(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 712 157**

(51) Int. CI.:

G06F 9/50 (2006.01)
G06F 9/46 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **14.08.2008 E 08290774 (2)**

(97) Fecha y número de publicación de la concesión europea: **21.11.2018 EP 2031512**

(54) Título: **Gestor de procesos mejorado**

(30) Prioridad:

31.08.2007 FR 0706119

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
09.05.2019

(73) Titular/es:

BULL S.A.S. (100.0%)
Rue Jean Jaurès, BP 68
78340 Les Clayes-sous-Bois, FR

(72) Inventor/es:

MENYHART, ZOLTAN

(74) Agente/Representante:

AZNÁREZ URBIETA, Pablo

DESCRIPCIÓN

Gestor de procesos mejorado

La presente invención se refiere a un gestor de procesos en el contexto de la ejecución de procesos relacionados con un programa informático dentro de un sistema operativo.

Un proceso informático es un programa ejecutado por un ordenador, con un espacio de direccionamiento adecuado adjunto.

En los sistemas operativos modernos, el espacio de direccionamiento asociado a un proceso dado está protegido, para evitar que otro proceso (por ejemplo, de otra aplicación) altere sus datos críticos. Esta protección es de hardware, es decir, los procesadores constan de registros de hardware especializados para cada proceso.

Esta protección de hardware sin embargo plantea problemas de rendimiento, en el sentido de que algunos procesos tienen que compartir datos o servicios, pero no puede debido a esta partición. Entonces es necesario hacer varias copias de los datos a compartir, lo que es costoso en términos de potencia y tiempo.

Se han desarrollado soluciones del tipo multihilo (multi-threading) para permitir acelerar el rendimiento de la ejecución del proceso, aprovechando el hecho de que varios procesadores estaban disponibles. Sin embargo, estas soluciones requieren aplicaciones de escritura particulares y, por lo tanto, no son compatibles con las aplicaciones ya existentes.

También se conoce a partir del documento EP 1280051 un método de procesamiento en el entorno de ejecución de JAVA para ejecutar varias aplicaciones JAVA en una sola máquina virtual JAVA. Para hacer esto, un módulo de control activa una sola máquina virtual JAVA y genera un proceso interno liviano para cada aplicación JAVA que se ejecutará.

Según este método, se reduce la cantidad de memoria necesaria para activar la máquina virtual Java y el tiempo de procesamiento de inicio necesario para iniciar una nueva aplicación Java. No obstante, esta solución presenta una ausencia de protección entre los recursos.

La presente invención viene a mejorar la situación.

Para tal efecto, la invención propone un gestor de procesos que comprende una memoria de datos de proceso y una unidad de procesamiento de procesos, capaz de ejecutar un código de proceso que interactúa con una parte designada de la memoria de datos de proceso.

La unidad de procesamiento de procesos es capaz de establecer un superproceso asociado a una porción designada de la memoria capaz de alojar datos de uno o varios procesos, y es capaz de ejecutar el código de cada uno de estos procesos en interacción distinta con sus datos.

Este gestor también comprende una función de agrupación dispuesta para alojar procesos de incidentes en un superproceso en respuesta a una condición, al menos en parte, relacionada con estos procesos elementales incidentes.

Además, la unidad de procesamiento de procesos está dispuesta para alojar procesos de incidentes en un superproceso en respuesta a una condición relacionada al menos parcialmente con el código de estos procesos de incidentes.

Tal gestor de procesos es particularmente interesante porque permite que los procesos seleccionados se recopilen como subprocesos dentro del mismo proceso compartido y, por lo tanto, puedan trabajar en un espacio de memoria común.

En unos modos de realización particulares, el gestor presenta las siguientes características:

* la parte designada de la memoria de un superproceso dado contiene todos los datos de proceso de todos los procesos asociados a él;

* la condición de alojamiento de los procesos de incidentes afecta a una lista recibida por el gestor de procesos que asocia un identificador de un proceso de incidente con un identificador de un superproceso;

* el identificador de un proceso incidente está asociado con un identificador de un superproceso en función de al menos una parte del código del proceso;

* el gestor de procesos se implementa en forma de un demonio;

* el gestor de procesos se implementa en un sistema operativo Linux;

* los datos del proceso constan de una pila, un montón, un bloque BSS y un bloque de datos;

* los datos de proceso constan, además, de un bloque de motor de registro asociado con la pila para formar una trama;

* los datos de proceso constan, además, de además de bloques de memoria para la evolución del proceso; y

* la unidad de procesamiento está dispuesta para implementar la ejecución multihilo.

Surgirán otras características y ventajas de la invención a la luz de la descripción que sigue, dada con fines ilustrativos y no limitativos sobre la base de ejemplos extraídos de los dibujos en los que:

- la figura 1 representa un espacio de memoria asociado a un proceso convencional;
- la figura 2 representa un gestor de procesos según la invención en su entorno;
- la figura 3 representa un espacio de direccionamiento de memoria asociado a un superproceso por parte del gestor de procesos de la figura 2; y
- la figura 4 representa un diagrama de flujo de una ejecución de un proceso.

Como se mencionó anteriormente, para ser ejecutado, un proceso convencional se almacena en una memoria (generalmente memoria RAM), cuyos derechos de acceso se almacenan en un registro de hardware del procesador.

Este registro está protegido físicamente, es decir, el procesador implementa reglas de acceso que aseguran que el proceso asociado a un espacio de memoria dado sea el único capaz de modificar ese espacio.

En los sistemas informáticos modernos, este espacio se denomina espacio de direccionamiento y se organiza como se muestra en la figura 1.

Como se puede ver en la figura 1, un espacio de direccionamiento 2 consta de una sucesión de bits de memoria que están asociados a bloques funcionales distintas.

El espacio de direccionamiento 2 consta de un bloque de trama 4 que consta de una pila 6, un bloque 8 de memoria libre, y un bloque de motor de pila de registro 10 (RSE). En la trama 4, las flechas indican la dirección de apilamiento de los datos en los bloques respectivos.

La pila 6 se usa para apilar datos de funciones a medida que una función llama a otra, etc. Como cualquier pila, funciona según un principio de FILO, es decir, que la primera instrucción introducida es la última en ser recuperada. Esta pila permite, por lo tanto, almacenar variables locales y ejecutar las instrucciones del proceso de forma secuencial.

El bloque 8 sirve para proporcionar un espacio de extensión a la pila 6 y al bloque de motor 10, para permitir que crezca en el caso de que su tamaño exceda el tamaño asignado para crear el espacio de direccionamiento asociado al proceso.

El bloque de motor 10 es un elemento opcional que está presente solo en algunas arquitecturas de hardware. Su función es almacenar el contenido de los registros del procesador cuando una función llama a otra, etc. El bloque 12 cumple la misma función que el bloque 8, pero esta vez frente al bloque de motor 10.

Por consiguiente, el bloque de motor 10 permite que, cuando una primera función llama una segunda función, la primera función puede acceder rápidamente a los datos de esta segunda función. Esta implementación en particular se implementa, por ejemplo, en los procesadores Itanium (marca registrada).

En ausencia del bloque de motor 10 y la arquitectura del procesador correspondiente, la primera función debe acceder a la pila asociada a la segunda función, luego apila esos datos en su propia pila para manipularlos.

Se pueden hacer otros usos del bloque de motor 10, en particular para recibir datos dinámicos dependientes del procesador. Éste también puede estar totalmente ausente. Los datos que este bloque almacena son los mismos que los datos para las llamadas de función.

El espacio de direccionamiento 2 comprende entonces un bloque 12 de memoria libre y un bloque de mapeo 14 que forma un grupo de mapeo 15. El bloque 12 cumple la misma función que el bloque 8, pero aquí en relación con el bloque de mapeo 14.

El bloque de mapeo 14 se usa para recibir en la memoria de trabajo del procesador datos de la memoria de solo lectura del ordenador. Este bloque de datos se sincroniza regularmente con el archivo correspondiente en la memoria de solo lectura, de modo que cualquier cambio realizado en el bloque de memoria de trabajo también se refleje en el archivo correspondiente.

Siguiendo el grupo de mapeo 15, el espacio de direccionamiento 2 consta de un bloque 16 de memoria libre y un montón 18 que forma un grupo de montones 19. El bloque 16 cumple la misma función que el bloque 8, pero esta vez frente al bloque de montón 18.

El montón 18 se utiliza para recibir datos que el proceso decide crear, como variables globales, por ejemplo. A diferencia de la pila 6, este bloque de memoria permanece asignado incluso después del final del proceso, hasta que haya recibido una instrucción de liberación.

Esto significa que no se puede acceder a este bloque de memoria para escribir otros datos hasta que se haya liberado, es decir, hasta que la memoria haya recibido instrucciones de que este bloque ya no se usa.

El espacio de direccionamiento 2 consta de un bloque BSS 20 que sigue al grupo de montones 19. El bloque BSS 20 se utiliza para recibir datos que se inicializan a 0 al inicio de la ejecución del proceso.

Finalmente, después del bloque BSS 20, el espacio de direccionamiento consta de un bloque de datos 22. El bloque de datos 22 recibe los datos contextuales de ejecución del proceso, a partir del archivo ejecutable.

Se observará que lo anterior representa un modo de realización particular de un espacio de direccionamiento. De este modo, ciertos bloques, como el bloque motor 10 puede omitirse, y otros pueden tener un papel diferente.

Por ejemplo, los bloques 8, 12 y 16 se muestran aquí, respectivamente, como los bloques 6 y 10, 14 y 18. Sin embargo, en otras implementaciones, estos bloques no están necesariamente unidos a otros bloques. Estos bloques sirven como memoria libre que puede llenarse en función de la evolución del proceso, cualquiera que sea el bloque original.

Como se mencionó anteriormente, no es posible compartir los datos de un proceso con otro proceso. De hecho, el paso datos de un proceso a otro requiere dos operaciones para copiar los datos a compartir.

Es necesaria una primera operación de copia para rastrear estos datos en un recurso global del sistema operativo. A continuación, una segunda copia es necesaria. Esta vez, el segundo proceso buscará los datos en el recurso global del sistema que se acaba de crear, para que pueda manipular los datos compartidos.

Esto es extremadamente caro en términos de rendimiento, ya que los tiempos de acceso a la memoria son prohibitivos en comparación con la velocidad del procesador.

El gestor de procesos que representado en la figura 2 hace posible eludir las protecciones físicas entre procesos y, por lo tanto, ofrece la posibilidad de evitar costes innecesarios de acceso a la memoria.

Como se puede ver en la figura 2, un sistema operativo 30 consta de una aplicación 32 para ejecutar los procesos.

El sistema operativo 30 puede ser de cualquier tipo, como, por ejemplo, de la familia Windows (marca registrada), de la familia Unix (marca registrada) o la familia Linux (marca registrada) u otro tipo de sistema operativo similar. En el caso de una familia Windows (marca registrada), la aplicación 32 se llama "explorer", y en el caso de las familias Unix (marca registrada) y Linux (marca registrada), la aplicación 32 se llama "shell".

En un modo de realización preferente de la invención, es el sistema operativo Linux y su aplicación "shell" se utilizan.

Para ejecutar un proceso 34 en el ámbito de un sistema operativo convencional, la aplicación 32 llamaría a las funciones de carga del sistema operativo para cargar el proceso 34 en la memoria para su ejecución.

Por cargar, se entienden todas las operaciones necesarias para definir el espacio de direccionamiento asociado a un proceso dado, la definición correspondiente de los derechos de acceso en el procesador y la programación de la ejecución de este proceso entre los procesos a ejecutar.

En el ámbito de la invención, la aplicación 32 llama a funciones de carga separadas. Desde un punto de vista práctico, estas funciones cumplen sustancialmente la misma función que las funciones de carga convencionales, pero definiendo los procesos de manera diferente y utilizando un gestor de procesos 36 cuya función se explicará ahora.

El gestor de procesos 36 es un componente de software agregado en el contexto de la invención y que no existe en los sistemas operativos convencionales. Los procesos 34 que son procesados por el gestor de procesos 36 se

llamarán procesos incidentes a partir de entonces.

De este modo, a diferencia del caso convencional en donde se asigna un espacio de direccionamiento privada al proceso 34, el gestor 36 procesa los procesos de incidentes de manera selectiva y controlada, asociándolos a uno o varios procesos diferentes incidentes en ejecución o pendientes, y cargándolos todos juntos en un espacio de direccionamiento que se describirá con la figura 3.

Los procesos cargados en el mismo espacio de dirección pueden compartir de este modo datos previamente privados, manteniendo algunos de los datos limpios, como aparecerá a continuación.

En el modo de realización descrito en este caso, el gestor de procesos 36 es un demonio (daemon) que recibe solicitudes para la ejecución de las funciones de carga llamadas por la aplicación 32, y que a su vez garantiza la carga de procesos incidentes en espacios de direccionamiento compartidos.

El gestor de procesos 36 consta de una unidad de procesamiento 38 y una función de agrupación 40.

La unidad de procesamiento 38 tiene la función de crear superprocesos a petición de las funciones de carga. Un superproceso es un proceso cuyo espacio de direccionamiento se dispone para contener los datos del proceso de incidentes asociados a él.

En el ámbito de su asociación con un superproceso, los procesos incidentes 34 se llaman procesos elementales, ya que, aunque sus datos son idénticos a los procesos incidentes de los que se deriva, no poseen espacio de direccionamiento privado asociado como en el caso de los sistemas operativos convencionales.

La unidad de procesamiento 38 también tiene la función de ejecutar el código de los procesos elementales asociados a los superprocesos. Esta unidad también puede implementar la ejecución multihilo de procesos elementales.

La función de la agrupación 40 es asociar un proceso de incidente 34 con un superproceso. Para ello, la función de agrupación debe cargar un proceso incidente 34 en el espacio de direccionamiento 42 del superproceso asociado en la memoria de acceso aleatorio (RAM) 44.

La función de agrupación 40 no decide la asociación entre un proceso incidente y un superproceso: son las funciones de carga las que indican a qué superproceso debe asociarse un proceso incidente determinado.

Los superprocesos son procesos similares a los procesos 34, a excepción de su espacio de direccionamiento 42, que ahora se describirá por medio de la figura 3.

Como se puede ver en la figura 3, se representa el espacio de direccionamiento 42 de un superproceso al que están asociados dos procesos elementales.

El espacio de direccionamiento 42 consta de una trama para cada proceso elemental, referenciada respectivamente 50 y 51. Las tramas 50 y 51 son similares a la trama 6 de la figura 1, y cada una consta de una pila, un bloque de memoria 8, y un motor de pila de registro 10.

Las tramas 50 y 51 son seguidos por grupos de mapeo 52 y 53 similares al grupo de mapeo 15 de la figura 1 y cada uno asociado a uno de los dos procesos elementales. Los grupos de mapeo 52 y 53 constan cada uno de un bloque de memoria 12 y de un bloque de mapeo 14.

Siguiendo los grupos de mapeo 52 y 53, el espacio de direccionamiento 42 consta de dos estructuras 54 y 55 cada una asociada con uno de los procesos elementales, comprendiendo cada estructura:

- un grupo de montones similar al grupo de montones 19 en la figura 1, que comprende un bloque de memoria 16 y un montón 18;
- un bloque BSS similar al bloque BSS 20 de la figura 1; y
- un bloque de datos similar al bloque de datos 22 de la figura 1.

De este modo, el espacio de direccionamiento 42 consta efectivamente de todos los datos de los dos procesos elementales. Dentro de este espacio de direccionamiento, cada proceso elemental posee un espacio de direccionamiento propio para almacenar los datos globales, que son referenciados 54 y 55 en la figura 3.

Esto permite su ejecución a través de su pila respectiva, al mismo tiempo que es posible compartir datos entre ellos sin acceso a la memoria asociada.

Por otra parte, los datos de los procesos elementales están totalmente presentes por separado. Esto asegura que los procesos posean un espacio de memoria limpio tan pronto como se cargan, sin que una aplicación tenga que reorganizar sus datos.

Los procesos pueden así ejecutarse en interacción separada con sus datos, mientras se tiene acceso a los datos de otros procesos presentes en el espacio de direccionamiento que evita el problema de doble copia mencionado anteriormente.

La figura 4 muestra un diagrama de flujo que ilustra el procesamiento de un proceso dado, desde el sistema operativo hasta el procesador.

En un primer momento, la aplicación 32 lanza una función Launch() con como argumento, un proceso Proc en una etapa 60.

La función Launch() es una de las funciones de carga implementadas en el ámbito de la invención, y su función es recuperar los datos del proceso Proc en los datos Proc.Dat.

Estos datos son de dos naturalezas funcionales: datos de ejecución, es decir, el código real a ejecutar, y los datos contextuales, característicos del contexto de ejecución del proceso, como las variables de una función.

Cuando se recuperan los datos del proceso Proc, se llama a una función Assign() con como argumento, los datos Proc.Dat en una etapa 62.

Esta función se utiliza para asignar el proceso de incidente Proc a un superproceso existente, o para crear un superproceso para recibirlo. El resultado de esta función es un par (Proc.Dat., SProc) que asocia el proceso incidente a su superproceso.

Una vez que la función Assign() está terminada, se llama a una función Instruct() con como argumento, el par (Proc.Dat., SProc.) en una etapa 64.

La función Instruct() tiene la función de transmitir el par de proceso incidente/superproceso al gestor de procesos 36.

En el ámbito de la invención, la función Instruct() registra el par (Proc.Dat., SProc.) en una parte compartida de la memoria del sistema operativo que se llama memoria Shared Memory (SHM), que el gestor de procesos 36 viene regularmente a consultar para descubrir y procesar los nuevos procesos elementales a ejecutados.

Otras implementaciones serían posibles para transmitir los procesos incidentes al gestor de procesos 36, por ejemplo, por comunicaciones de mensajes del tipo de send/receive (enviar /recibir) disponibles en los sistemas operativos.

A continuación, el proceso elemental se ejecuta en una etapa 66, como se describió anteriormente, llamando a la unidad de procesamiento 38 para crear un superproceso si es necesario, y luego llamando a la función de agrupación 40 para cargar el proceso elemental en el espacio de direccionamiento del superproceso.

Aunque las funciones 60, 62 y 64 se han descrito como funciones separadas, sería posible realizarlas de manera diferente, reuniendo a dos de ellas, o agrupándolas en una sola función.

Aunque la presente invención se ha descrito con referencia a modos particulares de realización, el experto en la materia podrá considerar todas las realizaciones equivalentes particulares que se encuentran dentro del alcance de las siguientes reivindicaciones.

La invención se refiere igualmente, como productos, los elementos de software descritos, disponibles bajo cualquier "medio" (soporte) legible por ordenador. La expresión "medio legible por ordenador" comprende medios de almacenamiento de datos, magnéticos, ópticos y/o electrónicos, así como un soporte o vehículo de transmisión, como una señal analógica o digital.

REIVINDICACIONES

1. Gestor de procesos informáticos, que comprende:

- una memoria de datos de proceso (44), y
- una unidad de procesamiento de proceso (38), capaz de ejecutar un código de proceso que interactúa con una parte designada de la memoria de datos de proceso,

siendo la unidad de procesamiento de procesos capaz de establecer un superproceso asociado a un espacio de direccionamiento (42) designado de la memoria (44), **caracterizado por que** comprende, además:

- una función de agrupación (40) dispuesta para cargar procesos incidentes (34) adecuados para ser ejecutados, cada uno, como un proceso en un espacio de direccionamiento limpio, en el espacio de direccionamiento del superproceso en respuesta a una condición relacionada, al menos en parte, con estos procesos incidentes, de modo que el espacio de direccionamiento del superproceso aloja datos de los procesos incidentes (34) que constan para cada proceso incidente de una trama (50, 51) seguida de un grupo de mapeo (52, 53) y una estructura (54, 55) para almacenar sus datos,

y **por que** la unidad de procesamiento de proceso (38) es capaz de ejecutar el código de cada uno de los procesos incidentes (34) interactuando por separado con sus datos.

2. Gestor de procesos según la reivindicación 1, **caracterizado por que** el espacio de direccionamiento del superproceso dado contiene el conjunto de los datos de procesos de todos los procesos incidentes que se asocian con el mismo.

3. Gestor de procesos según la reivindicación 1 o 2, **caracterizado por que** la condición de alojamiento de los procesos incidentes se relaciona con una lista recibida por el gestor de procesos que asocia un identificador de un proceso incidente a un identificador de un superproceso.

4. Gestor de procesos según la reivindicación 3, **caracterizado por que** el identificador de un proceso incidente está asociado con un identificador de un superproceso en función de al menos una parte del código del proceso.

5. Gestor de procesos según una de las reivindicaciones anteriores, **caracterizado por que** está implementado en forma de un demonio (36).

6. Gestor de procesos según una de las reivindicaciones anteriores, **caracterizado por que** está implementado en un sistema operativo Linux.

7. Gestor de procesos según una de las reivindicaciones anteriores, **caracterizado por que** los datos de proceso constan de una pila (6), un montón (18), un bloque BSS (20) y un bloque de datos (22).

8. Gestor de procesos según la reivindicación 7, caracterizado por que los datos de proceso constan, además, de un bloque de motor de registro (10) asociado con la pila (6) para formar una trama (4).

9. Gestor de procesos según la reivindicación 7 u 8, caracterizado por que los datos de proceso constan, además, de además de bloques de memoria (8, 12, 16) para la evolución del proceso.

10. Gestor de procesos según una de las reivindicaciones anteriores, **caracterizado por que** la unidad de procesamiento (38) está dispuesta para implementar una ejecución multihilo de los procesos incidentes alojados en un superproceso.

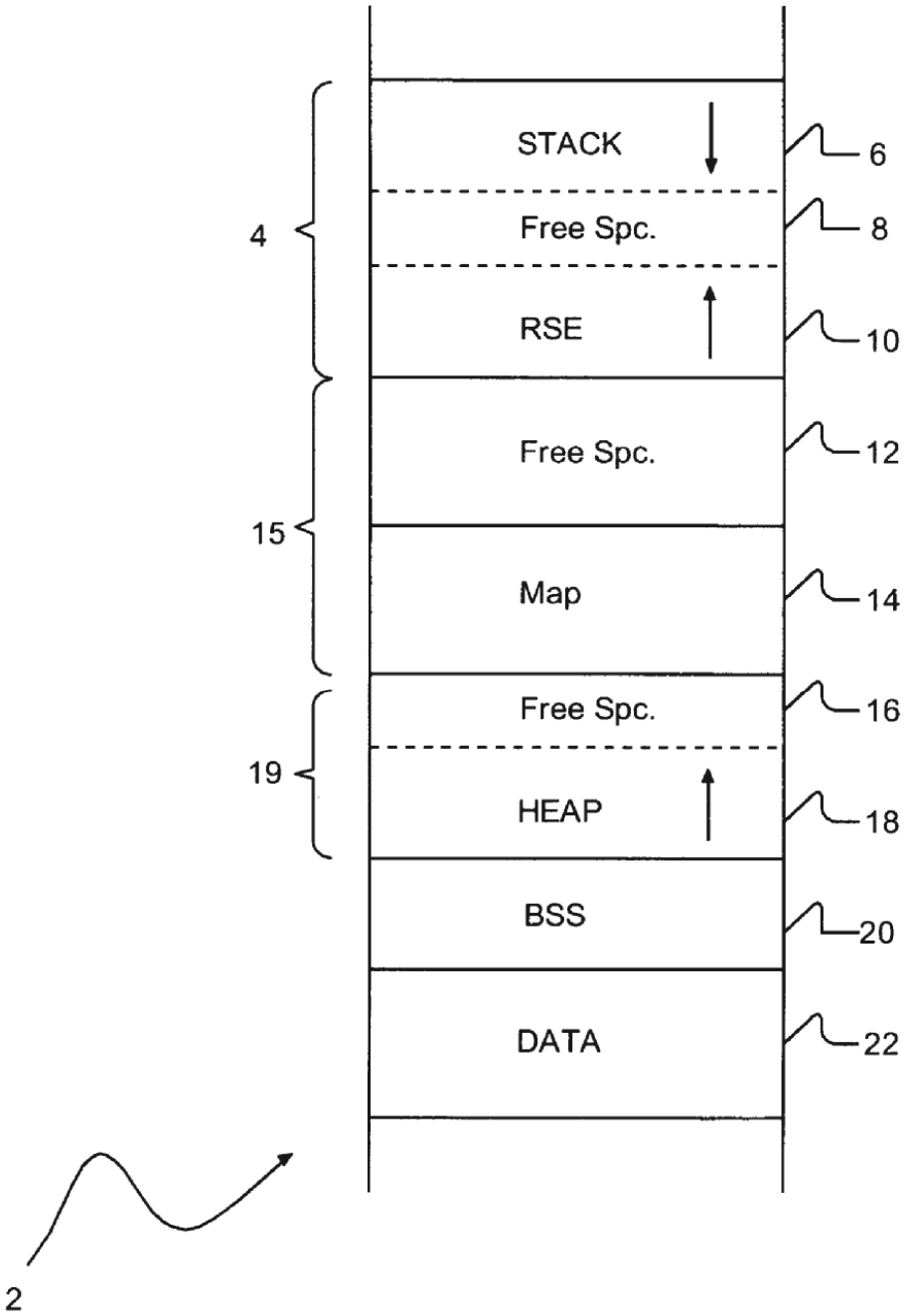
STACK
Free Spc.
RSE
Free Spc.
Map
Free Spc.
HEAP
BSS
DATA
4
15
19
2
6
8
10
12
14
16
18
20
22

Fig.1

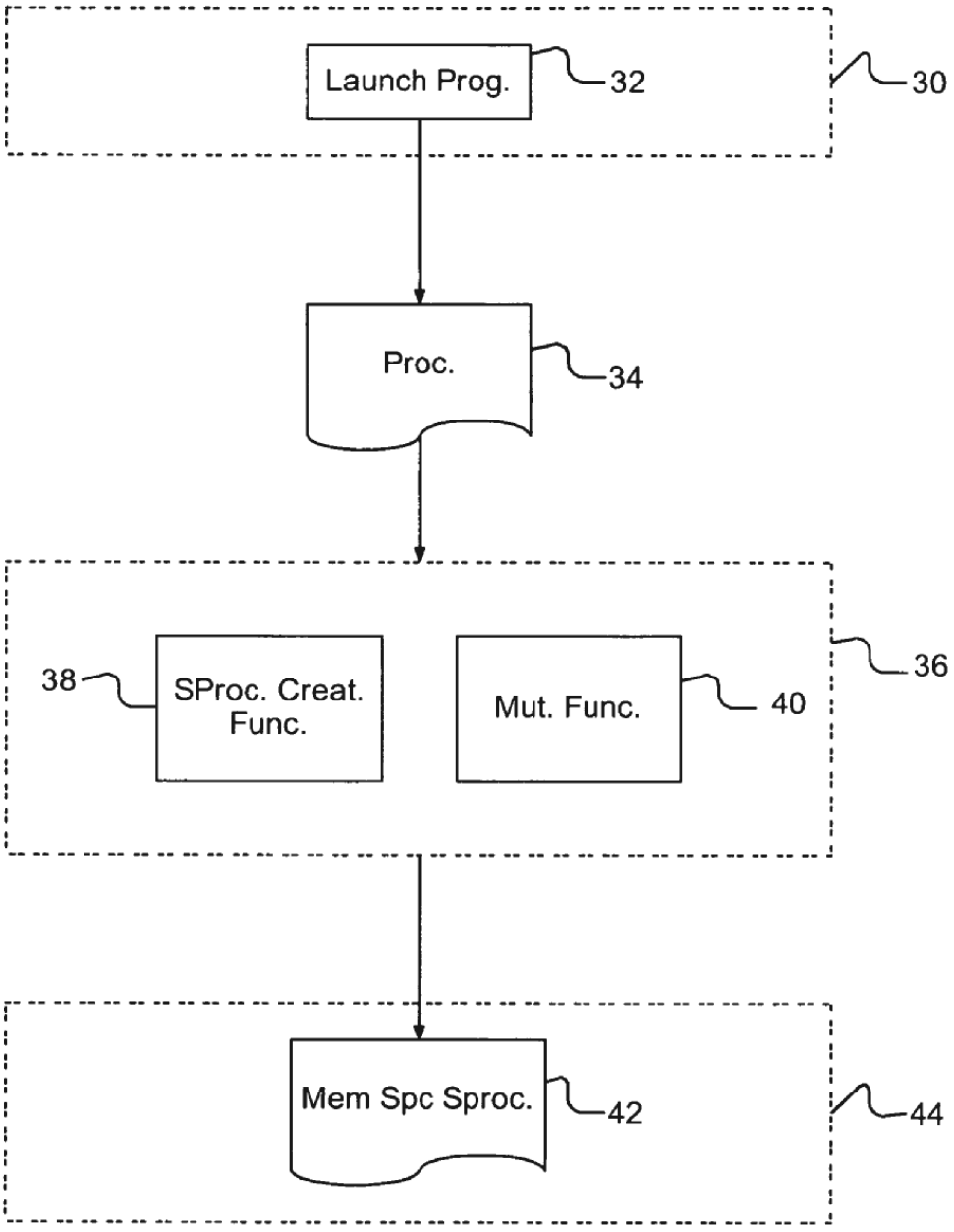
Launch Prog.
32
30
Proc.
34
38
SProc. Creat. Func.
Mut. Func.
40
36
Mem Spc Sproc.
42
44

Fig.2

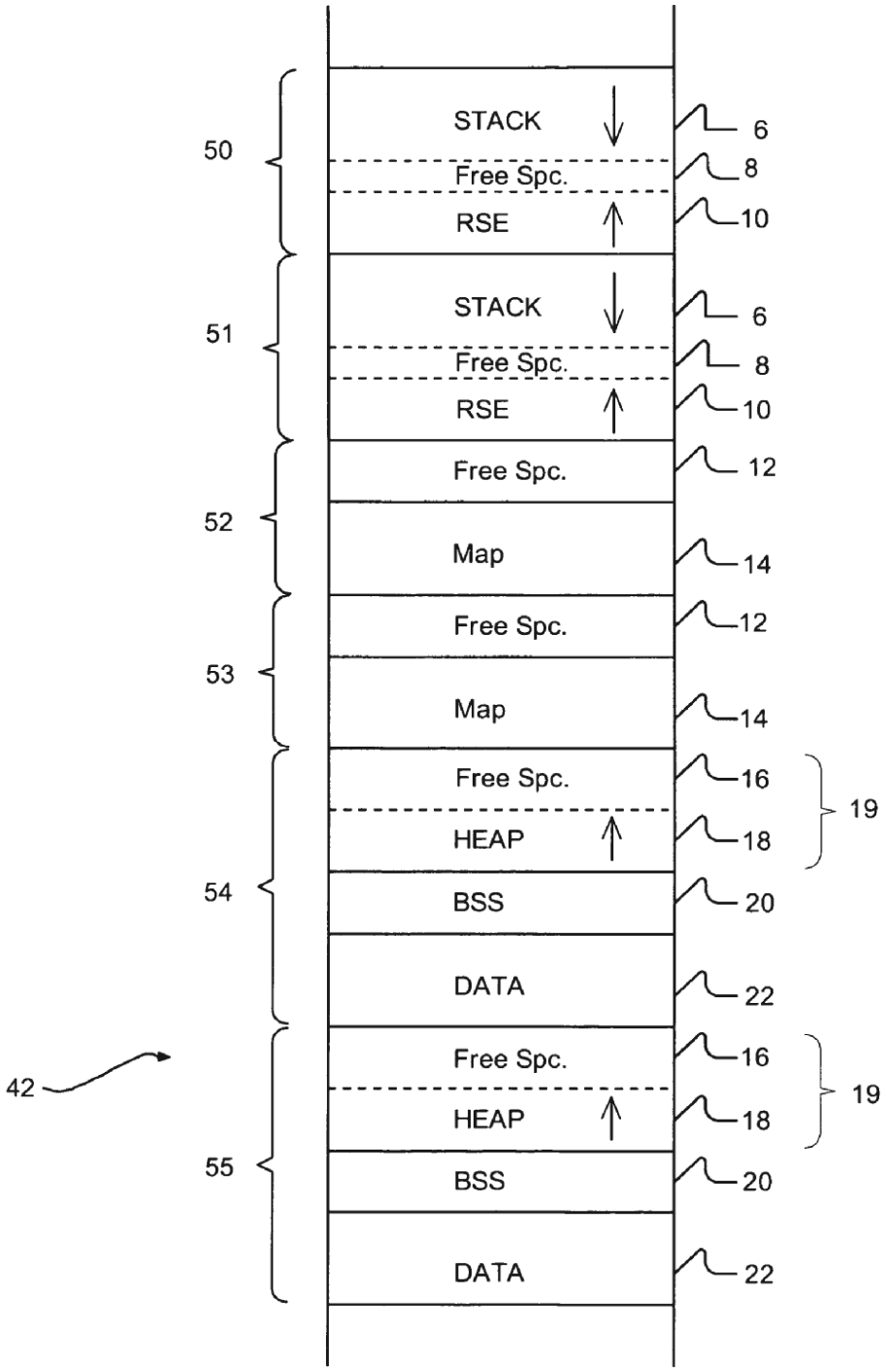
STACK
Free Spc.
RSE
STACK
Free Spc.
RSE
Free Spc.
Map
Free Spc.
Map
Free Spc.
HEAP
BSS
DATA
Free Spc.
HEAP
BSS
DATA
50
51
52
53
54
55
42
6
8
10
6
8
10
12
14
12
14
16
18
19
20
22
16
18
19
20
22

Fig.3

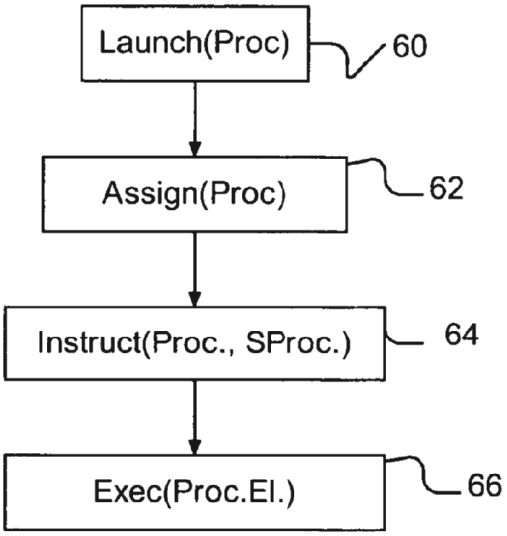
Launch(Proc)
60
Assign(Proc)
62
Instruct(Proc., SProc.)
64
Exec(Proc.El.)
66

Fig.4