Sistemas Operativos

Unidad 7: Gestión de la memoria.

"Nunca memorices algo que puedas buscar."

~ Albert Einstein

Gestión de la memoria

La gestión de la memoria es una tarea fundamental que debe realizar el sistema operativo. En un sistema monoprogramado, la memoria principal sólo se reparte entre el sistema operativo y el programa en ejecución, pero en un sistema multiprogramado, esta división se debe realizar de forma tal que se pueda albergar en memoria la cantidad óptima de procesos de manera que el procesador permanezca ocioso la menor cantidad de tiempo posible.

La gestión de la memoria también es responsable por el intercambio de información entre la memoria principal y la memoria secundaria, básicamente esto es la esencia de la gestión de la memoria.

Requisitos

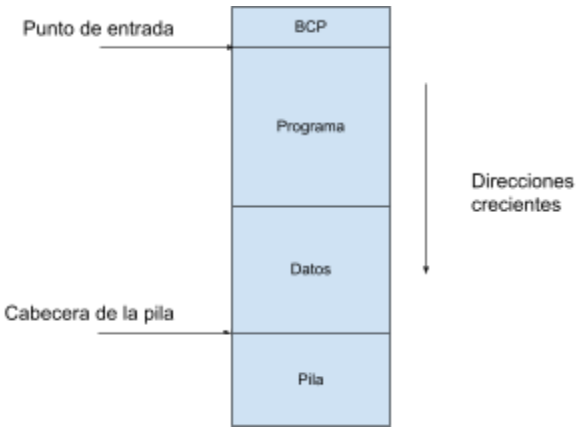
Todo sistema operativo debe satisfacer 5 requisitos de cara a la gestión de la memoria. A continuación los describiremos.

Reubicación

La reubicación es una función sumamente importante que cumple el sistema operativo y que es vital en un sistema que ofrece multiprogramación. La reubicación se encarga de ubicar/alojar un proceso en memoria. Este proceso puede ser o bien nuevo o bien un proceso que fue suspendido y debe ser traído a memoria principal nuevamente.

Pensemos que sin esta función el programador debería conocer anticipadamente en qué porción de memoria debería cargar su programa. Por otro lado hay que tener en cuenta que cuando un proceso es traído nuevamente a memoria, sería muy difícil tener que cargarlo en la misma posición de memoria en la que se encontraba antes de ser suspendido.

El proceso de reubicación, facilita en cierta forma el proceso de traducción de referencias a memoria (instrucciones y datos) de un proceso. Esto se debe a que el encargado de cargar un proceso en memoria principal es el sistema operativo y por tal motivo va a conocer la dirección de memoria donde se ubican el BCP, el punto de entrada del programa y la cabecera de la pila de ejecución del programa. A modo de ejemplo podemos pensar que la imagen de un proceso ocupa una región contínua de memoria como en el siguiente gráfico:



Protección

En un sistema multiprogramado es importante satisfacer el requisito de que ningún proceso pueda acceder al espacio de memoria de otro. Este requisito se complejiza aún más al implementar la reubicación. Por tal motivo, todas las referencias a memoria generadas por un proceso deben ser comprobadas de forma dinámica en tiempo de ejecución.

Los mecanismos de protección, son ejecutados por el procesador (a nivel hardware) ya que sería muy costoso para el sistema operativo calcular y comprobar todas las referencias a memoria de cada proceso y además porque el sistema operativo es otro proceso y no debería controlarse a sí mismo.

Compartición

Los mecanismos de protección deben contar con la flexibilidad suficiente para que si dos o más procesos están compartiendo determinada porción de memoria, puedan acceder a la misma. Por ejemplo, procesos que están cooperando para realizar una tarea podrían tener que acceder a la misma porción de memoria donde se encuentra una estructura de datos.

Organización lógica y Organización física

Tanto la memoria principal, como la secundaria, se organiza como una secuencia unidimensional de bytes o palabras.

Por otra parte, los programas (código + datos) generalmente se organizan en módulos.

El sistema operativo y el hardware del computador deben tratar de forma efectiva los programas de usuario para asignarlos de la mejor manera posible en la memoria. Veremos distintos esquemas, como puede ser la *Segmentación* o la *Paginación*.

Particionamiento

Estático (fijo)

Este es el esquema más simple. Divide la memoria principal en partes fijas, una para el sistema operativo y otra para los programas de usuario. La porción de memoria principal asignada a los programas de usuario, también está dividida en partes fijas.

Particiones de igual tamaño

En este caso, cualquier proceso cuyo tamaño es menor o igual que el tamaño de la partición, puede cargarse en cualquier partición disponible. Si en algún momento todas las particiones se encuentran ocupadas y no hay ningún proceso en estado *Listo* o *Ejecutando* el sistema operativo puede mandar a disco (swap) a un proceso.

La ubicación de un proceso en memoria es un problema trivial debido a que todas las particiones son del mismo tamaño, por lo tanto un proceso que es traído a memoria se ubicará en la primer partición que se encuentre libre.

Esta técnica presenta dos problemas:

- Un programa podría ser más grande que el tamaño de una partición. En este caso el programador deberá utilizar distintas técnicas para cargar en memoria principal el programa.
- Se realiza un uso ineficiente de la memoria principal, ya que sin importar el tamaño de un proceso, siempre ocupa una partición. Esto se conoce como ***fragmentación interna***.

Particiones de tamaño variable

Con particiones de tamaño variable existen 2 formas de ubicar procesos en la memoria principal. La primera y más sencilla consiste en asignar cada proceso en la partición más chica que lo pueda contener. En este caso el sistema operativo mantendrá una cola de procesos por cada partición con el objetivo de que los procesos sean asignados siempre en la misma partición.

La ventaja de este esquema es que reduce la fragmentación interna, pero la desventaja es que podrían quedar particiones de memoria sin utilizar, lo que llevaría el sistema a un uso ineficiente del procesador y una ejecución de procesos poco eficiente.

La segunda alternativa sería tener una única cola de procesos y al momento de cargar un proceso en memoria, se selecciona la partición más pequeña que pueda contener al proceso en cuestión. Nuevamente, si todas las particiones se encuentran ocupadas se podrá seleccionar un proceso para ser enviado a disco (swap). A los algoritmos de planificación que ya mencionamos

en la unidad anterior, podemos agregar que los procesos que ocupen la partición más pequeña que permita almacenar el proceso entrante son los candidatos a ser suspendidos.

Tanto sea un esquema de particiones fijas como uno de particiones variables, las ventajas son su simpleza a la hora de la implementación (respecto de otros esquemas que veremos a continuación) y que presentan una sobrecarga mínima al procesador.

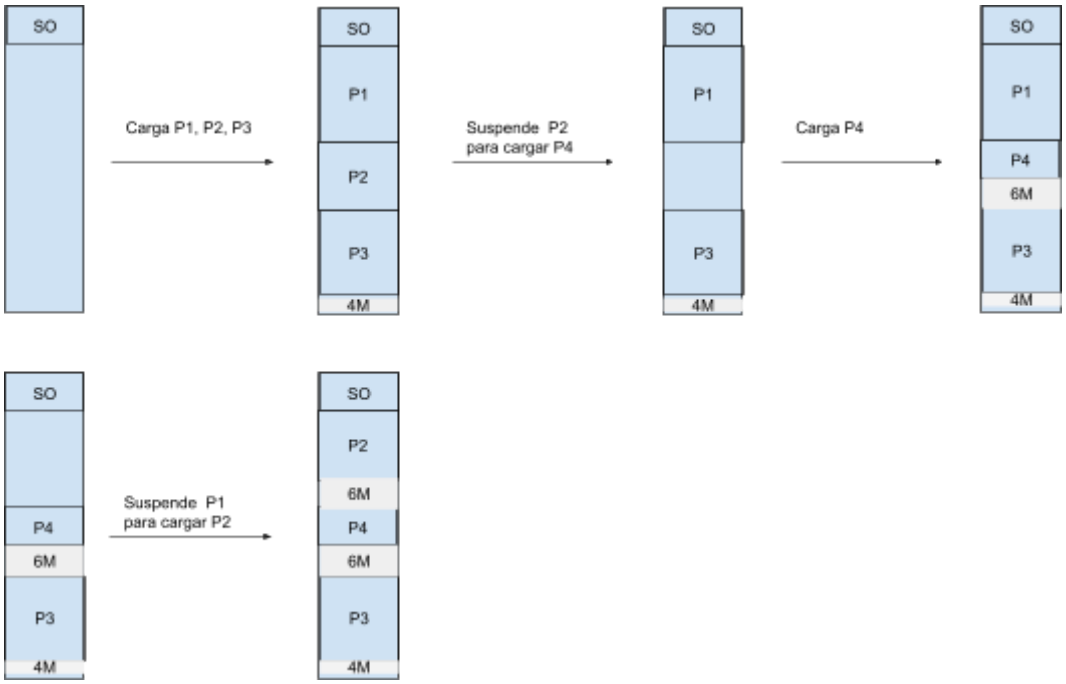
Las desventajas son:

- Se limita la cantidad de procesos activos (no suspendidos)
- El uso de la memoria es ineficiente ya que los tamaños de las particiones se definen en tiempo de compilación del sistema operativo.

Dinámico (variable)

En este esquema las particiones son de longitud y número variable. Cuando un proceso se carga en memoria, se le asigna exactamente la cantidad de memoria necesitada.

Este esquema, si bien parece ser óptimo, presenta la desventaja que genera **fragmentación externa**, es decir, después de cada asignación quedan huecos de memoria que no son asignados a ningún proceso. Veamos esto con un ejemplo: supongamos que tenemos una memoria principal de 64MB. Nuestro sistema operativo ocupa 8M y se necesita ejecutar 4 procesos P1 (20M), P2 (14M), P3 (18M) y P4 (8M).



En el ejemplo vemos que si bien la situación inicial es correcta, el resultado final deja muchos “huecos” en la memoria. A esto se le denomina fragmentación externa.

La diferencia con la fragmentación interna es que la fragmentación interna se produce dentro de una partición, mientras que la externa es sobre la memoria principal.

Para lidiar con este nuevo problema surge la **compactación**. Este proceso lo ejecuta el sistema operativo de forma periódica, reubica los procesos de manera que queden en posiciones de memoria contigua y junta los fragmentos de memoria libre en un solo fragmento.

Algoritmo de ubicación

Debido a que el proceso de *compactación* consume un tiempo considerable, cuando se diseña un sistema operativo se debe pensar bien el algoritmo de ubicación de procesos en memoria. Existen 3 alternativas:

- **Best-fit** o mejor ajuste: selecciona el bloque de memoria que mejor se ajuste al tamaño del proceso a cargar
- **First-fit** o primer ajuste: analiza la memoria desde el principio y escoge el primer bloque libre que sea suficientemente grande para contener al proceso a cargar.
- **Next-fit** o siguiente ajuste: similar a First-fit sólo que analiza la memoria desde la última posición en donde se asignó un proceso.

A modo de resumen podemos decir que ambos esquemas presentan desventajas, por un lado el esquema de particionamiento fijo hace un aprovechamiento ineficiente de la memoria y por otro, el esquema de particionamiento dinámico es más complejo de mantener además de imponer la sobrecarga de la compactación.

Sistema Buddy

Este sistema surge como una alternativa que resuelve las desventajas de los esquemas presentados. Para gestionar la memoria disponible, este sistema divide los bloques de memoria por la mitad hasta obtener un bloque que se ajuste lo mejor posible a la petición de memoria analizada.

El sistema *Buddy* propone que los bloques de memoria disponible son de tamaño 2^k , siendo $L \leq k \leq U$, donde:

- 2^L = bloque de tamaño más pequeño
- 2^U = bloque de tamaño máximo.

El algoritmo empleado para asignar bloques de memoria a un proceso realiza los siguientes pasos:

1. Antes de comenzar se asume que se dispone de un único bloque de tamaño 2^U.
2. Si el proceso a cargar tiene un tamaño *s*, tal que $2^{U-1} < s < 2^U$ se asigna el bloque de tamaño 2^U entero.
3. Si la relación del paso anterior no se cumple, entonces el bloque se divide en 2 bloques de tamaño 2^{U-1}.
4. Si el proceso a cargar tiene un tamaño *s*, tal que $2^{U-2} < s < 2^{U-1}$ se asigna el bloque de tamaño 2^{U-1} entero.
5. En caso de que no se cumpla el paso 4, volver al paso 3.
6. El proceso se repite hasta encontrar el bloque de menor tamaño que pueda almacenar el proceso.

Veamos un ejemplo:

Supongamos que la memoria principal es de 1M y un proceso solicita 100K, el sistema *Buddy* realizará lo siguiente:

1. Se verifica que la relación $512 \leq 100 \leq 1024$ no se cumple, por lo que el bloque de 1M (1024K) se divide en 2 de 512K cada uno.
2. Nuevamente se verifica que $256 \leq 100 \leq 512$no se cumple, por lo que se divide el primer bloque de 512 en 2 de 256.
3. El proceso se repite hasta que se llega a $64 \leq 100 \leq 128$ donde la relación si se cumple, por lo tanto, al proceso se le asigna un bloque de 128K.

<table>
<tr><td>#</td><td colspan="5">Memoria</td></tr>
<tr><td>1</td><td colspan="5">1M</td></tr>
<tr><td>2</td><td colspan="4">512K</td><td>512K</td></tr>
<tr><td>3</td><td colspan="3">256K</td><td>256K</td><td>512K</td></tr>
<tr><td>4</td><td colspan="2">128K</td><td>128K</td><td>256K</td><td>512K</td></tr>
<tr><td>5</td><td>64K</td><td>64K</td><td>128K</td><td>256K</td><td>512K</td></tr>
</table>

¿Cómo es el proceso para recuperar memoria asignada?

Cada vez que un proceso libera memoria que ya no utiliza o el mismo finaliza liberando por completo el bloque de memoria asignada, en la memoria se generan “huecos”, es decir bloques de memoria que no está asignada a ningún proceso.

Cuando otro proceso realice una nueva petición de memoria, el sistema Buddy realizará una búsqueda entre los bloques de memoria libre e intentará realizar la asignación. Si dos bloques libres se encuentran en posiciones contiguas de memoria y la solicitud de memoria realizada, es mayor que uno de los bloques, el sistema unifica dichos bloques en uno solo para poder satisfacer la petición.

Técnicas avanzadas de gestión de memoria

Todos los esquemas vistos hasta ahora son ineficientes desde el punto de vista de la gestión de la memoria y por lo tanto en los sistemas operativos contemporáneos fueron reemplazados por esquemas más sofisticados y eficientes.

Antes de adentrarnos en el análisis de estos nuevos esquemas conviene hacer mención a los distintos “tipos” de direcciones de memoria que maneja el sistema operativo en conjunto con el procesador:

- **Dirección lógica o relativa**: es una referencia a una dirección de memoria y es independiente de la ubicación actual. Debe ser traducida a una dirección física antes de poder ser utilizada. Se dice que es relativa, ya que se define en relación a un punto conocido. Este punto conocido es otra dirección de memoria, normalmente conocida por el procesador, que es el registro base
- **Dirección física o absoluta**: es una dirección real en la memoria principal.

En esquemas de particiones fijas, donde el sistema operativo maneja una cola para cada partición, se espera que cada proceso se cargue siempre en la misma partición a la que fue asignado inicialmente, por lo tanto luego de que el proceso se carga por primera vez, todas las referencias a direcciones de memoria son reemplazadas por direcciones absolutas. Esto se realiza tomando como referencia la dirección de memoria contenida en el registro base del procesador.

En esquemas de particiones fijas, con una única cola para los procesos o esquemas de particiones dinámicos (con reubicación y/o compactación) las direcciones de memoria deben ser traducidas cada vez que el proceso es llevado a memoria.

El proceso de traducción de direcciones de memoria contempla 2 pasos:

1. Primero el procesador traduce la dirección relativa sumando la dirección base del proceso.
2. Verifica que la dirección traducida se encuentre dentro de los límites del proceso.
    a. Si la dirección traducida no se encuentra dentro de los límites del proceso se genera una interrupción, conocida como *segmentation fault* (falla de segmento)

Paginación

Este esquema divide la memoria en particiones de tamaño fijo (relativamente pequeño) denominados **marcos** y cada proceso también es dividido en porciones fijas del mismo tamaño denominadas **páginas**.

Este esquema elimina la fragmentación externa y la fragmentación interna la reduce de tal forma que sólo afecta la última página de cada proceso.

Veamos un ejemplo:

Memoria principal

0	
1	
2	
3	
4	

Carga proc. A y B con 5 marcos libres

A0
A1
B0
B1

Supongamos que A se suspende y para cargar el proceso C necesitamos 3 marcos

0	C0
1	C1
2	B0
3	B1
4	C2

Como vemos las páginas del proceso C pueden cargarse sin necesidad de estar en forma consecutiva.

¿Ahora bien, cómo hace el sistema operativo junto al procesador para resolver una referencia a memoria en un esquema como el graficado?

Un sistema operativo que trabaje con paginación mantiene tablas de páginas por cada proceso. Cada tabla de páginas contiene la ubicación del marco por cada página del proceso, es decir, la dirección de memoria de dicho marco.

Ejemplo:

Proc. A

0	-
1	-

Proc. B

0	2
1	3

Proc. C

0	0
1	1
2	4

Cada tabla de página, contiene el número de marco donde está cargada. Si el proceso se suspende, la tabla es “reiniciada”. Adicionalmente, el sistema operativo mantiene una tabla con los marcos que se encuentran libres.

En este esquema, las direcciones lógicas están expresadas en base al registro base de cada programa y se definen como un número de página más un desplazamiento dentro de la página. Por lo tanto, para traducir una dirección lógica a una física, el procesador deberá:

1. extraer la dirección de la página
2. encontrar cuál es el marco donde se encuentra dicha página.
3. sumar a la dirección del marco el desplazamiento para obtener la dirección física.

Segmentación

Podemos considerar que un programa de usuario se puede dividir en al menos 2 segmentos, uno para el código y otro para datos. Por lo general, estos segmentos son 3, pero 1 de esos segmentos se subdivide en 2:

- **Código**: o segmento de texto, contiene el código ejecutable del programa
- **Datos**:
    - Locales o globales: segmento donde se establecen las variables locales del proceso, así como también las globales y estáticas.
    - heap: segmento donde se establecen las variables cuya reserva de memoria es dinámica. Este segmento puede crecer o disminuir su tamaño.
- **Stack**: segmento utilizado para “apilar” las llamadas a funciones y sus parámetros.

La segmentación ve al espacio de direcciones de memoria de un proceso como un conjunto de segmentos (código, datos, stack). Cada uno de estos segmentos es una unidad lógica que permite al programador organizar el código y los datos de sus programas y a su vez, permite al sistema operativo ubicarlos en porciones de memoria distintas que no necesariamente deben ser contiguas.

La segmentación, al asignar la cantidad de memoria requerida por cada segmento, elimina la fragmentación interna y reduce la fragmentación externa debido a que los tamaños de segmentos son relativamente pequeños.

La segmentación posibilita que varios procesos compartan el segmento de código y el de datos.

Al igual que con la paginación, el sistema operativo crea tablas de segmentos por cada proceso y una lista de bloques de memoria libres.

Para traducir una dirección de memoria de un segmento (lógica) a una dirección de memoria física, se debe recuperar el número de segmento y luego acceder a la tabla de segmentos para recuperar la dirección física donde se encuentra dicho segmento. Finalmente hay que sumar el desplazamiento y así formar la dirección física.

Una vez que se formó la dirección física el procesador verifica que dicha dirección no supere la longitud del segmento. Si así fuera se produce una interrupción conocida como *segmentation fault* o falla de segmento.

Lectura recomendada:

- Segmentation fault

Bibliografía utilizada

● William Stallings. Sistemas operativos. Pearson Education. S.A., Madrid, 2005. ISBN-84-205-4462-0.

● AndrewS.Tanenbaum. Modern Operating System. Pearson Education Inc., 2009. ISBN-Q-lB-filBMST-L

● Multilevel Feedback queue. Wikipedia, La enciclopedia libre, 2019 [consulta: 21 de marzo del 2019]. Disponible en https://en.wikipedia.org/wiki/Multilevel_feedback_queue