Memoria Virtual

Carrera: Licenciatura en Sistemas de Información

Materia: Sistemas Operativos

Año: 2019

Profesores: Aguiar Osvaldo

Rapallini Ulises

Alumnos: Curtoni Juan

Gonzalez Exequiel

Soria Lucas

1. Las dos características de la paginación y la segmentación que son la clave de este comienzo son:
   - Todas las referencias a la memoria dentro un proceso se realizan a direcciones lógicas, que se traducen dinámicamente en direcciones físicas durante la ejecución. Esto significa que un proceso puede ser llevado y traído a memoria de forma que ocupe diferentes regiones de la memoria principal en distintos instantes de tiempo durante su ejecución.
   - Un proceso puede dividirse en varias porciones (páginas o segmentos) y estas porciones no tienen que estar localizadas en la memoria de forma contigua durante la ejecución. La combinación de la traducción de direcciones dinámicas en ejecución y el uso de una tabla de páginas o segmentos lo permite.

2. Supongamos que se tiene que traer un nuevo proceso de memoria. El sistema operativo comienza trayendo únicamente una o dos porciones, que incluye la porción inicial del programa y la porción inicial de datos sobre la cual acceden las primeras instrucciones acceden. Esta parte del proceso que se encuentra realmente en la memoria principal para, cualquier instante de tiempo, se denomina conjunto **residente del proceso**. Sus ventajas son:
   - **Pueden mantenerse un mayor número de procesos en memoria principal**. Debido a que sólo vamos a cargar algunas de las porciones de los procesos a ejecutar, existe espacio para más procesos. Esto nos lleva a una utilización más eficiente del procesador porque es más probable que haya al menos uno o más de los numerosos procesos que se encuentre en el estado Listo, en un instante de tiempo concreto.
   - **Un proceso puede ser mayor que toda la memoria principal**: Con la memoria virtual basada en paginación o segmentación, este trabajo se delega al sistema operativo y al hardware. En lo que concierne al programador, él está trabajando con una memoria enorme, con un tamaño asociado al almacenamiento en disco. El sistema operativo automáticamente carga porciones de un proceso en la memoria principal cuando éstas se necesitan.

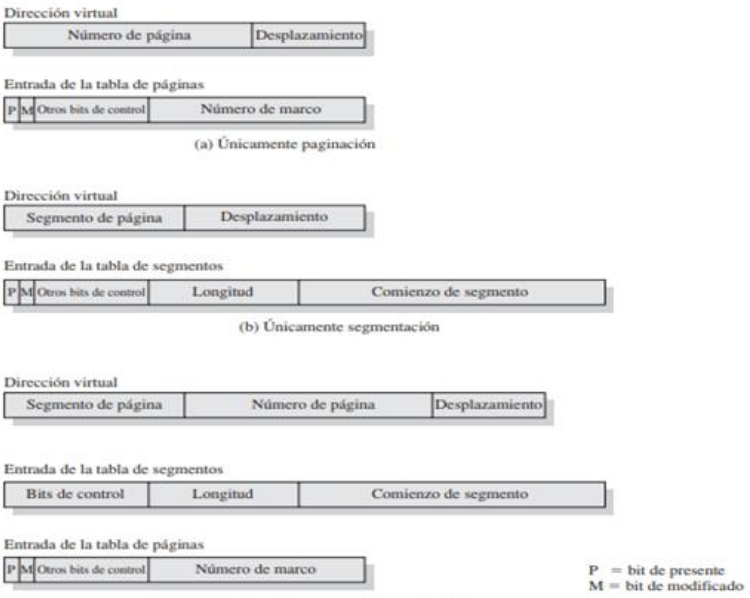
3. Un proceso de gran tamaño, consiste en un programa largo más un gran número de vectores de datos. Sería un desperdicio cargar docenas de porciones de dicho proceso cuando sólo unas pocas porciones se usarán antes de que el programa se suspenda o se mande a **zona de intercambio o swap**. Se puede hacer un mejor uso de la memoria cargando únicamente unas pocas porciones. Entonces, si el programa salta a una destrucción o hace referencia a un dato que se encuentra en una porción de memoria que no está en la memoria principal, se dispara un fallo. Éste indica al sistema operativo que debe conseguir la porción deseada. Así sólo unas pocas porciones de cada proceso se encuentran en memoria, y por tanto se pueden mantener más procesos alojados en la misma. Además, se ahorra tiempo porque las porciones del proceso no usadas no se expulsarán de la **memoria a swap y de swap a la memoria**.

   En estado estable, prácticamente toda la memoria principal se encontrará ocupada con porciones de procesos. Cuando el sistema operativo traiga una porción a la

memoria, debe expulsar otra. Si elimina una porción justo antes de que vaya a ser utilizada, deberá recuperar dicha porción de nuevo casi de forma inmediata. Un abuso de esto lleva al **thrashing**: el sistema consume la mayor parte del tiempo enviando y trayendo porciones de swap en lugar de ejecutar instrucciones.

4. El **principio de proximidad** indica que las referencias al programa y a los datos dentro de un proceso tienden a agruparse. Por tanto, se resume que sólo unas pocas porciones del proceso se necesitarán a lo largo de un periodo de tiempo corto. También, es posible hacer suposiciones inteligentes sobre cuáles son las porciones del proceso que se necesitarán en un futuro próximo, para evitar este trasiego.s

5. **Estructura de la tabla de páginas**. El mecanismo básico de lectura de una palabra de la memoria implica la traducción de la dirección virtual, o lógica, consistente en un número de página y un desplazamiento, a la dirección física, consistente en un número de marco y un desplazamiento, usando para ello la tabla de páginas. Debido a que la tabla de páginas es de longitud variable dependiendo del tamaño del proceso, no podemos suponer que se encuentra almacenada en los registros. En lugar de eso, debe encontrarse en la memoria principal para poder ser accedida.



6. La mayoría de esquemas de memoria virtual almacena las **tablas de páginas** también en la memoria virtual, en lugar de en la memoria real. Esto representa que las tablas de páginas están sujetas a paginación igual que cualquier otra página. Cuando un proceso está en ejecución, al menos parte de su tabla de páginas debe encontrarse en memoria, incluyendo la entrada de tabla de páginas de la página actualmente en ejecución.

7. **Tabla de páginas multinivel:** Una manera de afrontar el problema de gasto de memoria de las tablas de páginas es utilizar tablas de página multinivel. Con este esquema, en vez de tener una unida tabla de páginas por proceso, hay una jerarquía de tablas. Existe una única tabla de páginas de primer nivel. Cada entrada de esta tabla

apunta a tablas de páginas de segundo nivel. A su vez, a las tablas de página de segundo nivel apuntan a tablas de página de tercer nivel. Así, sucesivamente, por cada nivel de la jerarquía. Las tablas de páginas de ultimo nivel apuntan directamente a marcos de página.

La ventaja de este modelo es que si todas las entradas de una tabla de páginas de cualquier nivel están marcadas como invalidas, no es necesario almacenar esta tabla de páginas. Bastaría con marcar como invalida la entrada de la tabla de páginas de nivel superior que corresponden con esta tabla vacía.

8. Una **tabla de páginas invertida** contiene una entrada por cada marco de página. Cada entrada identifica que página está almacenada en ese marco y cuáles son sus características. Para ello contiene el número de página y un identificador de proceso al que pertenece la página. Dado que la tabla está organizada por marcos, no se puede hacer una búsqueda directa. Se debería acceder a todas las entradas buscando la página solicitada. Para agilizar esta búsqueda, generalmente la tabla de páginas se organiza como una tabla hash.

9. **TLB:** Es una cache especial de alta velocidad para entradas de la tabla de páginas. Esta cache funciona de forma similar a una memoria cache general., y contiene aquellas entradas de las tablas de páginas que han sido usada de forma más reciente.

   ¿Cómo funciona?: Dada una dirección virtual, el procesador primero examina la TLB, si la entrada de la tabla de páginas solicitada está presente (acierto en TLB), entonces se recupera el número de marco y se construye la dirección real. Si la entrada de la tabla de páginas solicitada no se encuentra (fallo en la TLB), el procesador utiliza el número de página para indexar la tabla de páginas del proceso y examinar la correspondiente entrada de la tabla de páginas. Si el bit de presente está puesto a 1, entonces la página se encuentra en memoria principal, y el procesador puede recuperar el número de marco desde la entrada de la tabla de páginas para construir la dirección real. El procesador también autorizará la TLB para incluir esta nueva entrada de tabla de páginas. Finalmente, si el bit presente no está puesto a 1, entonces la página solicitada no se encuentra en la memoria principal y se produce un fallo de acceso memoria, llamado fallo de página.

10. **Tamaño de página:** Hay varios factores a considerar: Por un lado, está la fragmentación interna. Evidentemente, cuanto mayor es el tamaño de la página, menor cantidad de fragmentación interna. Para optimizar el uso de la memoria principal, sería beneficioso reducir la fragmentación interna. Por otro lado, cuanto menor es la página, mayor número de páginas son necesarias para cada proceso. Un mayor número de páginas por proceso significa también mayores tablas de páginas.

11. **Implicancias del esquema de segmentación de memoria:**
    A. Simplifica el tratamiento de estructuras de datos que pueden crecer. Si el programador no conoce a priori el tamaño que una estructura de datos en particular puede alcanzar es necesario hacer una estimación salvo que se utilicen tamaños de

segmento dinámicos. Con la memoria virtual segmentada, a una estructura de datos se le puede asignar su propio segmento, y el sistema operativo expandirá o reducirá el segmento bajo demanda. Si un segmento que necesita expandirse se encuentre en la memoria principal y no hay suficiente tamaño, el sistema operativo poder mover el segmento a un área de la memoria principal mayor, si se encuentra disponible, o enviarlo a swap. En este último caso el segmento al que se ha incrementado el tamaño volverá a la memoria principal en la siguiente oportunidad que tenga.

B. Permite programas que se modifican o recopilan de forma independiente, sin requerir que el conjunto completo de programas se re-enlacen y se vuelvan a cargar. De nuevo, esta posibilidad se puede articular por medio de la utilización de múltiples segmentos.

C. Da soporte a la compartición entre procesos. El programador puede situar un programa de utilidad o una tabla de datos que resulte útil en un segmento al que pueda hacerse referencia desde otros procesos.

D. Soporta los mecanismos de protección. Esto es debido a que un segmento puede definirse para contener un conjunto de programas o datos bien descritos, el programador o el administrador de sistemas puede asignar privilegios de acceso de una forma apropiada.

12. En la exposición de la segmentación sencilla, indicamos que cada proceso tiene su propia tabla de segmentos, y que cuando todos estos segmentos se han cargado en la memoria principal, la tabla de segmentos del proceso se crea y se carga también en la memoria principal. Cada entrada de la tabla de segmentos contiene la dirección de comienzo del correspondiente segmento en la memoria principal, así como la longitud del mismo. El mismo mecanismo, una tabla segmentos, se necesita cuando se están tratando esquemas de memoria virtual basados en segmentación. De nuevo, lo habitual es que haya una única tabla de segmentos por cada uno de los procesos. En este caso sin embargo, las entradas en la tabla de segmentos son un poco más complejas. Debido a que sólo algunos de los segmentos del proceso pueden encontrarse en la memoria principal, se necesita un bit en cada entrada de la tabla de segmentos para indicar si el correspondiente segmento se encuentra presente en la memoria principal o no. Si indica que el segmento está en memoria, la entrada también debe incluir la dirección de comienzo y la longitud del mismo.

13. **Segmentación paginada:** En un sistema combinado de paginación/segmentación, el espacio de direcciones del usuario se divide en un número de segmentos, a discreción del programador. Cada segmento es, por su parte, dividido en un número de páginas de tamaño fijo, que son del tamaño de los marcos de la memoria principal. Si un segmento tiene longitud inferior a una página, el segmento ocupará únicamente una página. Desde el punto de vista del programador, una dirección lógica sigue conteniendo un número de segmento y un desplazamiento dentro de dicho segmento. Desde el punto de vista del sistema, el desplazamiento dentro del segmento es visto

como un número de página y un desplazamiento dentro de la página incluida en el segmento.

14. **Con paginación bajo demanda**, una página se trae a memoria sólo cuando se hace referencia a una posición en dicha página. Si el resto de elementos en la política de gestión de la memoria funcionan correctamente, ocurriría lo siguiente. Cuando un proceso se arranca inicialmente, va a haber una ráfaga de fallos de página. Según se van trayendo más y más páginas a la memoria, el principio de proximidad sugiere que las futuras referencias se encontrarán en las páginas recientemente traídas. Así, después de un tiempo, la situación se estabilizará y el número de fallos de página caerá hasta un nivel muy bajo.

15. Para sistemas que usan o bien paginación pura o paginación combinada con segmentación, la ubicación es habitualmente irrelevante debido a que el hardware de traducción de direcciones y el hardware de acceso a la memoria principal pueden realizar sus funciones en cualquier combinación de página-marco con la misma eficiencia.

16. **Bloqueo de marcos:** Cuando un marco está bloqueado, la página actualmente almacenada en dicho marco no puede reemplazarse. Gran parte del núcleo del sistema operativo se almacena en marcos que están bloqueados, así como otras estructuras de control claves. Adicionalmente, los buffers de E/S y otras áreas de tipo critico también se ponen en marcos bloqueados en la memoria principal. El bloqueo se puede realizar asociando un bit de bloqueo a cada uno de los marcos. Este bit se puede almacenar en la tabla de marcos o también incluirse en la tabla de páginas actual.

17. **<u>Algoritmos básicos de reemplazo:</u>**
    - **La política óptima** de selección tomará como reemplazo la página para la cual el instante de la siguiente referencia se encuentra más lejos. Esta política es imposible de implementar, porque requiere que el sistema operativo tenga un perfecto conocimiento de los eventos futuros. Sin embargo, se utiliza como un estándar a partir del cual contrastar algoritmos reales.
    - **La política de reemplazo de la página usada menos recientemente (LRU)** seleccionará como candidata la página de memoria que no se haya referenciado desde hace más tiempo. Debido al principio de proximidad referenciada, esta página sería la que tiene menos probabilidad de volverá a tener referencias en un futuro próximo. El problema con esta alternativa es la dificultad en su implementación.
    - **La política FIFO** trata los marcos de página ocupados como si se tratase de un buffer circular, y las páginas se remplazan mediante una estrategia cíclica de tipo round-robin. Todo lo que se necesita es un puntero que recorra de forma circular los marcos de página del proceso. Por tanto, se trata de una de las políticas de reemplazo más sencilla de implementar.

      El razonamiento tras este modelo, además de su simplicidad, es el reemplazo de la página que lleva en memoria más tiempo: una página traída a la memoria hace

mucho tiempo puede haber dejado de utilizarse. Este razonamiento a menudo es erróneo, debido a que es habitual que en los programas haya una zona del mismo o regiones de datos que son utilizados de forma intensiva durante todo el tiempo de vida del proceso. Esas páginas son expulsadas de la memoria y traídas de nuevo de forma repetida por un algoritmo de tipo FIFO.

- **La política del reloj**, da vueltas a través de todas las páginas del buffer buscando una que no se haya modificado desde que se ha traído y que no haya sido accedida recientemente. Esta página es una buena opción para reemplazo y tiene la ventaja que, debido a que no se ha modificado, no necesita escribirse de nuevo en la memoria secundaria. Si no se encuentra una página candidata en la primera vuelta, el algoritmo da una segunda vuelta al buffer, buscando una página modificada que no se haya accedido recientemente