



Sistemas Operativos 2

Unidad 1: Administración de memoria

Asignación y liberación

René Ornelis
julio de 2025

Contenido

1	Operaciones sobre el manejo de la memoria	4
2	Particiones fijas.....	5
3	Particiones variables	6
4	Métodos de Ajuste secuencial (Sequential fit).....	7
4.1	Proceso general	7
4.1.1	Asignación de un bloque de tamaño n	7
4.1.2	Liberación	8
4.2	Primer ajuste	8
4.3	Mejor ajuste	8
4.4	Peor ajuste.....	9
4.5	Óptimo ajuste	9
4.6	Optimizaciones	9
4.7	Ordenamiento de la lista de bloques libres	10
4.8	Apuntador Vagabundo	10
5	Sistemas compañero.....	10
5.1	Fibonacci.....	14
6	Compactación	14
7	Colección de basura	15

Índice de figuras

Figura 1: Asignación de memoria con particiones fijas.....	5
Figura 2: Efecto de las particiones dinámicas.....	6
Figura 3: Bloque de memoria en métodos secuenciales	7
Figura 4: Ejemplo de sistemas compañero	12
Figura 5: Representación de árbol de sistemas compañeros.....	13
Figura 6: Compactación de memoria.....	14

Asignación y liberación de memoria

1 Operaciones sobre el manejo de la memoria

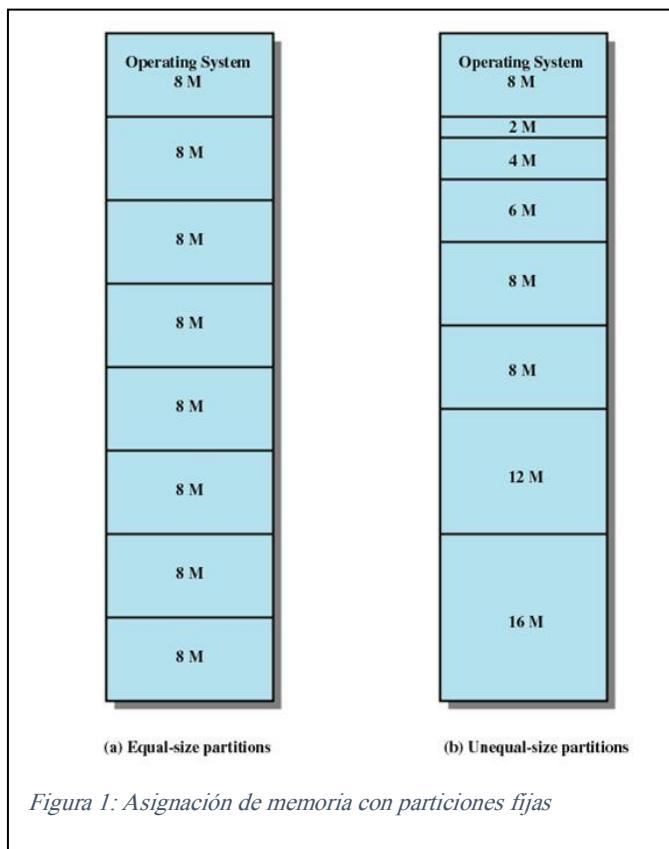
El sistema operativo, al realizar el manejo de memoria, realiza las siguientes operaciones sobre la misma:

- **Asignación de memoria:** entregar a los procesos bloques de memoria solicitados con distintos tamaños
- **Liberación de memoria:** recibir de los procesos los bloques de memoria que éstos ya no utilizarán y dejarlos como parte de la memoria libre que podrá ser utilizada por otros procesos.
- **Compactación:** cuando se produce *fragmentación externa*, todos los bloques de memoria se mueven hacia un extremo de la memoria, de modo que queda sólo un bloque libre al final

Estas operaciones pueden ser realizadas a nivel de sistema operativo o a nivel de aplicación. Para requerimientos de bloques pequeños, si se invoca al sistema operativo, provoca *fragmentación interna*. Por lo que los compiladores normalmente generan un segmento de HEAP para manejar internamente (a nivel de librerías de usuario) el uso de la memoria dinámica. Esta es la operación del sistema operativo en el cual un proceso solicita un bloque de memoria de tamaño K y el sistema le debe proporcionar la dirección de un bloque de memoria para uso del proceso. Dicho bloque debe tener al menos el tamaño solicitado (K).

2 Particiones fijas

Para realizar esta operación, anteriormente se utilizaban métodos que utilizaban bloques de *tamaño uniforme*, los cuales consistían en dividir la memoria en un número de bloques, todos del mismo tamaño o en grupos de diferente tamaño (ver Figura 1).



Estas políticas, aunque son simples de implementar, ya que, al ser todas las particiones iguales, no importa que partición se le asigna al proceso; rápidamente cayeron en desuso, debido a que la programación de los procesos se complicaba, debido a los casos en que, si se tienen bloques de tamaño N , y se necesita un bloque de memoria $2N$, el programador debía tener en mente que, al solicitar dos bloques, en muchos casos no había garantía que los bloques entregados sean continuos, y aunque así fuera, siempre existía el problema de tener que calcular los requerimientos en términos de bloques y no de bytes, además de tener que utilizar librerías para trabajar sobreposiciones (*overlay*) y usar más de una partición

Más determinante que las complicaciones de programación, fue la *fragmentación interna*

FRAGMENTACIÓN INTERNA: Es desperdicio de memoria que existe dentro de un bloque de tamaño K , que se entrega a un proceso que solicitó un bloque de tamaño N y $K < N$. Existe un desperdicio de memoria *dentro* de bloque de tamaño $N-K$

Tal como se muestra en la Figura 1, para combatir la fragmentación interna se utilizó una variación de esta política que consistía en tener grupos de bloques de diferentes tamaños predefinidos, así, cada requerimiento será asignado a la partición más pequeña que pueda contenerlo.

3 Particiones variables

Actualmente, los sistemas utilizan métodos de bloques de *tamaño variable*, los cuales dividen la memoria en bloques cuyo tamaño se define conforme los procesos requieren y liberan memoria, garantizando que cada bloque atendido, sea uno sólo de memoria contigua.

Estos métodos resultan mucho más eficientes en términos de utilización de la memoria, sin embargo, varían en complejidad, rendimiento, manejo de la fragmentación interna y *fragmentación externa*.

FRAGMENTACIÓN EXTERNA: Es cuando en la memoria existen pequeños bloques de memoria libre, intercalados entre los bloques utilizados por procesos, de modo que, aunque la memoria libre total sea de tamaño N (la suma de todos los bloques libres), el sistema no puede atender requerimientos de dicho tamaño, y aún tamaños menores, dado que está distribuido a lo largo de toda la memoria.

La fragmentación externa se da por la dinámica de los procesos de solicitar y liberar la memoria, por lo que eventualmente los bloques de memoria libre quedarán distribuidos a lo largo de toda la memoria.

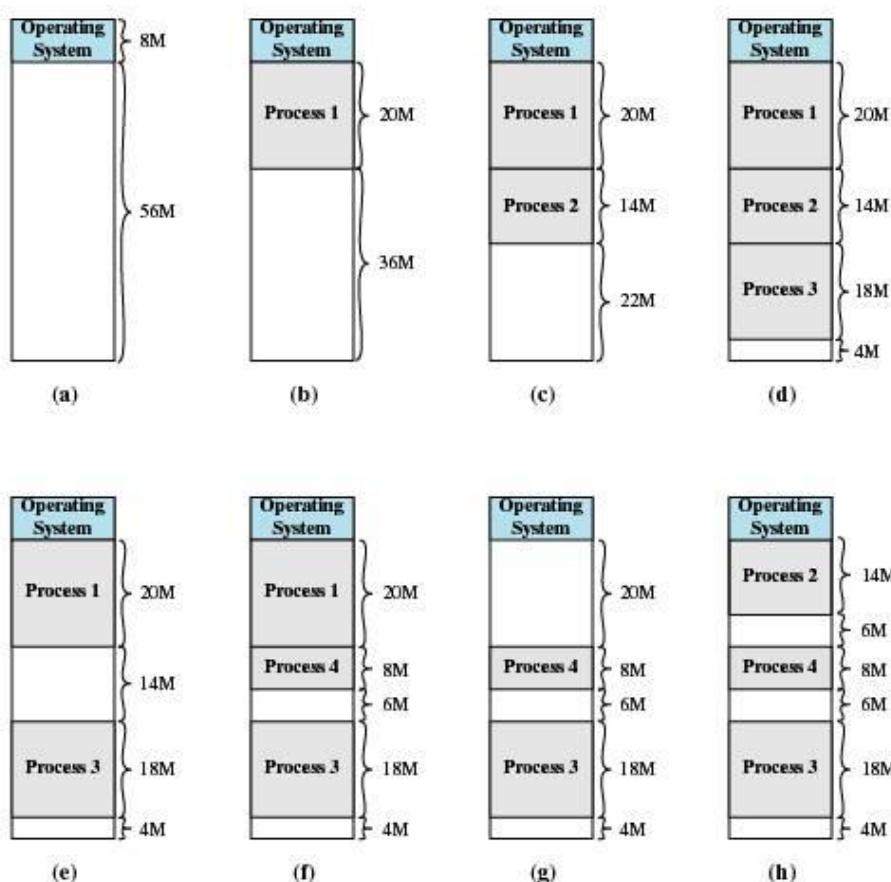


Figura 2: Efecto de las particiones dinámicas

La solución a este problema es aplicar la compactación y/o recolección de basura, que se debe estar realizando constantemente.

En los métodos actuales, de tamaño variable, al liberar memoria, si el

bloque liberado es adyacente a uno que estaba libre, estos bloques se unen para formar un bloque más grande, de tamaño igual a la suma de los tamaños de los bloques mezclados. De este modo, al liberarse toda la memoria, sólo existirá un bloque libre, del tamaño de la memoria total.

Los métodos que estudiaremos son:

- Ajuste secuencial
- Sistemas compañero

4 Métodos de Ajuste secuencial (Sequential fit)

Este es un conjunto de métodos que se basa en una lista de bloques libres mantenida dentro de la memoria disponible.

4.1 Proceso general

4.1.1 Asignación de un bloque de tamaño n

- Se recorre la lista de bloques libres en busca de un bloque de tamaño k, *según alguna política*, donde $k \geq n$.
- se marca RESERVADO y se desenlaza de la lista de libres.
- Si $k > n$, se puede decidir si la fragmentación será interna o externa (no se divide o se divide).
- Si se divide se enlaza el bloque restante como un nuevo bloque libre en la lista.

El criterio para dividir un bloque puede ser un parámetro que indique el máximo de fragmentación interna.

Este parámetro no puede ser menor que el tamaño de las banderas y apuntadores necesarios.

Las políticas que se pueden aplicar para seleccionar el bloque a trabajar pueden ser:

- Primer ajuste
- Mejor ajuste
- Peor ajuste
- Óptimo ajuste

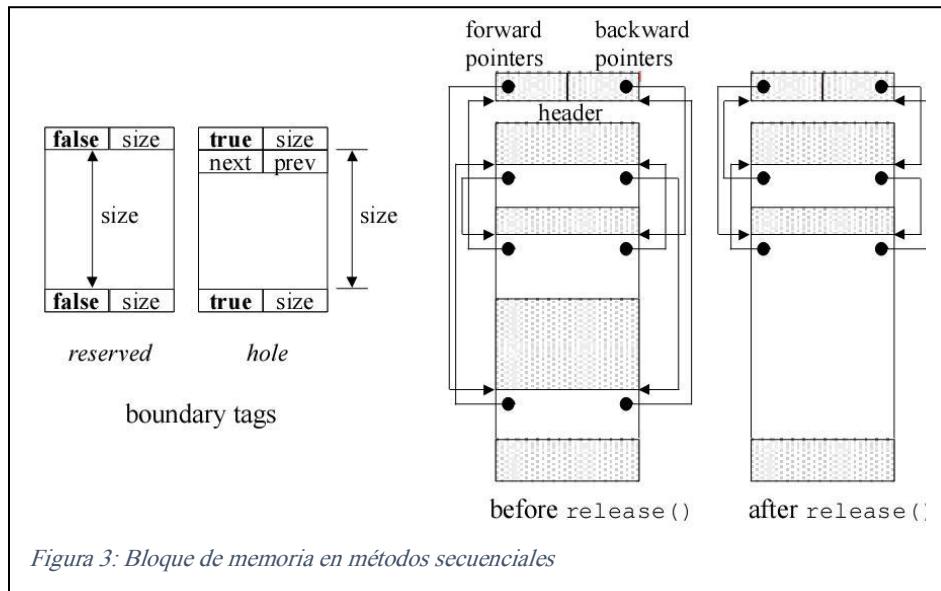


Figura 3: Bloque de memoria en métodos secuenciales

4.1.2 Liberación

- Se marca el bloque como LIBRE
- Mientras uno de los vecinos (izquierdo o derecho) estén libres
 - El bloque se combina con sus vecinos libres para formar un sólo bloque, cuyo tamaño será la suma del tamaño de los bloques combinados

Este proceso permite que cuando se liberen todos los bloques ocupados, sólo existirá un bloque de memoria, cuyo tamaño será el de toda la memoria disponible.

4.2 Primer ajuste

Se toma el primer bloque tal que pueda cumplir con el requerimiento, es decir, para un requerimiento de tamaño n , retorna el primer bloque de la lista de tamaño k , donde $k \geq n$. En el peor caso se recorre toda la lista.

Ventajas:

- Resulta bastante simple de implementar
- Es relativamente rápido en su desempeño, dado que las búsquedas en la lista, en promedio no se busca en toda la lista

Desventajas:

- Produce un alto índice de fragmentación externa, dado que los bloques tienden a dividirse constantemente

4.3 Mejor ajuste

Busca eliminar la fragmentación externa, se busca el bloque menor que cumpla con el requerimiento, es decir se toma el más pequeño de los bloques, cuyo tamaño $k \geq n$.

Ventajas:

- Salva los bloques de mayor tamaño para pedidos grandes
- Reduce la fragmentación interna

Desventajas:

- Implica que en cada requerimiento se realiza una búsqueda en toda la lista
- Tiende a fragmentar externamente

4.4 Peor ajuste

Se busca eliminar al máximo la fragmentación interna, buscando siempre el bloque más grande y dividiéndolo. Y además elimina gran parte de la fragmentación externa del mejor ajuste.

Ventajas:

- Reduce la fragmentación interna al máximo.

Desventajas:

- Implica que en cada requerimiento se realiza una búsqueda en toda la lista
- Tiende a fragmentar externamente

4.5 Óptimo ajuste

Esta política consiste en buscar en dividir la búsqueda en dos fases:

- Sobre una muestra de los bloques libres se aplica la política del mejor ajuste.
- Sobre el resto de la lista después de la muestra, se toma el primer bloque que sea mejor que el mejor de la muestra.

El tamaño de la muestra se toma como una muestra estadística y según algunos autores puede ser N/e (donde e es la constante de Euler).

Ventajas:

- El resultado de esta política produce una eficiencia intermedia entre primer ajuste (más rápido) y el mejor ajuste (menos fragmentación interna y externa)
- En el mejor de los casos no se tiene que recorrer la lista completa, y se devuelve un bloque que es un buen aproximado al mejor ajuste, y en el peor de los casos, se tiene que recorrer toda la lista, el resultado es igual al del mejor ajuste

Desventajas:

- Se debe conocer el número de bloques libres.
- Aumenta la fragmentación interna respecto al mejor ajuste.

4.6 Optimizaciones

En busca de mejorar el rendimiento de estos métodos, se puede aplicar dos optimizaciones importantes: ordenamiento de la lista por tamaño de bloques y/o apuntador vagabundo.

4.7 Ordenamiento de la lista de bloques libres

Cualquiera de estas políticas, a excepción del Ajuste Óptimo (ya que está diseñado para la lista sin un orden), se les puede agregar la variante de mantener la lista de bloques libres ordenada por el tamaño. Esto tendría los siguientes efectos:

- **Primer Ajuste:** reduce el tiempo de búsqueda, ordenado ascendente, no se tiene que recorrer toda la lista para determinar que no existe un bloque adecuado, y en promedio se recorre la mitad para satisfacer los requerimientos, ya que la búsqueda de un bloque de tamaño n implica búsqueda de todos los bloques de tamaño $k < n$. Si esta ordenado descendente, el efecto es el mismo que el peor ajuste, y se toma sólo el primer bloque
- **Mejor ajuste:** tiene el mismo comportamiento que el primer ajuste con orden en la lista. Ya no se tiene que recorrer por completo la lista.
- **Peor Ajuste:** Si se mantiene la lista ordenada descendente, en TODAS las asignaciones de memoria, sólo se necesita examinar el primer nodo de la lista para determinar, y si el tamaño k del primer bloque es mayor o igual n , se atiende el requerimiento con dicho bloque, en caso contrario se determina que no se puede atender la solicitud. Lo cual incrementa el rendimiento de la asignación de memoria, en forma inmejorable.

Sin embargo, el mantener una lista ordenada, implica un retardo en el procesamiento de las liberaciones de memoria, dado que al combinar bloques adyacentes e insertarlos en la lista, implica realizar una búsqueda para determinar el lugar adecuado del bloque combinado.

4.8 Apuntador Vagabundo

Otra variante a cualquiera de las políticas originales consiste en mantener circulando la cabeza de la lista a lo largo de la misma, según algún criterio, como colocar redireccionar la cabeza de la lista en el nodo anterior (el último de la lista, dado que es doblemente encadenada), o colocar la cabeza en el siguiente nodo del que se está entregando al proceso solicitante.

Los efectos de esta variante son:

- Evita que la fragmentación externa se acumule en un solo punto.
- Los bloques divididos son alcanzados “a la vuelta” y son los de mayor duración. Esto implica que aumenta la probabilidad de la combinación.

5 Sistemas compañero

Los sistemas compañero, al contrario de los métodos de ajuste secuencial, no se basan en listas, sino en cálculos de direcciones y divisiones por pares para manejar los bloques de memoria, por lo que el rendimiento es mejor que el Ajuste Secuencial, dado que no existen búsquedas en listas, las cuales son reemplazadas por cálculos matemáticos.

Los sistemas compañeros binarios, están basados en la propiedad de las direcciones de memoria, las cuales al final (como todo número en la computadora) es una potencia de dos. Este método, Al haber una solicitud de tamaño n , se procede a dividir el bloque de memoria por la mitad, hasta que se tenga un bloque de tamaño k , tal que $k \geq n$ y $k/2 < n$,

En este método, para satisfacer una solicitud, toma un bloque libre y lo divide sucesivamente en dos bloques compañeros, hasta que se tiene el menor bloque de tamaño k , que pueda satisfacer la solicitud de tamaño n ($k \geq n$).

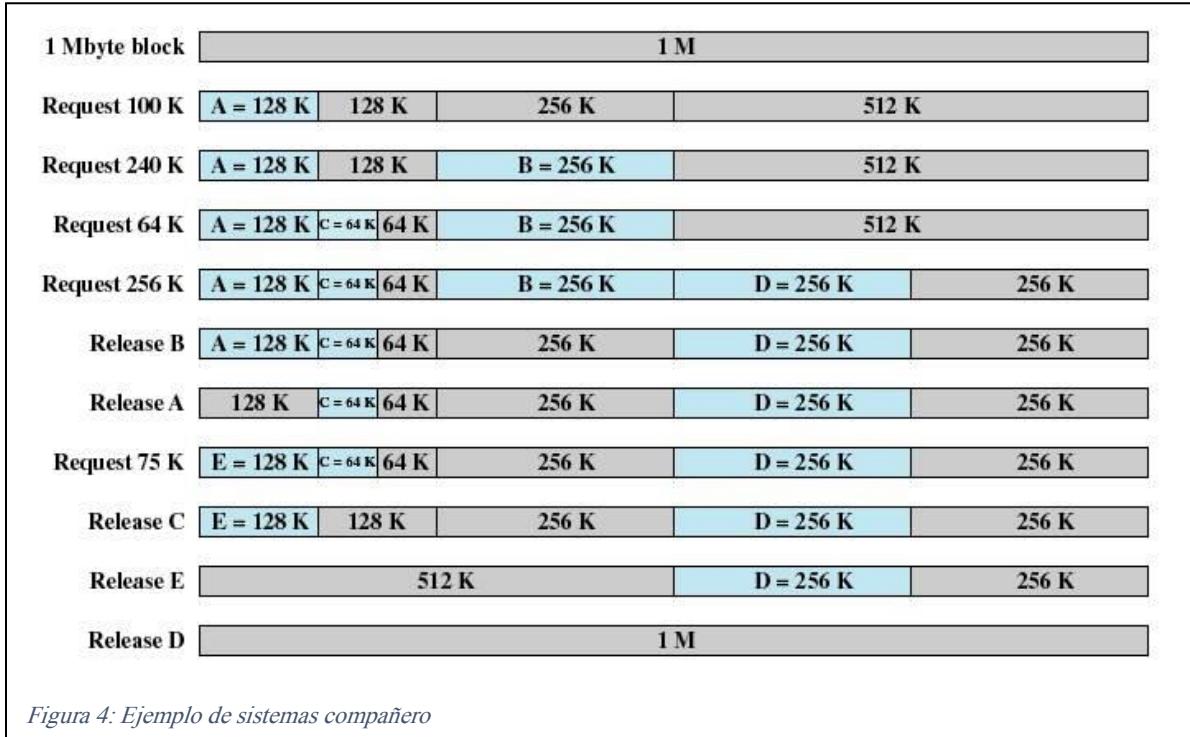
Bloques compañero: son dos bloques adyacentes, resultado inmediato de la división de un bloque mayor, que por lo tanto son del mismo tamaño

Al momento de realizarse una liberación de un bloque, procede a combinarlo con su bloque compañero, si este está libre. Obsérvese que dos bloques adyacentes libres, no necesariamente son compañeros, por lo que la liberación permite que existan dos bloques libres adyacentes sin combinar, si es que éstos no cumplen con la definición de bloque compañero, lo cual puede resultar en una deficiencia en la utilización de la memoria.

La asignación se realiza de la siguiente manera:

- El espacio de memoria se trata como un bloque de 2^U
- Para un requerimiento de tamaño s es requerido, tal que $2^{u-1} \leq s \leq 2^U$, entonces se entrega dicho bloque, de lo contrario, se divide el bloque en dos y se continua el proceso hasta que un bloque cumpla con $2^{k-1} \leq s \leq 2^k$

Ejemplo:



Una ventaja es que no se necesitan listas para recorrer los bloques, sino que puede usarse sólo aritmética binaria (*xor*) de las direcciones y tamaños. Por ejemplo supongamos un bloque total de memoria de 64 bytes, y las siguientes direcciones:

Bloque	A	B	C	D
Tamaño del bloque (decimal/hexadecimal/binario)	16/10/0010000	08/08/0001000	08/08/0001000	32/20/0100000
Dirección (decimal/hexadecimal/binario)	00/00/0000000	16/0F/0010000	24/18/0011000	32/20/0100000
Dirección del bloque compañero (decimal/hexadecimal/binario)	16/10/0010000	24/18/0011000	16/10/0010000	00/00/0000000

Podemos notar que se cumple que:

$$\text{Dir. Bloque compañero} = \text{Dir. bloque xor Tam.Bloque}$$

Dado que un bloque compañero es un bloque *del mismo tamaño, resultado de la división de un bloque mayor*, podemos calcular la dirección del compañero de cualquier bloque, y si dicho compañero es del mismo tamaño, entonces sí es su compañero. Puede ser que la dirección

calculada no corresponda a un bloque del mismo tamaño, debido a una división posterior, por lo tanto, ya no será su compañero.

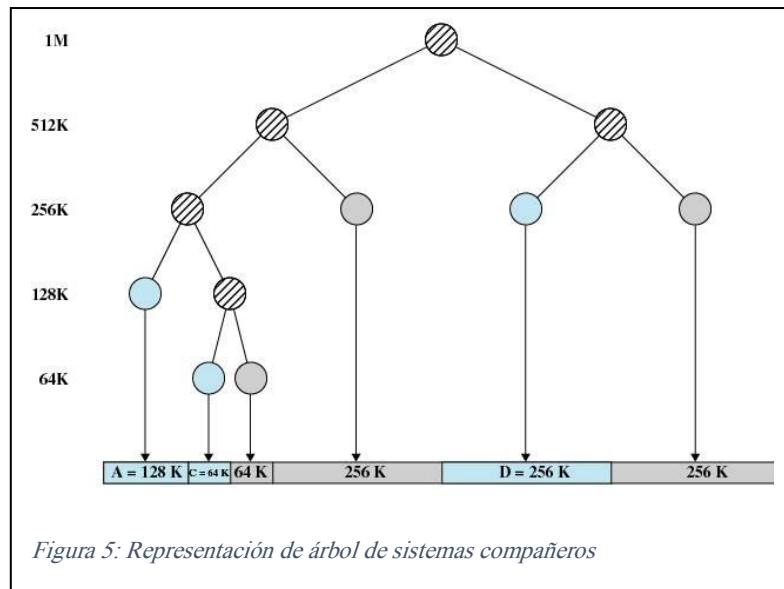
En la tabla anterior:

1. Vemos que la dirección del compañero de A es la dirección 10h, que corresponde a B. En este caso A y BC, fueron compañero, pero luego BC se dividió en B y C, por lo tanto, A y B ya no son compañeros, dado que no tienen en mismo tamaño, ni proceden de la misma división.
2. El mismo caso sucede con el compañero de D, que es la dirección 00h, correspondiente a A.
3. Sólo en el caso de B y C, se cumple con la definición de compañeros y el cálculo de la dirección de compañero, corresponde en ambos casos.

También se puede representar por medio de un árbol binario como se muestra en la Figura 5.

En esta representación, cada nodo puede representar dos casos:

1. Es un bloque dividido, por lo que es un nodo interno del árbol.
2. Corresponde a un bloque directo, por lo que es un nodo hoja, el cual puede estar siendo utilizado o está libre.



El recorrido de este árbol facilita la operatoria y el cálculo en los siguientes aspectos:

1. Cada nivel del árbol representa una división entre dos, por lo que el cálculo del tamaño del bloque es directo
2. Al liberar un bloque, si el bloque hermano está libre, entonces procede la unión con el compañero, y el nodo padre se convierte en un nodo hoja.

Ventajas:

- El cálculo direcciones de memoria, para determinar los bloques compañeros es directo ya que está basado en números binarios

Desventajas:

- Aumenta la fragmentación interna, dado que el tamaño de cada bloque es una potencia de 2
- No siempre se pueden unir los bloques libres adjuntos
- Una variedad de fragmentación externa, que consiste en que no se pueden unir dos bloques libres adyacentes.

5.1 Fibonacci

Este sistema, no realiza la división de los bloques por la mitad, sino que se basa en la serie de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ..) para realizar la división y combinación de bloques.

Por ejemplo: si un bloque de memoria es de 55 y se solicita un bloque de 10, el bloque de 55 se dividirá en un bloque de 21 y uno de 34, luego el de 21 se dividirá en uno de 8 y otro de 13 y se entregará este último al proceso solicitante

Ventajas:

- Dado que el tamaño de los bloques está determinado por la serie de Fibonacci en vez de potencias de dos, se reduce la fragmentación interna

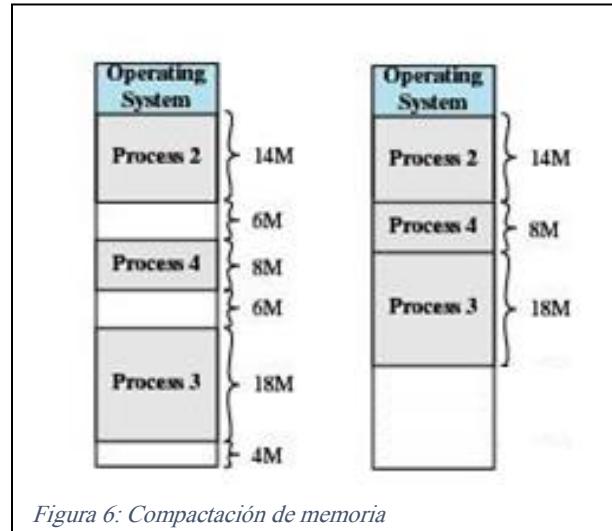
Desventajas:

- El cálculo de los bloques compañeros se complica, dado que se tiene que calcular constantemente los predecesores y sucesores en la serie de Fibonacci.

6 Compactación

Esta operación sobre la memoria se utiliza para eliminar la fragmentación externa. Básicamente consiste en mover hacia un extremo de la memoria todos los bloques ocupados, dejando en el otro extremo un sólo bloque libre, cuyo tamaño es la suma de los tamaños de los bloques libres existentes antes de la compactación, por lo que aumenta la disponibilidad de un bloque de memoria más grande (Ver Figura 6).

Esta operación supone un acceso exclusivo a la memoria, por lo que todos los procesos quedarán bloqueados hasta terminar la compactación, lo cual tiene un impacto directo en el rendimiento del sistema. Esto obliga a realizar un balance entre rendimiento y espacio libre: si se realiza frecuentemente, habrá más memoria disponible, pero el costo es una degradación del rendimiento y viceversa. Para que



pueda realizarse la compactación es necesario que el sistema tenga la capacidad de **relocalización**.

La compactación es una operación que el sistema puede elegir hacer en dos formas (posiblemente combinadas):

- **Por demanda:** Se invoca cuando hay un requerimiento de un bloque de un tamaño K tal que ninguno que los bloques libres puede satisfacer, y la suma del tamaño de todos los bloques libres es mayor que el requerimiento K
- **Anticipada:** Un proceso de fondo del sistema analiza periódicamente el estado de memoria y basado en un índice de fragmentación externa (por ejemplo: número de bloques libres / número de bloques ocupados) determina que un proceso (o los procesos de cierto tipo, o prioridad) se debe compactar.

En ambos casos, el alcance de la compactación se puede hacer a nivel **global** (involucra memoria de todos los procesos) o **local** (solo se compacta la memoria de un proceso).

7 Colección de basura

La colección de basura se utiliza en aquellos lenguajes donde no se obliga al programador a liberar la memoria utilizada, sino que el programa, automáticamente, recupera la memoria no utilizada cuando existe un requerimiento de memoria que no puede ser completado.

El algoritmo básico de la colección de basura es la siguiente:

- marcar todos los bloques de memoria como libres, ya sea que estén usados o no
- para todas las variables, marcar su bloque de memoria como ocupado
- los bloques de memoria que queden marcados como libres, se enlazan en la lista de libres.

Otra estrategia utilizada, es llevar contadores por área de memoria. Cada vez que se asigna un área de memoria a un apuntador, se incrementa el contador de la región, y al designar, se decrementa. De esta forma la colección de basura se reduce a eliminar las regiones de memoria que tienen su contador a cero.