente la sincronización requerida es para mantener el esclavo en un estado de reinicio y liberarlo cuando se finaliza la carga. Sin embargo, el método de "empuje" funciona solamente cuando la memoria o memorias del esclavo son visibles para el anfitrión. Esta visibilidad puede ser implementada de varias maneras. Por ejemplo, una memoria puede ser visible en la dirección y los buses de datos tanto de los procesadores anfitriones como de los esclavos o las transferencias de acceso directo memoria (DMA) pueden ser permitidos desde la memoria o memorias del anfitrión a la memoria o memorias del esclavo.

Cuando la memoria del esclavo que ha de ser cargada es invisible para el anfitrión, el método de "empuje" no puede ser utilizado. En esa situación, ha de utilizarse alguna forma de carga de arranque. Como se ha observado anteriormente, la técnica del cargador de arranque requiere, o bien que el código de arranque pueda ser empujado al esclavo (lo que en este caso no es posible), o bien que el esclavo pueda cargar el código desde una memoria no volátil. El cargador de arranque inicia a continuación una transferencia de código desde el anfitrión al esclavo y finaliza cargando la memoria.

Son posibles los sistemas multiprocesador en los cuales parte o toda la memoria del esclavo no está visible para un anfitrión. En tales sistemas, puede ser ventajoso aprovecharse de las estructuras de software bien establecidas para la carga y comunicación entre procesadores, que renderizan cargadores de arranque tradicionales indeseables. Además, un cargador de arranque puede entrar en conflicto con el sistema operativo, que se puede decir que quiere tener el control sobre todo el sistema y toda la memoria.

Entre los problemas a los que se hace frente al integrar un cargador de arranque con un sistema operativo (OS) están asegurar que el código que no está aún cargado no es ejecutado, cargar de manera eficiente el código a una memoria o memorias invisibles para el anfitrión, y sincronizar con el anfitrión la carga y el arranque del esclavo o esclavos. Además, es necesario determinar qué partes del sistema han de ser cargadas en la memoria visible tanto para los procesadores anfitriones como para los esclavos y cómo la imagen binaria que ha de ser cargada debería estar dispuesta para que el cargador de arranque funcione junto con el OS. Otro problema que puede ser importante

es la integración del cargador de arranque y el OS, ya que una estructura ya establecida para la comunicación entre el anfitrión y el esclavo puede ser a continuación utilizada durante la carga. Tal estructura incluiría típicamente uno o más primitivos para la comunicación que dependen de las característica del OS.

Compendio

Esta invención proporciona, en un aspecto, un método para cargar información que incluye un cargador de arranque en un procesador esclavo en un sistema multiprocesador que incluye un procesador maestro y el procesador esclavo. El método incluye los pasos de reiniciar el procesador esclavo y mantener el procesador esclavo en un estado de reinicio; empujar una primera parte de la información a una primera memoria que es accesible mediante los procesadores maestro y esclavo, incluyendo la primera parte de la información el cargador de arranque; arrancar el procesador esclavo; poner en marcha un sistema operativo en el procesador esclavo, incluyendo el bloqueo de la planificación de procesos que tienen código de programa ubicado en una segunda memoria que es accesible mediante el procesador esclavo e inaccesible mediante el procesador maestro; reservar un área de almacenamiento intermedio en la primera memoria; enviar al procesador maestro la información sobre una ubicación y tamaño del área de almacenamiento intermedio con una segunda parte de la información que ha de ser cargada en la segunda memoria; enviar un primer mensaje al procesador esclavo que indica que el área de almacenamiento intermedio ha sido cargada y se ha finalizado la carga o si se va a cargar más información; copiar la información en el área de almacenamiento intermedio a la segunda memoria; enviar un segundo mensaje al procesador maestro que indica que la información en el área de almacenamiento intermedio ha sido copiada; y liberar el bloqueo de la planificación de procesos que tienen código de programa ubicado en la segunda memoria.

En otro aspecto de la invención, un sistema multiprocesador incluye un procesador anfitrión, al menos un procesador cliente, una primera memoria accesible mediante los procesadores anfitrión y cliente, una segunda memoria accesible mediante el procesador cliente y no accesible mediante el procesador anfitrión, y un cargador de arranque. La primera memoria incluye un área de almacenamiento intermedio, y el cargador de arranque incluye una parte anfitrión y una parte cliente. La parte anfitrión se puede cargar en la primera memoria y tiene una primera etapa y una segunda etapa. La primera etapa reinicia y mantiene el procesador cliente en un estado de reinicio y empuja la información a la primera memoria. La segunda etapa es iniciada por la parte cliente, carga el área de almacenamiento intermedio con información que ha de ser cargada en la segunda memoria, y envía a la parte cliente un primer mensaje que indica que el área de almacenamiento intermedio está cargada. La parte cliente que se puede cargar en la primera memoria, pone en marcha un sistema operativo que incluye inicialmente el bloqueo de la planificación de todos los procesos que tienen código de programa ubicado en la segunda memoria, copia la información cargada en el área de almacenamiento intermedio a la segunda memoria, envía a la parte anfitrión un segundo mensaje que indica que la información ha sido copiada, y libera el bloqueo de la planificación de los procesos que tienen código de programa ubicado en la segundo memoria.

En otro aspecto de la invención, un medio legible por ordenador contiene un programa informático para cargar información que incluye un cargador de arranque en un procesador esclavo en un sistema multiprocesador que incluye un procesador maestro y el procesador esclavo. El programa informático realiza los pasos del método resumido anteriormente.

Breve descripción de los dibujos

Las distintas características, objetos y ventajas de esta invención serán comprendidos mediante la lectura de esta descripción en unión con los dibujos, en los que:

La fig. 1 representa un sistema multiprocesador;

La fig. 2 es un diagrama de flujo de un cargador de arranque de sistema operativo fácil de usar; y

La fig. 3 representa un ejemplo de una organización de un área de almacenamiento intermedio.

Descripción detallada

Como se ha observado anteriormente, un cargador de arranque convencional puede entrar en conflicto con el sistema operativo de un sistema multiprocesador. Esta solicitud describe un cargador de arranque de OS fácil de usar y los métodos que satisfacen el desafío de integrar un OS con un cargador de arranque en sistemas en los cuales el anfitrión y un cliente tienen un mecanismo de comunicación que requiere al OS para que el mecanismo funcione y el cliente tiene dos sistemas de memoria: una visible tanto para el anfitrión como para el cliente y una visible solamente para el cliente.

La fig. 1 representa tal sistema 100 multiprocesador que incluye un procesador 102 anfitrión y un procesador 104 cliente. Se apreciará que aunque la fig. 1 muestra un procesador 104 cliente, puede haber más previstos. También se apreciará que los procesadores anfitrión y cliente pueden ser cualesquiera procesadores electrónicos programables. En el ejemplo representado en la fig. 1, el procesador 102 es mostrado como la unidad de tratamiento central (CPU) de una máquina RISC avanzada (ARM), y el procesador 104 es mostrado como la CPU de un dispositivo de procesador de señal digital (DSP). La línea discontinua en la fig. 1 representa el límite hardware entre

los dispositivos anfitrión y esclavo, en este ejemplo, el ARM y el DSP, y también una memoria 106 no volátil. La memoria 106 puede ser una ROM, una memoria flash, u otro tipo de dispositivo de memoria no volátil.

La mayoría de los dispositivos DSP comercialmente disponibles incluyen memorias en el chip, y como se ha indicado en la fig. 1, el DSP incluye RAM de un solo acceso "interno" (SARAM) y RAM de acceso doble (DARAM) 108, así como una RAM 110 "externa" (XRAM). Un área de almacenamiento intermedio, indicada por la línea discontinua, está definida dentro de la memoria 108 como se ha descrito en más detalle a continuación. Las flechas en la fig. 1 indican trayectos de acceso, por ejemplo, buses y trayectos DMA, entre las CPU y las memorias. La CPU 102 de anfitrión ARM puede acceder a la memoria 106 no volátil y a la SARAM y a la DARAM 108 del DSP, pero no a la XRAM 110 del DSP, y la CPU 104 del esclavo DSP puede acceder a todas las RAM 108, 110.

La SARAM y la DARAM 108 pueden ser cargadas desde la memoria 106 no volátil por el método de "empuje" trivial. Cuando el código necesita ser cargado a la XRAM 110 durante el arranque, sin embargo, se requiere una solución de cargador de arranque por que la XRAM 110 es invisible a, es decir, no accesible mediante, la CPU 102 y así el código de arranque no puede ser empujado a la XRAM 110.

Como se ha descrito en más detalle a continuación en conexión con el diagrama de flujo de la fig. 2, un cargador de arranque de OS fácil de usar tiene ventajosamente una parte anfitrión y una parte cliente que es cargada en una memoria o memorias visibles tanto para el maestro como para el esclavo (por ejemplo, la SARAM y la DARAM 108).

La parte anfitrión del cargador de arranque de OS fácil de usar puede ser considerada como que incluye dos etapas o modos de operación. La primera etapa reinicia y mantiene el esclavo 104 en el estado de reinicio (Paso 202) y empuja la información (instrucciones de programa y/o datos) (Paso 204) del modo normal desde la memoria 106 no volátil a las memorias 108 comúnmente visibles. La información empujada a estas memorias es principalmente el cargador de arranque, el OS, y cualquier código de puesta en marcha necesario para el OS. Debería apreciarse que una aplicación o aplicaciones o partes de las mismas pueden ser empujadas también a estas memorias al inicio y pueden comenzar a ejecutarse durante la carga de la memoria 110 "externa". Cuando este "empuje" es finalizado (Paso 206), se permite que el esclavo 104 arranque (Paso 208) e inicie el OS (por ejemplo, es liberado desde el estado de reinicio) y sus mecanismos de comunicación normal (Paso 210). La parte anfitrión espera a continuación un mensaje desde el esclavo, el cual inicia la operación de su segunda etapa como se ha descrito en más detalle a continuación.

La parte esclavo del cargador de arranque de OS fácil de usar que es cargada ("empujada" por la primera etapa de la parte anfitrión) a las memorias 108 comúnmente visibles inicia el sistema operativo, llevando a cabo las siguientes operaciones (Paso 210). En primer lugar, se crean controladores de interrupción. El código para los controladores de interrupción debe estar ubicado en la memoria que ya está cargada porque puede ocurrir una interrupción en cualquier momento. En segundo lugar, se crean estructuras de datos (por ejemplo, bloques y pilas de control de procesos) de procesos comunes, es decir, procesos que se ejecutan tanto en el anfitrión como en el esclavo. Debería comprenderse que ya que estos procesos comunes no han sido ejecutados todavía, su código puede ser cargado en un momento posterior y puede estar ubicado muy bien en la memoria "externa" visible solamente para el esclavo, por ejemplo, la XRAM 110. En tercer lugar, se crea el proceso inactivo del sistema. El código para el proceso inactivo debe estar ubicado en la memoria que ya está cargada porque el proceso inactivo es el proceso seleccionado para ejecutar por el OS si no hay nada útil que hacer. En cuarto lugar, se bloquea la planificación de al menos todos los procesos que residen en, es decir, que tienen código de programa o datos ubicados en, la memoria 110 "externa". La ejecución de los procesos que residen en la memoria "interna" puede así iniciar o continuar ventajosamente en paralelo con la carga de la memoria "externa" como se ha observado anteriormente. Es también posible detener la planificación de todos los procesos excepto del proceso inactivo, pero esto no es necesario. Haciendo este bloqueo lo último que se ha hecho antes de que el planificador del OS se active asegura que el código en estos procesos no se ejecutará cuando se libere el planificador. Finalmente, el planificador del OS es liberado, lo cual permite al OS poner en marcha el código de ejecución y los procesos de planificación. Se comprenderá que ya que al menos fue bloqueada toda la planificación de los procesos de memoria externa, todo lo que el OS puede hacer ahora es planificar interrupciones y el proceso inactivo.

En este punto, el esclavo 104 está parcialmente en marcha y ejecutándose. La parte esclava del cargador de arranque de OS fácil de usar ha sido cargada, y se está ejecutando el proceso inactivo del esclavo. El OS del esclavo puede planificar y ejecutar código en respuesta a interrupciones y puede planificar el proceso inactivo y cualesquiera procesos desbloqueados que tienen código que reside en la memoria interna. Los mecanismos del OS para que todos los accesos de código y datos estén en la memoria que ya ha sido cargada (SARAM y DARAM 108, en este ejemplo) están disponibles, incluyendo los mecanismos de comunicación normal. Estos mecanismos de comunicación del OS, que son abstracciones de alto nivel de la DMA, memoria compartida, y registros estructurados, son más capaces que simples semáforos y habilitan al procesador anfitrión para comunicar eficientemente con un procesador (el esclavo) que no se ha iniciado completamente, es decir un procesador que está ejecutando principalmente solamente el OS, los servicios de interrupción, y procesos que residen en la RAM "interna".

El proceso inactivo reserva un bloque de memoria de la pila de memoria del esclavo que está ubicada en la memoria visible para el anfitrión, tal como la memoria 108 "interna" (Paso 212). Como se ha descrito en más detalle a

continuación, este bloque de memoria reservado es utilizado para almacenamiento intermedio de información (código y/o datos) que ha de ser transferida a la memoria privada del esclavo, es decir, la memoria que es invisible para el anfitrión, tal como la XRAM 110 "externa". El proceso inactivo del esclavo utiliza ventajosamente los mecanismos de comunicación establecidos para enviar al anfitrión (Paso 214) información sobre la dirección y tamaño o longitud del área de almacenamiento intermedio reservada en el paso anterior. Después de enviar la información, que puede estar contenida en uno o más mensajes adecuados, los bloques esclavos, esperan un mensaje desde el anfitrión. Mientras está "bloqueado" el esclavo no realiza ninguna otra actividad de carga hasta que recibe la respuesta del anfitrión.

Se comprenderá que si el OS del esclavo actúa sobre una interrupción en esta etapa depende de la naturaleza de la interrupción. Dado que muchos mecanismos del OS (como los utilizados para comunicar con el anfitrión, por ejemplo) dependen de interrupciones, y no puede saberse por adelantado cuándo ocurrirá una interrupción, todo el código de interrupción debe haber sido cargado en la memoria "interna". En ese sentido, las interrupciones son servidas durante la segunda etapa de la carga de arranque. Sin embargo, si una interrupción va a activar una cadena de eventos tales como procesos que comienzan a hacer algún tratamiento de datos y el código o datos para esos procesos están ubicados o serán ubicados en la memoria "externa", la interrupción es bloqueada y el servicio de interrupción pone la solicitud en la "cola de entrada" de ese proceso de manera que la solicitud será servida después de que el arranque haya finalizado y que el proceso puede ejecutarse.

A la recepción de la información del esclavo, la segunda etapa del cargador de arranque del anfitrión llena el área de almacenamiento intermedio con información (código y/o datos) que ha de ser cargada en la memoria invisible del esclavo (Paso 216). El código y los datos son empujados al área de almacenamiento intermedio de un modo normal porque este área es la memoria a la que ambos procesadores pueden acceder, pero el empuje es activado a través de los mecanismos de comunicación del OS.

El anfitrión envía ahora un mensaje al esclavo (Paso 218) que indica que el área de almacenamiento intermedio ha sido cargado y si la carga ha finalizado o hay disponibles más código y/o datos. Este es el mensaje que está esperando el esclavo. El anfitrión a su vez se bloquea ahora, esperando un mensaje desde el esclavo. El esclavo copia los contenidos del área de almacenamiento intermedio a ubicaciones adecuadas en su memoria privada de esclavo (Paso 220), implementando por ello su carga real. El esclavo envía a continuación un mensaje al anfitrión (Paso 222) que indica que el esclavo ha copiado los contenidos del área de almacenamiento intermedio.

Si existe más código y/o datos para cargar (Paso 224), este ciclo de copiar y enviar mensajes (Pasos 216-224) puede ser repetido tantas veces como se requiera. Cuando la carga es finalizada, es decir, cuando no se necesita copiar más información al esclavo, el esclavo libera el bloqueo de los procesos que fueron bloqueados antes, permitiendo por tanto la planificación del código en su memoria privada de esclavo (Paso 226). La carga está ahora completa.

Como se ha descrito anteriormente, el anfitrión llena el área de almacenamiento intermedio en la memoria 108 con código y datos que el esclavo copia además a destinos finales en la memoria 110 privada del esclavo. Quizás el modo más simple de hacer esto es preceder todo el código y datos en el área de almacenamiento intermedio con una etiqueta que contiene la dirección de destino y la longitud del bloque que ha de ser cargado. La fig. 3 representa un ejemplo de tal organización del área de almacenamiento intermedio. Un bloque de código y/o datos que han de ser transferidos al área de almacenamiento intermedio incluye una cabecera que indica la longitud del bloque y dónde ha de ser cargado en la memoria del esclavo, es decir, la dirección de destino. Como se ha indicado por las líneas discontinuas en la fig. 3, varios de tales bloques pueden ser concatenados en el área de almacenamiento intermedio.

La información (código y datos) que ha de ser cargada puede estar dispuesta de muchas maneras en el área de almacenamiento intermedio y las memorias. A menudo la información está dispuesta como bloques de información consecutiva que han de ser cargados a diferentes direcciones, y así un tamaño arbitrariamente elegido del área de almacenamiento intermedio puede no coincidir con los tamaños de todos los bloques. Aún, debería comprenderse que el sistema operará más eficientemente cuando el área de almacenamiento intermedio esté siempre llena. Esto significa que si los bloques que han de ser cargados son más pequeños que este área, debería hacerse una alta transferencia de varios bloques (más pequeños) al mismo tiempo. Esto significa también que un bloque debería ser dividido si es más grande que la parte restante del área de almacenamiento intermedio, y una parte transferida al área de almacenamiento intermedio con la parte restante transferida en el siguiente bloque. Además, si un bloque es varias veces más grande que el área de almacenamiento intermedio, puede que tenga que dividirse más de una vez. Toda esta división y concatenación es hecha en la parte anfitrión del cargador de arranque de OS fácil de usar de manera que son bien conocidos para los informáticos. Desde el punto de vista de ingenieros de comunicaciones de datos, la parte anfitrión del cargador de arranque de OS fácil de usar es así un tipo de "capa de transporte".

El técnico comprenderá el beneficio de esta división y concatenación de información en bloques de transferencia. Se requiere algún tipo de mecanismo de comunicación para realizar las transferencias reales de información entre memorias, y cualquiera que sea el mecanismo utilizado, transferencias un poco más grandes son preferibles típicamente a transferencias más pequeñas. Un área de almacenamiento intermedio mantenida llena puede hacer el uso del ancho de banda disponible más eficiente minimizando ventajosamente la sobrecarga en el canal de

comunicaciones. Cada mensaje requiere alguna cantidad de administración e información administrativa, y por lo tanto menos mensajes significa menos sobrecarga.

Un buen ejemplo del beneficio del efecto de división y concatenación del bloque es el DMA como el mecanismo de comunicación. El DMA requiere típicamente alguna sobrecarga de configuración (es decir, lleva algún tiempo configurarlo), pero a continuación el DMA es muy eficiente una vez que ha sido puesto en marcha porque las transferencias pueden ser llevadas a cabo en ciclos de CPU mínimos. Con el fin de obtener el mayor beneficio del uso del DMA, debería hacerse cada vez más grande la transferencia de DMA permitida por el hardware. Así, actualmente se cree que es ventajoso ajustar el tamaño del área de almacenamiento intermedio al tamaño del bloque de DMA máximo.

La parte anfitrión del cargador de arranque de OS fácil de usar debería "saber " cuando dejar su primera etapa (cargar la información empujándola a la memoria) y entrar en su segunda etapa (cargar la información a través de uno o más mecanismos de comunicación). Después de todo, el anfitrión no puede empujar la información a la memoria que es invisible para él. Aunque el esclavo envía un mensaje a la parte anfitrión cuando ha alcanzado el proceso inactivo, esto puede no ser suficiente para que la parte anfitrión le diga al esclavo que comience a ejecutarse. Esta transición del empuje a la carga de arranque será vista como un cambio desde el paradigma de la carga pasiva (es decir, no se está ejecutando código en el esclavo) al paradigma de la carga activa (es decir, un esclavo parcialmente vivo, ejecutándose).

Un modo de que la parte anfitrión sepa cuando cambiar las etapas es etiquetar el código y los datos que han de ser cargados con información sobre a qué memoria serán cargados. Por ejemplo, la información destinada a la memoria invisible podría incluir una etiqueta o etiquetas que indican la información que ha de ser cargada en la memoria invisible. La ausencia de tal etiqueta podría indicar que la información ha de ser cargada en la memoria visible, aunque se apreciará que podría utilizarse también una etiqueta que indica explícitamente que la información ha de ser cargada en la memoria visible. Esto permite al anfitrión hacer dos pasadas sobre la información y cargar solamente la información requerida en cada pasada. En la primera pasada, las cosas que van a la memoria interna serían encontradas y cargadas, y en la segunda pasada, las cosas que van a la memoria externa serían encontradas y cargadas.

Otro modo, que parece actualmente ser más simple, es disponer la memoria privada del esclavo de tal manera que toda ella resida por encima (o por debajo) de una dirección predeterminada. La información que ha de ser transferida es a continuación ordenada en consecuencia, con todas las secciones de código y datos que han de ser cargados a la memoria privada del esclavo puestas al final (o al comienzo) de la imagen ordenada. A continuación, todo lo que ha de hacer la parte anfitrión del cargador de arranque de OS fácil de usar es entrar en su segunda etapa cuando es encontrada una dirección más grande (o más pequeña) que la dirección predeterminada (límite).

Con el fin de ahorrar memoria o aumentar la integridad del código y la seguridad de la plataforma en el lado del anfitrión, la información que ha de ser cargada al esclavo puede ser también procesada previamente de diferentes maneras. Por ejemplo, la información puede ser comprimida según un algoritmo adecuado, reduciendo por tanto el tamaño de la memoria necesaria para ello en el lado del anfitrión. Como otro ejemplo, la información puede ser cifrada, fortaleciendo por ello la seguridad de la plataforma, ya que un hacker potencial no será capaz de desensamblar la información fácilmente. Actualmente se cree que el cifrado es valioso si la información que ha de ser cargada al esclavo es almacenada en el sistema de archivos interno del anfitrión, donde la información está disponible (al menos en teoría) para cualquiera.

A partir de esta descripción, se comprenderá que los mecanismos del OS están disponibles para la parte esclavo del cargador de arranque de OS fácil de usar que es ejecutada por el procesador esclavo y que el esclavo puede reutilizar el código dependiente del OS existente requerido para la comunicación. Además, el cargador de arranque de OS fácil de usar utiliza recursos de carga (por ejemplo, DMA) eficientemente, con la parte anfitrión que decide automáticamente cuando conmutar desde una primera etapa, o modo de empuje, a una segunda etapa, o modo de cargador de arranque.

Se espera que esta invención pueda ser implementada en una amplia variedad de entornos, incluyendo por ejemplo dispositivos de comunicación móviles. Los dispositivos más nuevos de entre ellos pueden emplear el cargador de arranque de OS fácil de usar descrito aquí para arrancar sus DSP, lo que puede estar previsto para manejar tareas multimedia, en cooperación con sus sistemas de software de procesador principal.

El cargador de arranque de OS fácil de usar descrito aquí tiene en cuenta el sistema operativo y se ejecuta realmente sobre un sistema operativo. El anfitrión se está ejecutando completamente cuando el esclavo es arrancado o vuelto a arrancar de nuevo. Este cargador de arranque no requiere que el procesador anfitrión esté en cierto estado con el fin de poner en marcha el procesador esclavo. En efecto, la puesta en marcha del procesador esclavo puede ser llevado a cabo en cualquier momento durante la ejecución del software del procesador anfitrión. El cargador de arranque de OS fácil de usar no necesita que un archivo ejecutable especial sea ejecutado en el procesador esclavo mientras la información está siendo cargada a éste desde el procesador anfitrión y la RAM inaccesible para la anfitrión. Un ejecutable es vinculado a todas las memorias del procesador esclavo. El esclavo es arrancado antes de que todo el código sea cargado, pero el código que está vinculado a la memoria inaccesible para

el anfitrión no es ejecutado hasta que es cargado con ayuda del código que está vinculado a la memoria accesible para el anfitrión del procesador esclavo.

Se comprenderá por tanto que el cargador del OS fácil de usar descrito aquí hace posible cambiar la ejecución del software en el procesador esclavo e iniciar la ejecución de esclavo de un software de aplicación antes de que sea completamente cargado. Uno o más procesos de aplicación pueden ser elegidos para "empujar" con el cargador de arranque en la memoria accesible para el anfitrión del procesador esclavo, y dichos procesos iniciarán la ejecución en el mismo punto del tiempo en el que la memoria inaccesible para el anfitrión del procesador esclavo comienza a ser cargada.

Esta capacidad puede ser importante en muchos dispositivos y en muchos casos de uso. En un teléfono móvil, por ejemplo, tales casos de uso incluyen hacer una llamada, recibir una llamada, comprimir/descomprimir la voz, reproducir archivos de música, etc. Con el cargador de arranque de OS fácil de usar descrito aquí, uno puede cargar y ejecutar software nuevo en el procesador esclavo virtualmente en cualquier momento mientras que el procesador anfitrión está funcionando.

Se apreciará que los procedimientos descritos anteriormente son llevados a cabo repetitivamente si es necesario. Para facilitar la comprensión, muchos aspectos de la invención son descritos en términos de secuencias de acciones que pueden ser realizadas mediante, por ejemplo, elementos de un sistema informático programable. Se reconocerá que distintas acciones podrían ser realizadas por circuitos especializados (por ejemplo, puertas lógicas discretas interconectadas para realizar una función especializada o circuitos integrados específicos de aplicación), por instrucciones de programa ejecutadas por uno o más procesadores, o por una combinación de ambos.

Además, la invención descrita aquí puede ser considerada adicionalmente que ha de ser realizada completamente dentro de cualquier forma de medio de almacenamiento legible por ordenador que tiene almacenada en él un conjunto de instrucciones adecuadas para utilizar por o en conexión con un sistema, aparato, o dispositivo de ejecución de instrucción, tal como un sistema basado en ordenador, sistema que contiene procesador, u otro sistema que puede buscar instrucciones desde un medio y ejecutar las instrucciones. Como se ha utilizado aquí, un "medio legible por ordenador" puede ser cualquier medio que contiene, almacena, comunica, propaga, o transporta el programa para utilizar por o en conexión con el sistema, aparato, o dispositivo de ejecución de instrucción. El medio legible por ordenador puede ser, por ejemplo pero no está limitado a, un sistema, aparato, dispositivo, o medio de propagación electrónico, magnético, óptico, electromagnético, infrarrojo o semiconductor. Más ejemplos específicos (una lista no exhaustiva) del medio legible por ordenador incluyen una conexión eléctrica que tiene uno o más cables, un disquete de ordenador portátil, una RAM, una ROM, una memoria de sólo lectura programable regrabable (EPROM o memoria Flash), y una fibra óptica.

Así, la invención puede ser realizada de muchas formas diferentes, no todas las cuales son descritas anteriormente, y todas estas formas son contempladas para estar dentro del alcance de la invención. Para cada uno de los distintos aspectos de la invención, cualquier forma puede ser referida como "lógica configurada para" realizar una acción descrita, o alternativamente como "lógica que" realiza una acción descrita.

Se enfatiza que el término "comprende" y "que comprende ", cuando es utilizado en esta solicitud, especifica la presencia de características declaradas, números enteros, pasos, o componentes y no excluye la presencia o adición de una o más de otras características, números enteros, pasos, componentes, o grupos de los mismos.

Las realizaciones particulares descritas anteriormente son meramente ilustrativas y no deberían ser consideradas restrictivas de ningún modo. El alcance de la invención es determinado por las siguientes reivindicaciones, y todas las variaciones y equivalencias que caen dentro del rango de las reivindicaciones están destinadas para ser abarcadas en ellas.

REIVINDICACIONES

1.- Un método para cargar información en un procesador (104) esclavo en un sistema (100) multiprocesador que incluye un procesador (102) maestro y el procesador esclavo, incluyendo la información un cargador de arranque, que comprende los pasos de:

a) reiniciar el procesador (104) esclavo y mantener el procesador (104) esclavo en un estado de reinicio;

b) empujar una primera parte de la información a una primera memoria (108) que es accesible mediante el procesador (102) maestro y el procesador (104) esclavo, incluyendo la primera parte de la información el cargador de arranque;

c) arrancar el procesador (104) esclavo;

d) poner en marcha un sistema operativo en el procesador (104) esclavo, que incluye el bloqueo de la planificación de procesos que tienen código de programa ubicado en una segunda memoria (110) que es accesible mediante el procesador (104) esclavo e inaccesible mediante el procesador (102) maestro;

e) reservar un área de almacenamiento intermedio en la primera memoria (108):

f) enviar al procesador (102) maestro la información sobre una ubicación y tamaño del área de almacenamiento intermedio reservada;

g) basándose en la información enviada, cargar el área de almacenamiento intermedio con una segunda parte de la información que ha de ser cargada en la segunda memoria (110);

h) enviar un primer mensaje al procesador (104) esclavo que indica que el área de almacenamiento intermedio ha sido cargada y si la carga ha finalizado o ha de cargarse más información;

i) copiar información en el área de almacenamiento intermedio a la segunda memoria (110);

j) enviar un segundo mensaje al procesador (102) maestro que indica que la información en el área de almacenamiento intermedio ha sido copiada; y

k) liberar el bloqueo de la planificación de procesos que tienen código de programa ubicado en la segunda memoria (110).

2.- El método de la reivindicación 1, que comprende además el paso de, si hay más información que ha de ser cargada, repetir los pasos g-j hasta que no haya más información que ha de ser cargada.

3.- El método de la reivindicación 1, en donde el paso d) incluye crear controladores de interrupción y crear estructuras de datos de los procesos que se ejecutan tanto en el procesador (102) maestro como en el procesador (104) esclavo.

4.- El método de la reivindicación 1, en donde la primera parte de la información empujada a la primera memoria (108) incluye además el sistema operativo y al menos una parte de al menos una aplicación.

5.- El método de la reivindicación 1, en donde el paso d) incluye el bloqueo de la planificación de todos los procesos excepto de un proceso inactivo.

6.- El método de la reivindicación 1, en donde el paso g) incluye preceder la segunda parte de la información que ha de ser cargada a la segunda memoria (110) con al menos una etiqueta que indica una dirección en la segunda memoria (110) y una longitud de la información que ha de ser cargada.

7.- El método de la reivindicación 1, en donde el paso g) incluye dividir la segunda parte de la información que ha de ser cargada a la segunda memoria (110) en bloques y cargar el área de almacenamiento intermedio con un primer bloque.

8.- El método de la reivindicación 7, que comprende además el paso de repetir los pasos g-j para cada bloque en sucesión.

9.- El método de la reivindicación 1, en donde la primera parte de la información empujada a la primera memoria (108) incluye al menos una etiqueta que indica si la información ha de ser cargada a la segunda memoria (110).

10.- El método de la reivindicación 1, en donde la segunda parte de la información que ha de ser cargada a la segunda memoria (110) es al menos una de entre información comprimida y cifrada.

11.- Un sistema (100) multiprocesador, que comprende:

un procesador (102) anfitrión;

al menos un procesador (104) cliente;

una primera memoria (108) accesible mediante el procesador anfitrión y el procesador cliente, en donde la primera memoria incluye un área de almacenamiento intermedio;

una segunda memoria (110) accesible mediante el procesador cliente y no accesible mediante el procesador anfitrión; y

un cargador de arranque que incluye:

una parte de anfitrión que se puede cargar a la primera memoria, que tiene una primera etapa que reinicia y mantiene el procesador (104) cliente en un estado de reinicio y empuja la información a la primera memoria (108), y que tiene una segunda etapa que es iniciada por la parte cliente, que carga el área de almacenamiento intermedio con información que ha de ser cargada a la segunda memoria (110) y que envía a la parte cliente un primer mensaje que indica que el área de almacenamiento intermedio está cargada; y

una parte cliente que se puede cargar a la primera memoria (108), que pone en marcha un sistema operativo que incluye bloquear inicialmente la planificación de todos los procesos que tienen código de programa ubicado en la segunda memoria (110), que copia la información cargada al área de almacenamiento intermedio a la segunda memoria (110), que envía a la parte anfitrión un segundo mensaje que indica que la información ha sido copiada, y que libera el bloqueo de la planificación de todos los procesos que tienen código de programa ubicado en la segunda memoria.

12.- El sistema de la reivindicación 11, en donde la información empujada a la primera memoria (108) mediante la primera etapa incluye la parte cliente del cargador de arranque y el sistema operativo, y la segunda etapa permite a los procesadores (102 y 104) anfitrión y cliente intercambiar mensajes

13.- El sistema de la reivindicación 12, en donde la información empujada a la primera memoria (108) mediante la primera etapa incluye al menos una parte de al menos una aplicación.

14.- El sistema de la reivindicación 11, en donde la parte cliente bloquea inicialmente la planificación de todos los procesos excepto el proceso inactivo.

15.- El sistema de la reivindicación 11, en donde el proceso inactivo envía al procesador (102) anfitrión la información sobre una dirección y una longitud del área de almacenamiento intermedio.

16.- El sistema de la reivindicación 11, en donde la información que ha de ser cargada a la segunda memoria (110) incluye al menos una etiqueta que indica una dirección en la segunda memoria (110) y una longitud de la información que ha de ser cargada.

17.- El sistema de la reivindicación 11, en donde el área de almacenamiento intermedia está organizada como bloques de información, y cada bloque incluye una cabecera que indica una longitud del bloque y una dirección para el bloque en la segunda memoria (110).

18.- El sistema de la reivindicación 11, en donde la información empujada a la primera memoria (108) incluye al menos una etiqueta que indica si la información ha de ser cargada a la segunda memoria (110).

19.- El sistema de la reivindicación 11, en donde la información que ha de ser cargada a la segunda memoria (110) es al menos una de entre información comprimida y cifrada.

20.- Un medio legible por ordenador que contiene un programa informático para cargar información a un procesador (104) esclavo en un sistema (100) multiprocesador que incluye un procesador (102) maestro y el procesador (104) esclavo, en donde la información incluye un cargador de arranque y el programa informático realiza los pasos de:

a) reiniciar el procesador (104) esclavo y mantener el procesador (104) esclavo en un estado de reinicio;

b) empujar una primera parte de la información a una primera memoria (108) que es accesible mediante el procesador (102) maestro y el procesador (104) esclavo, incluyendo la primera parte de la información el cargador de arranque;

c) arrancar el procesador (104) esclavo

d) poner en marcha un sistema operativo en el procesador (104) esclavo, que incluye el bloqueo de la planificación de procesos que tienen código de programa ubicado en una segunda memoria (110) que es accesible mediante el procesador (104) esclavo e inaccesible mediante el procesador (102) maestro;

e) reservar un área de almacenamiento intermedio en la primera memoria (108):

f) enviar al procesador (102) maestro la información sobre una ubicación y tamaño del área de almacenamiento intermedio reservada;

g) basándose en la información enviada, cargar el área de almacenamiento intermedio con una segunda parte de la información que ha de ser cargada en la segunda memoria (110);

h) enviar un primer mensaje al procesador (104) esclavo que indica que el área de almacenamiento intermedio ha sido cargada y si la carga ha finalizado o ha de cargarse más información;

i) copiar información en el área de almacenamiento intermedio a la segunda memoria (110);

j) enviar un segundo mensaje al procesador maestro (102) que indica que la información en el área de almacenamiento intermedio ha sido copiada; y

k) liberar el bloqueo de la planificación de procesos que tienen código de programa ubicado en la segunda memoria (110).

21.- El medio legible por ordenador de la reivindicación 20, en donde el programa informático realiza además el paso de, si hay más información que ha de ser cargada, repetir los pasos g-j hasta que no haya más información que ha de ser cargada.

22.- El medio legible por ordenador de la reivindicación 20, en donde el paso g) incluye preceder la segunda parte de la información que ha de ser cargada a la segunda memoria (110) con al menos una etiqueta que indica una dirección en la segunda memoria (110) y una longitud de la información que ha de ser cargada.

23.- El medio legible por ordenador de la reivindicación 20, en donde el paso g) incluye dividir la segunda parte de la información que ha de ser cargada a la segunda memoria (110) en bloques y cargar el área de almacenamiento intermedio con un primer bloque.

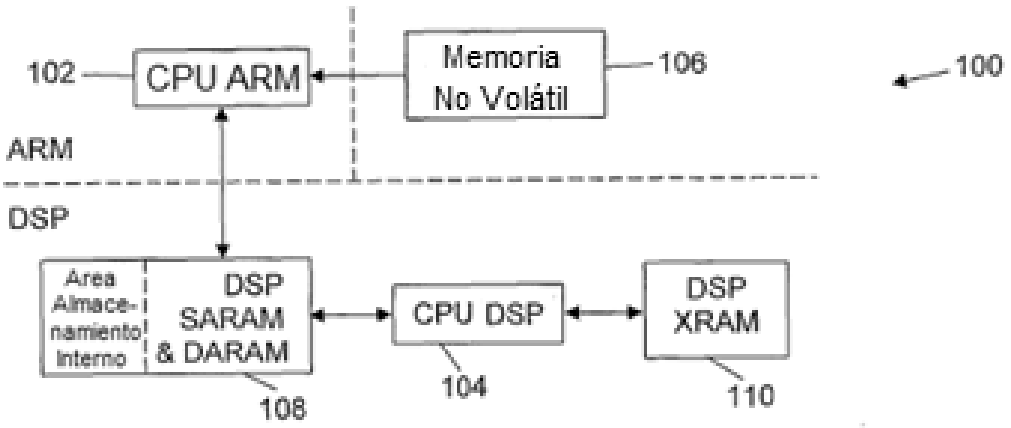
102
CPU ARM
Memoria
No Volátil
106
100
ARM
DSP
Area
Almace-
namiento
Interno
DSP
SARAM
& DARAM
108
CPU DSP
104
DSP
XRAM
110

FIG. 1

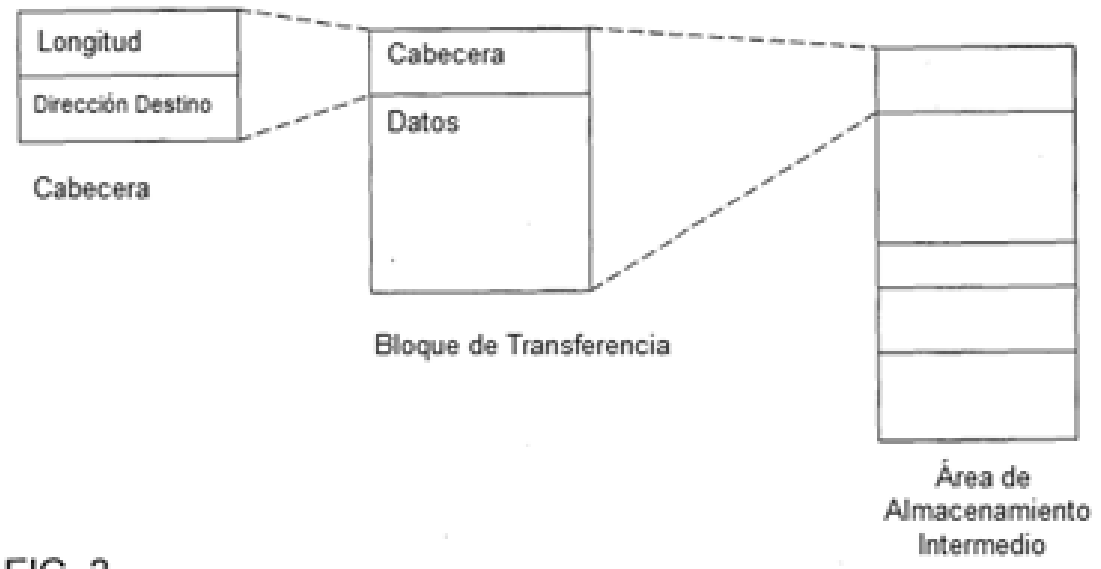
Longitud
Dirección Destino
Cabecera
Cabecera
Datos
Bloque de Transferencia
Área de
Almacenamiento
Intermedio

FIG. 3

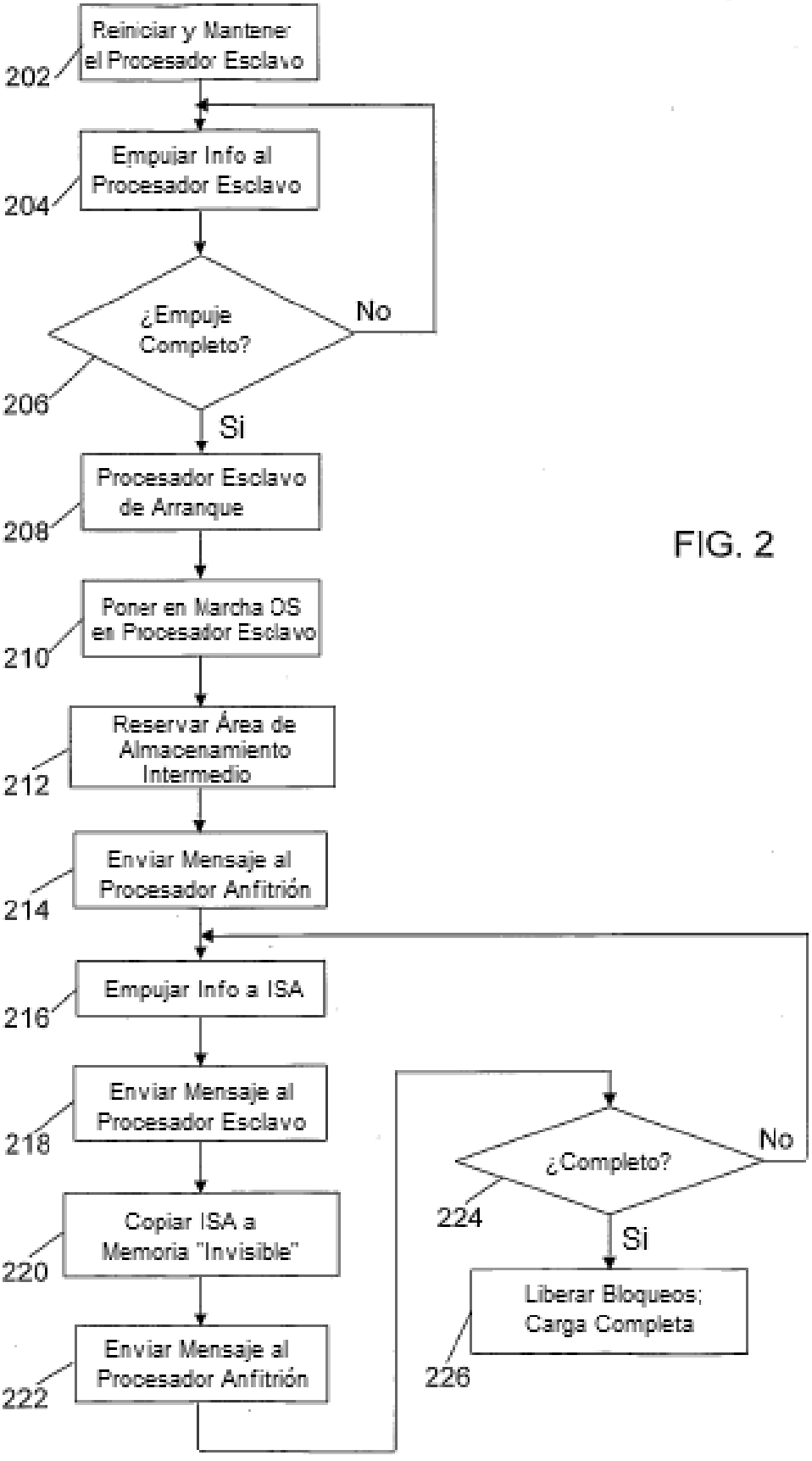
Reiniciar y Mantener el Procesador Esclavo
202
Empujar Info al Procesador Esclavo
204
¿Empuje Completo?
206
No
Si
Procesador Esclavo de Arranque
208
Poner en Marcha OS en Procesador Esclavo
210
Reservar Área de Almacenamiento Intermedio
212
Enviar Mensaje al Procesador Anfitrión
214
Empujar Info a ISA
216
Enviar Mensaje al Procesador Esclavo
218
Copiar ISA a Memoria "Invisible"
220
Enviar Mensaje al Procesador Anfitrión
222
¿Completo?
224
No
Si
Liberar Bloqueos; Carga Completa
226

FIG. 2