Gestion de Memoria

Los requisitos que la gestión de la memoria debe satisfacer son :

- Reubicación.
- Protección.
- Compartición.
- Organización lógica.
- Organización física.

REUBICACIÓN

En un sistema multiprogramado, la memoria principal disponible se comparte generalmente entre varios procesos. Normalmente, no es posible que el programador sepa anticipadamente qué programas residirán en memoria principal en tiempo de ejecución de su programa. Adicionalmente, sería bueno poder intercambiar procesos en la memoria principal para maximizar la utilización del procesador, proporcionando un gran número de procesos para la ejecución. Una vez que un programa se ha lleva- do al disco, sería bastante limitante tener que colocarlo en la misma región de memoria principal don- de se hallaba anteriormente, cuando éste se trae de nuevo a la memoria. Por el contrario, podría ser necesario **reubicar** el proceso a un área de memoria diferente.

Por tanto, no se puede conocer de forma anticipada dónde se va a colocar un programa y se debe permitir que los programas se puedan mover en la memoria principal, debido al intercambio o *swap*.

Cada proceso debe protegerse contra interferencias no deseadas por parte de otros procesos, sean accidentales o intencionadas. Por tanto, los programas de otros procesos no deben ser capaces de referenciar sin permiso posiciones de memoria de un proceso, tanto en modo lectura como escritura. Por un lado, lograr los requisitos de la reubicación incrementa la dificultad de satisfacer los requisitos de protección. Por tanto, todas las referencias de memoria generadas por un pro- ceso deben comprobarse en tiempo de ejecución para poder asegurar que se refieren sólo al espacio de memoria asignado a dicho proceso.

Obsérvese que los requisitos de protección de memoria deben ser satisfechos por el procesador (hardware) en lugar del sistema operativo (software). Esto es debido a que el sistema operativo no puede anticipar todas las referencias de memoria que un programa hará. Incluso si tal anticipación fuera posible, llevaría demasiado tiempo calcularlo para cada programa a fin de comprobar la viola-ción de referencias de la memoria. Por tanto, sólo es posible evaluar la permisibilidad de una referencia (acceso a datos o salto) en tiempo de ejecución de la instrucción que realiza dicha referencia. Para llevar a cabo esto, el hardware del procesador debe tener esta capacidad.

COMPARTICIÓN

Cualquier mecanismo de protección debe tener la flexibilidad de permitir a varios procesos acceder a la misma porción de memoria principal. Por tanto, el sistema de gestión de la memoria debe permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección esencial.

ORGANIZACIÓN LÓGICA

La mayoría de los programas se organizan en módulos, algunos de los cuales no se pueden modificar (sólo lectura, sólo ejecución) y algunos de los cuales contienen datos que se pueden modificar. Si el sistema operativo y el hardware del computador pueden tratar de forma efectiva los programas de usuarios y los datos en la forma de módulos de algún tipo, entonces se pueden lograr varias ventajas:

1. Los módulos se pueden escribir y compilar independientemente, con todas las referencias de un módulo desde otro resueltas por el sistema en tiempo de ejecución.
2. Con una sobrecarga adicional modesta, se puede proporcionar diferentes grados de protección a los módulos (sólo lectura, sólo ejecución).

3. Es posible introducir mecanismos por los cuales los módulos se pueden compartir entre los procesos. La ventaja de proporcionar compartición a nivel de módulo es que se corresponde con la forma en la que el usuario ve el problema, y por tanto es fácil para éste especificar la compartición deseada.

ORGANIZACIÓN FÍSICA

La memoria del computador se organiza en al menos dos niveles, conocidos como memoria principal y memoria secundaria. La memoria principal proporciona acceso rápido a un coste relativamente alto. Adicionalmente, la memoria principal es volátil; es decir, no proporciona almacenamiento permanente. La memoria secundaria es más lenta y más barata que la memoria principal y normalmente no es volátil. Por tanto, la memoria secundaria de larga capacidad puede proporcionar almacenamiento para programas y datos a largo plazo, mientras que una memoria principal más pequeña contiene programas y datos actualmente en uso.

La responsabilidad para este flujo podría asignarse a cada programador en particular, pero no es practicable o deseable por dos motivos:

1. La memoria principal disponible para un programa más sus datos podría ser insuficiente.
2 .En un entorno multiprogramado, el programador no conoce en tiempo de codificación cuánto espacio estará disponible o dónde se localizará dicho espacio.

Por tanto, está claro que la tarea de mover la información entre los dos niveles de la memoria debería ser una responsabilidad del sistema. Esta tarea es la esencia de la gestión de la memoria.

Técnica	Descripción	Virtudes	Defectos
Particionamiento fijo	La memoria principal se divide en particiones estáticas en tiempo de generación del sistema. Un proceso se puede cargar en una partición con igual o superior tamaño.	Sencilla de implementar; poca sobrecarga para el sistema operativo.	Uso ineficiente de la memoria, debido a la fragmentación interna; debe fijarse el número máximo de procesos activos.
Particionamiento dinámico	Las particiones se crean de forma dinámica, de tal forma que cada proceso se carga en una partición del mismo tamaño que el proceso.	No existe fragmentación interna; uso más eficiente de memoria principal.	Uso ineficiente del procesador, debido a la necesidad de compactación para evitar la fragmentación externa.
Paginación sencilla	La memoria principal se divide en marcos del mismo tamaño. Cada proceso se divide en páginas del mismo tamaño que los marcos. Un proceso se carga a través de la carga de todas sus páginas en marcos disponibles, no necesariamente contiguos.	No existe fragmentación externa.	Una pequeña cantidad de fragmentación interna.

	contiguos.		
Segmentación sencilla	Cada proceso se divide en segmentos. Un proceso se carga cargando todos sus segmentos en particiones dinámicas, no necesariamente contiguas.	No existe fragmentación interna; mejora la utilización de la memoria y reduce la sobrecargada respecto al particionamiento dinámico.	Fragmentación externa.
Paginación con memoria virtual	Exactamente igual que la paginación sencilla, excepto que no es necesario cargar todas las páginas de un proceso. Las páginas no residentes se traen bajo demanda de forma automática.	No existe fragmentación externa; mayor grado de multiprogramación; gran espacio de direcciones virtuales.	Sobrecarga por la gestión compleja de la memoria.
Segmentación con memoria virtual	Exactamente igual que la segmentación, excepto que no es necesario cargar todos los segmentos de un proceso. Los segmentos no residentes se traen bajo demanda de forma automática.	No existe fragmentación interna; mayor grado de multiprogramación; gran espacio de direcciones virtuales; soporte a protección y compartición.	Sobrecarga por la gestión compleja de la memoria.

PARTICIONAMIENTO DE LA MEMORIA

La operación principal de la gestión de la memoria es traer los procesos a la memoria principal para que el procesador los pueda ejecutar. En casi todos los sistemas multiprogramados modernos, esto implica el uso de un esquema sofisticado denominado memoria virtual. Por su parte, la memoria virtual se basa en una o ambas de las siguientes técnicas básicas: segmentación y paginación.

PARTICIONAMIENTO FIJO

En la mayoría de los esquemas para gestión de la memoria, se puede asumir que el sistema operativo ocupa alguna porción fija de la memoria principal y que el resto de la memoria principal está disponible para múltiples procesos. El esquema más simple para gestionar la memoria disponible es repartir- la en regiones con límites fijos.

Tamaños de partición

Una posibilidad consiste en hacer uso de particiones del mismo tamaño. En este caso, cualquier proceso cuyo tamaño es menor o igual que el tamaño de partición puede cargarse en cualquier partición disponible. Si todas las particiones están llenas y no hay ningún proceso en estado Listo o Ejecutando, Existen dos dificultades con el uso de las particiones fijas del mismo tamaño:

•Un programa podría ser demasiado grande para caber en una partición. En este caso, el programador debe diseñar el programa con el uso de overlays, de forma que sólo se necesite una porción del programa en memoria principal en un momento determinado.

•La utilización de la memoria principal es extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupa una partición entera.

Ambos problemas se pueden mejorar, aunque no resolver, utilizando particiones de tamaño diferente.

Algoritmo de ubicación

Con particiones del mismo tamaño, la ubicación de los procesos en memoria es trivial. En cuanto haya una partición disponible, un proceso se carga en dicha partición. Debido a que todas las particio- nes son del mismo tamaño, no importa qué partición se utiliza. Si todas las particiones se encuentran ocupadas por procesos que no están listos para ejecutar, entonces uno de dichos procesos debe llevar- se a disco para dejar espacio para un nuevo proceso. Cuál de los procesos se lleva a disco es una deci- sión de planificación.

Con particiones de diferente tamaño, hay dos formas posibles de asignar los procesos a las parti- ciones. La forma más sencilla consiste en asignar cada proceso a la partición más pequeña dentro de la cual cabe1. En este caso, se necesita una cola de planificación para cada

partición, que mantenga procesos en disco destinados a dicha partición (Figura 7.3a). La ventaja de esta técnica es que los procesos siempre se asignan de tal forma que se minimiza la memoria malgastada dentro de una partición (fragmentación interna).
No es óptima desde el punto de vista del sistema completo. En la Figura 7.2b, por ejemplo, se considera un caso en el que no haya procesos con un tamaño entre 12 y 16M en un determinado instante de tiempo. En este caso, la partición de 16M quedará sin utilizarse, incluso aunque se puede asignar dicha partición a algunos procesos más pequeños. Por tanto, una técnica óptima sería emplear una única cola para to- dos los procesos (Figura 7.3b). En el momento de cargar un proceso en la memoria principal, se selecciona la partición más pequeña disponible que puede albergar dicho proceso. Si todas las particiones están ocupadas, se debe llevar a cabo una decisión para enviar a swap a algún proceso. Tiene preferencia a la hora de ser expulsado a disco el proceso que ocupe la partición más pequeña que pue- da albergar al proceso entrante.
Los esquemas de particiones fijas son relativamente sencillos y requieren un soporte mínimo por parte del sistema operativo y una sobrecarga de procesa- miento mínimo. Sin embargo, tiene una serie de desventajas:
•El número de particiones especificadas en tiempo de generación del sistema limita el número de proceso activos (no suspendidos) del sistema.
•Debido a que los tamaños de las particiones son preestablecidos en tiempo de generación del sistema, los trabajos pequeños no utilizan el espacio de las particiones eficientemente.

PARTICIONAMIENTO DINÁMICO

Para vencer algunas de las dificultades con particionamiento fijo, se desarrolló una técnica conocida como particionamiento dinámico.
Las particiones son de longitud y número variable. Cuando se lleva un proceso a la memoria principal, se le asigna exactamente tanta memoria como requiera y no más.
El método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos pequeños en la memoria. A medida que pasa el tiempo, la memoria se fragmenta cada vez más y la utilización de la memoria se decrementa. Este fenómeno se conoce como fragmentación externa, indicando que la memoria que es externa a todas las particiones se fragmenta de forma incremental, por contraposición a lo que ocurre con la fragmentación in- terna, descrita anteriormente.
Una técnica para eliminar la fragmentación externa es la compactación: de vez en cuando, el si tema operativo desplaza los procesos en memoria, de forma que se encuentren contiguos y de este modo toda la memoria libre se encontrará unida en un bloque.. La desventaja de la compactación es el hecho de que se trata de un procedimiento que consume tiempo y malgasta tiempo de procesador. Obsérvese que la compactación requiere la capacidad de reubicación dinámica. Es decir, debe ser posible mover un programa desde una región a otra en la memoria principal sin invalidar las referencias de la memoria de cada programa

Algoritmo de ubicación

Debido a que la compactación de memoria consume una gran cantidad de tiempo, el diseñador del sistema operativo debe ser inteligente a la hora de decidir cómo asignar la memoria a los procesos
Tres algoritmos de colocación que pueden considerarse son mejor-ajuste (best-fit), primer-ajuste (first-fit) y siguiente ajuste (next-fit). Todos, por supuesto, están limitados a escoger entre los bloques libres de la memoria principal que son iguales o más grandes que el proceso que va a llevarse a la memoria. Mejor-ajuste escoge el bloque más cercano en tamaño a la petición. Primer-ajuste comienza a analizar la memoria desde el principio y escoge el primer bloque disponible que sea suficientemente grande. Siguiente-ajuste comienza a analizar la memoria desde la última colocación y elige el siguiente bloque disponible que sea suficientemente grande
Cuál de estas técnicas es mejor depende de la secuencia exacta de intercambio de procesos y del ta- maño de dichos procesos. El algoritmo primer-ajuste no es sólo el más sencillo, sino que

normalmente es también el mejor y más rápido. El algoritmo siguiente-ajuste tiende a producir resulta- dos ligeramente peores que el primer-ajuste. El algoritmo siguiente-ajuste llevará más frecuentemente a una asignación de un bloque libre al final de la memoria. El resultado es que el bloque más grande de memoria libre, que normalmente aparece al final del espacio de la memoria, se divide rápidamente en pequeños fragmentos. Por tanto, en el caso del algoritmo siguiente-ajuste se puede requerir más fre- cuentemente la compactación. Por otro lado, el algoritmo primer-ajuste puede dejar el final del espacio de almacenamiento con pequeñas particiones libres que necesitan buscarse en cada paso del primer- ajuste siguiente. El algoritmo mejor-ajuste, a pesar de su nombre, su comportamiento normalmente es el peor. Debido a que el algoritmo busca el bloque más pequeño que satisfaga la petición, garantiza que el fragmento que quede sea lo más pequeño posible. Aunque cada petición de memoria siempre malgasta la cantidad más pequeña de la memoria, el resultado es que la memoria principal se queda rápidamente con bloques demasiado pequeños para satisfacer las peticiones de asignación de la memoria. Por tanto, la compactación de la memoria se debe realizar más frecuentemente que con el resto de los algoritmos.

Un esquema de particionamiento fijo limita el número de procesos activos y puede utilizar el espacio ineficientemente si existe un mal ajuste entre los tamaños de partición disponibles y los tamaños de los procesos. Un es- quema de particionamiento dinámico es más complejo de mantener e incluye la sobrecarga de la compactación.

En un sistema buddy, los bloques de memoria disponibles son de tamaño 2k, L □ K □ U, donde

2L = bloque de tamaño más pequeño asignado

2U = bloque de tamaño mayor asignado; normalmente 2U es el tamaño de la memoria completa disponible

Para comenzar, el espacio completo disponible se trata como un único bloque de tamaño 2U. Si se realiza una petición de tamaño s, tal que 2U-1< s □ 2U, se asigna el bloque entero. En otro caso, el bloque se divide en dos bloques buddy iguales de tamaño 2U-1. Si 2U-2< s □ 2U-1, entonces se asigna la petición a uno de los otros dos bloques. En otro caso, uno de ellos se divide por la mitad de nuevo. Este proceso continúa hasta que el bloque más pequeño mayor o igual que s se genera y se asigna a la petición. En cualquier momento, el sistema buddy mantiene una lista de huecos (bloques sin asignar) de cada tamaño 2i. Un hueco puede eliminarse de la lista (i+1) dividiéndolo por la mitad para crear dos bloques de tamaño 2i en la lista i. Siempre que un par de bloques de la lista i no se encuentren asigna- dos, son eliminados de dicha lista y unidos en un único bloque de la lista (i+1).

PAGINACIÓN

Tanto las particiones de tamaño fijo como variable son ineficientes en el uso de la memoria; los pri- meros provocan fragmentación interna, los últimos fragmentación externa. Supóngase, sin embargo, que la memoria principal se divide en porciones de tamaño fijo relativamente pequeños, y que cada proceso también se divide en porciones pequeñas del mismo tamaño fijo. A dichas porciones del proceso, conocidas como páginas, se les asigna porciones disponibles de memoria, conocidas como marcos, o marcos de páginas. Esta sección muestra que el espacio de memoria malgastado por cada proceso debido a la fragmentación interna corresponde sólo a una fracción de la última página de un proceso. No existe fragmentación externa.

. En un momento dado, algunos de los marcos de la memoria están en uso y algunos están libres. El sistema operativo mantiene una lista de mar- cos libres. El proceso A, almacenado en disco, está formado por cuatro páginas. En el momento de cargar este proceso, el sistema operativo encuentra cuatro marcos libres y carga las cuatro pá- ginas del proceso A en dichos marcos

Ahora supóngase que no hay suficientes marcos contiguos sin utilizar para ubicar un proceso. ¿Esto evitaría que el sistema operativo cargara el proceso D? La respuesta es no, porque una vez más se puede utilizar el concepto de dirección lógica. Un registro base senci- llo de direcciones no basta en esta ocasión. En su lugar, el sistema operativo mantiene una tabla de

páginas por cada proceso. La tabla de páginas muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página. Es importante recordar que en el caso de una partición simple, una dirección lógica es la ubicación de una palabra relativa al comienzo del programa; el procesador la traduce en una dirección física. Con paginación, la traducción de direcciones lógicas a físicas las realiza también el hardware del procesador. Ahora el procesador debe conocer cómo acceder a la tabla de páginas del proceso actual. Presentado como una dirección lógica (número de página, desplazamiento), el procesador utiliza la tabla de páginas para producir una dirección física (número de marco, desplazamiento).
Una tabla de páginas contiene una entrada por cada página del proceso, de forma que la tabla se indexe fácilmente por el número de página (iniciando en la página 0). Cada entrada de la tabla de páginas contiene el número del marco en la memoria principal, si existe, que contiene la página correspondiente, Adicionalmente, el sistema operativo mantiene una única lista de marcos libres de todos los marcos de la memoria que se encuentran actualmente no ocupados y disponibles para las páginas.
Por tanto vemos que la paginación simple, tal y como se describe aquí, es similar al particiona- miento fijo. Las diferencias son que, con la paginación, las particiones son bastante pequeñas; un pro- grama podría ocupar más de una partición; y dichas particiones no necesitan ser contiguas.
Resumiendo, con la paginación simple, la memoria principal se divide en muchos marcos pequeños de igual tamaño. Cada proceso se divide en páginas de igual tamaño; los procesos más pequeños requieren menos páginas, los procesos mayores requieren más. Cuando un proceso se trae a la memoria, todas sus páginas se cargan en los marcos disponibles, y se establece una tabla de páginas. Esta técnica resuelve muchos de los problemas inherentes en el particionamiento.

SEGMENTACIÓN

Un programa de usuario se puede subdividir utilizando segmentación, en la cual el programa y sus datos asociados se dividen en un número de segmentos. No se requiere que todos los programas sean de la misma longitud, aunque hay una longitud máxima de segmento. Como en el caso de la paginación, una dirección lógica utilizando segmentación está compuesta por dos partes, en este caso un número de segmento y un desplazamiento.
Debido al uso de segmentos de distinto tamaño, la segmentación es similar al particionamiento dinámico. En la ausencia de un esquema de overlays o el uso de la memoria virtual, se necesitaría que todos los segmentos de un programa se cargaran en la memoria para su ejecución. La diferencia, comparada con el particionamiento dinámico, es que con la segmentación un programa podría ocupar más de una partición, y estas particiones no necesitan ser contiguas. La segmentación elimina la frag- mentación interna pero, al igual que el particionamiento dinámico, sufre de fragmentación externa. Sin embargo, debido a que el proceso se divide en varias piezas más pequeñas, la fragmentación ex- terna debería ser menor.
Mientras que la paginación es invisible al programador, la segmentación es normalmente visible y se proporciona como una utilidad para organizar programas y datos.
El inconveniente principal de este servicio es que el programador debe ser consciente de la limitación de tamaño de segmento máximo.
Otra consecuencia de utilizar segmentos de distinto tamaño es que no hay una relación simple entre direcciones lógicas y direcciones físicas. De forma análoga a la paginación, un esquema de segmentación sencillo haría uso de una tabla de segmentos por cada proceso y una lista de bloques libre de memoria principal. Cada entrada de la tabla de segmentos tendría que proporcionar la dirección inicial de la memoria principal del correspondiente segmento. La entrada también debería proporcionar la longitud del segmento, para asegurar que no se utilizan direcciones no válidas. Cuando un pro- ceso entra en el estado Ejecutando, la dirección

de su tabla de segmentos se carga en un registro especial utilizado por el hardware de gestión de la memoria.

Memoria Virtual

Paginación sencilla	Paginación con memoria virtual	Segmentación sencilla	Segmentación con memoria virtual
Memoria principal particionada en fragmentos pequeños de un tamaño fijo llamados marcos	Memoria principal particionada en fragmentos pequeños de un tamaño fijo llamados marcos	Memoria principal no particionada	Memoria principal no particionada
Programa dividido en páginas por el compilador o el sistema de gestión de la memoria	Programa dividido en páginas por el compilador o el sistema de gestión de la memoria	Los segmentos de programa se especifican por el programador al compilador (por ejemplo, la decisión se toma por parte el programador)	Los segmentos de programa se especifican por el programador al compilador (por ejemplo, la decisión se toma por parte el programador)
Fragmentación interna dentro de los marcos	Fragmentación interna dentro de los marcos	Sin fragmentación interna	Sin fragmentación interna
Sin fragmentación externa	Sin fragmentación externa	Fragmentación externa	Fragmentación externa
El sistema operativo debe mantener una tabla de páginas por cada proceso mostrando en el marco que se encuentra cada página ocupada	El sistema operativo debe mantener una tabla de páginas por cada proceso mostrando en el marco que se encuentra cada página ocupada	El sistema operativo debe mantener una tabla de segmentos por cada proceso mostrando la dirección de carga y la longitud de cada segmento	El sistema operativo debe mantener una tabla de segmentos por cada proceso mostrando la dirección de carga y la longitud de cada segmento
El sistema operativo debe mantener una lista de marcos libres	El sistema operativo debe mantener una lista de marcos libres	El sistema operativo debe mantener una lista de huecos en la memoria principal	El sistema operativo debe mantener una lista de huecos en la memoria principal
El procesador utiliza el número de página, desplazamiento para calcular direcciones absolutas	El procesador utiliza el número de página, desplazamiento para calcular direcciones absolutas	El procesador utiliza el número de segmento, desplazamiento para calcular direcciones absolutas	El procesador utiliza el número de segmento, desplazamiento para calcular direcciones absolutas
Todas las páginas del proceso deben encontrarse en la memoria principal para que el proceso se pueda ejecutar, salvo que se utilicen solapamientos (*overlays*)	No se necesita mantener todas las páginas del proceso en los marcos de la memoria principal para que el proceso se ejecute. Las páginas se pueden leer bajo demanda	Todos los segmentos del proceso deben encontrarse en la memoria principal para que el proceso se pueda ejecutar, salvo que se utilicen solapamientos (*overlays*)	No se necesitan mantener todos los segmentos del proceso en la memoria principal para que el proceso se ejecute. Los segmentos se pueden leer bajo demanda
	La lectura de una página a memoria principal puede requerir la escritura de una página a disco		La lectura de un segmento a memoria principal puede requerir la escritura de uno o más segmentos a disco

HARDWARE Y ESTRUCTURAS DE CONTROL

Las dos características de la paginación y la segmentación que son la clave de este comienzo son:

1. 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 pro- ceso 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.
2. 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 combi- nación de la traducción de direcciones dinámicas en ejecución y el uso de una tabla de páginas o segmentos lo permite.

Si las dos características anteriores se dan, entonces es necesario que todas las páginas o todos los segmentos de un proceso se encuentren en la memoria principal durante la ejecución. Si la porción (segmento o página) en la que se encuentra la siguiente instrucción a buscar está y si la porción donde se encuentra la siguiente dirección de datos que se va a acceder también está, entonces al menos la siguiente instrucción se podrá ejecutar.

El término porción para referirnos o bien a una página o un segmento, dependiendo si estamos empleando paginación o segmentación. 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. Cuando el proceso está ejecutándose, las cosas ocurren de forma suave mientras que todas las referencias a la memoria se encuentren dentro del conjunto residente. Usando una tabla de segmentos o páginas, el procesador siempre es capaz de determinar si esto es así o no. Si el procesador encuentra una dirección lógica que no se encuentra en la memoria principal, generará una interrupción indicando un fallo de acceso a la memoria. El sistema operativo coloca al proceso interrumpido en un estado de bloqueado y toma el control. Para que la ejecución de este proceso pueda reanudarse más adelante, el sistema operativo necesita traer a la memoria principal la porción del proceso que contiene la dirección lógica que ha causado el fallo de acceso. Con este fin, el sistema operativo realiza una petición de E/S, una lectura a disco. Después de realizar la petición de E/S, el sistema operativo puede activar otro proceso que se ejecute mientras el disco realiza la operación de E/S. Una vez que la porción solicitada se ha traído a la memoria principal, una nueva interrupción de E/S se lanza, dando control de nuevo al sistema operativo, que coloca al proceso afectado de nuevo en el estado Listo.

Existen dos implicaciones, la segunda más sorprendente que la primera, y ambas dirigidas a mejorar la utilización del sistema:

1.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 pro- bable que haya al menos uno o más de los numerosos procesos que se encuentre en el estado Listo, en un instante de tiempo concreto.

2.Un proceso puede ser mayor que toda la memoria principal. Se puede superar una de las restricciones fundamentales de la programación. Con la memoria virtual basada en paginación o segmentación, este trabajo se delega al sistema operativo y al hardware.

Debido a que un proceso ejecuta sólo en la memoria principal, esta memoria se denomina memoria real. Pero el programador o el usuario perciben una memoria potencialmente mucho más grande

—la cual se encuentra localizada en disco. Esta última se denomina memoria virtual. La memoria virtual permite una multiprogramación muy efectiva que libera al usuario de las restricciones excesivamente fuertes de la memoria principal.

PAGINACIÓN

El término memoria virtual se asocia habitualmente con sistemas que emplean paginación, a pesar de que la memoria virtual basada en segmentación.

En la presentación de la paginación sencilla, indicamos que cada proceso dispone de su propia tabla de páginas, y que todas las páginas se encuentran localizadas en la memoria principal. Cada entra- da en la tabla de páginas consiste en un número de marco de la correspondiente página en la memoria principal. Para la memoria virtual basada en el esquema de paginación también se necesita una tabla de páginas. De nuevo, normalmente se asocia una única tabla de páginas a cada proceso. En este caso, sin embargo, las entradas de la tabla de páginas son más complejas .Debido a que sólo algunas de las páginas de proceso se encuentran en la memoria principal, se necesita que cada entrada de la tabla de páginas indique si la correspondiente página está presente (P) en memoria principal o no. Si el bit indica que la página está en memoria, la entrada también debe indicar el número de marco de dicha página.

La entrada de la tabla de páginas incluye un bit de modificado (M), que indica si los contenidos de la correspondiente página han sido alterados desde que la página se cargó por última vez en la memoria principal. Si no había ningún cambio, no es necesario escribir la página cuando llegue el momento de reemplazarla por otra página en el marco de página que actualmente ocupa.

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 des plazamiento, 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. La Figura 8.3 sugiere una implementación hardware. Cuando un proceso en particular se encuentra ejecutando, un registro con- tiene la dirección de comienzo de la tabla de páginas para dicho proceso. El número de página de la dirección virtual se utiliza para indexar esa tabla y buscar el correspondiente marco de página. Éste, combinado con la parte de desplazamiento de la dirección virtual genera la dirección real deseada.

En la mayoría de sistemas, existe una única tabla de página por proceso. Pero cada proceso puede ocupar una gran cantidad de memoria virtual.

Tabla de páginas invertida. Una desventaja del tipo de tablas de páginas que hemos visto es que su tamaño es proporcional al espacio de direcciones virtuales.

Una estrategia alternativa al uso de tablas de páginas de uno o varios niveles es el uso de la estructura de tabla de páginas invertida.

En esta estrategia, la parte correspondiente al número de página de la dirección virtual se referencia por medio de un valor hash usando una función hash sencilla1. El valor hash es un puntero para la tabla de páginas invertida, que contiene las entradas de tablas de página. Hay una entrada en la tabla de páginas invertida por cada marco de página real en lugar de uno por cada página virtual. De esta forma, lo único que se requiere para estas tablas de página siempre es una proporción fija de la memoria real, independientemente del número de procesos o de las páginas virtuales so- portadas. Las técnicas de hashing proporcionan habitualmente cadenas que no son excesivamente largas —entre una y dos entradas. La estructura de la tabla de páginas se denomina invertida debido a que se indexan sus entradas de la tabla de páginas por el número de marco en lugar de por el número de página virtual.

La Figura 8.6 muestra una implementación típica de la técnica de tabla de páginas invertida. Para un tamaño de memoria física de 2m marcos, la tabla de páginas invertida contiene 2m entradas, de forma que la entrada en la posición i-esima se refiere al marco i. La entrada en la tabla de páginas incluye la siguiente información:

•Número de página. Esta es la parte correspondiente al número de página de la dirección virtual.

•Identificador del proceso. El proceso que es propietario de está página. La combinación de número de página e identificador del proceso identifica a una página dentro del espacio de direcciones virtuales de un proceso en particular.

•Bits de control. Este campo incluye los flags.

•Puntero de la cadena. Este campo es nulo (indicado posiblemente por un bit adicional) si no hay más entradas encadenadas en esta entrada. En otro caso, este campo contiene el valor del índice (número entre 0 y 2m-1) de la siguiente entrada de la cadena.

Buffer de traducción anticipada. En principio, toda referencia a la memoria virtual puede causar dos accesos a memoria física: uno para buscar la entrada la tabla de páginas apropiada y otro para buscar los datos solicitados. De esa forma, un esquema de memoria virtual básico causaría el efecto de duplicar el tiempo de acceso a la memoria. Para solventar este problema, la mayoría de esquemas de la memoria virtual utilizan una cache especial de alta velocidad para las entradas de la tabla de página, habitualmente denominada buffer de traducción anticipada (translation lookaside buffer - TLB)2. Esta cache funciona de forma similar a una memoria cache general (véase Capítulo 1) y con- tiene aquellas entradas de la tabla de páginas que han sido usadas de forma más reciente. La organización del hardware de paginación resultante se ilustra en la Figura 8.7. 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 solicita- da no se encuentra en la memoria principal y se produce un fallo de acceso memoria, llamado fallo de página. En este punto, abandonamos el dominio del hardware para invocar al sistema operativo, el cual cargará la página necesaria y actualizada de la tabla de páginas.

Tamaño de página. Una decisión de diseño hardware importante es el tamaño de página a usar. 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

Otro factor importante son las características físicas de la mayoría de los dispositivos de la memoria secundaria, que son de tipo giratorio, favoreciendo tamaños de página grandes para mejorar la eficiencia de transferencia de bloques de datos.

Aumentando la complejidad de estos aspectos se encuentra el efecto que el tamaño de página tiene en relación a la posibilidad de que ocurra un fallo de página. Si el tamaño de página es muy pequeño, de forma habitual habrá un número relativamente alto de páginas disponibles en la memoria principal para cada proceso. Después de un tiempo, las páginas en memoria contendrán las partes de los procesos a las que se ha hecho referencia de forma reciente. De esta forma, la tasa de fallos de página debería ser baja. A medida que el tamaño de páginas se incrementa, la página en particular contendrá información más lejos de la última referencia realizada. Así pues, el efecto del principio de proximidad se debilita y la tasa de fallos de página comienza a crecer. En algún momento, sin embargo, la tasa de fallos de página comenzará a caer a medida que el tamaño de la página se aproxima al tamaño del proceso completo (punto P en el diagrama). Cuando una única página contiene el proceso completo, no habrá fallos de página.

Una complicación adicional es que la tasa de fallos de página también viene determinada por el número de marcos asociados a cada proceso. La Figura 8.11b muestra que, para un tamaño de página fijo, la tasa de fallos cae a medida que el número de páginas mantenidas en la memoria principal crece.

Para concluir, el aspecto de diseño del tamaño página se encuentra relacionado con el tamaño de la memoria física y el tamaño del programa. Al mismo tiempo que la memoria principal está siendo cada vez más grande, el espacio de direcciones utilizado por las aplicaciones también crece

SEGMENTACIÓN

Las implicaciones en la memoria virtual. La segmentación permite al programador ver la memoria como si se tratase de diferentes espacios de direcciones o segmentos. Los segmentos pueden ser de tamaños diferentes, en realidad de tamaño dinámico. Una referencia a la memoria consiste en un formato de dirección del tipo (número de segmento, desplazamiento). Esta organización tiene un gran número de ventajas para el programador sobre los espacios de direcciones no segmentados:

1. Simplifica el tratamiento de estructuras de datos que pueden crecer
2. 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..
3. 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 des- de otros procesos.

4. Soporta los mecanismos de protección.

Organización. 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 entra- da 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 (Figura 8.2b). 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.

Otro bit de control en la entrada de la tabla de segmentos es el bit de modificado, que indica si los contenidos del segmento correspondiente se han modificado desde que se cargó por última vez en la memoria principal. Si no hay ningún cambio, no es necesario escribir el segmento cuando se reemplace de la memoria principal.

El mecanismo básico para la lectura de una palabra de memoria implica la traducción de una dirección virtual, o lógica, consistente en un número de segmento y un desplazamiento, en una dirección física, usando la tabla de segmentos. Debido a que la tabla de segmentos es de tamaño variable, dependiendo del tamaño del proceso, no se puede suponer que se encuentra almacenada en un registro. En su lugar, debe encontrarse en la memoria principal para poder accederse. Cuando un proceso en particular está en ejecución, un registro mantiene la dirección de comienzo de la tabla de segmentos para dicho proceso. El número de segmento de la dirección virtual se utiliza para indexar esta tabla y para buscar la dirección de la memoria principal donde comienza dicho segmento. Ésta es añadida a la parte de desplazamiento de la dirección virtual para producir la dirección real solicitada.

PAGINACIÓN Y SEGMENTACIÓN COMBINADAS

Paginación y segmentación, cada una tiene sus propias ventajas. La paginación es transparente al programador y elimina la fragmentación externa, y por tanto proporciona un uso eficiente de la memoria principal. Adicionalmente, debido a que los fragmentos que se mueven entre la memoria y el disco son de un tamaño igual y prefijado, es posible desarrollar algoritmos de gestión de la memoria más sofisticados que exploten el comportamiento de los programas, como veremos más adelante. La segmentación sí es visible al programador y tiene los beneficios que hemos visto anterior- mente, incluyendo la posibilidad de manejar estructuras de datos que crecen, modularidad, y dar soporte a la compartición y a la protección. Para combinar las ventajas de ambos, algunos sistemas por medio del hardware del procesador y del soporte del sistema operativo son capaces de proporcionar ambos.

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.

SOFTWARE DEL SISTEMA OPERATIVO

El diseño de la parte de la gestión de la memoria del sistema operativo depende de tres opciones fundamentales a elegir:

- Si el sistema usa o no técnicas de memoria virtual.
- El uso de paginación o segmentación o ambas.
- Los algoritmos utilizados para los diferentes aspectos de la gestión de la memoria.

Las elecciones posibles para las dos primeras opciones dependen de la plataforma hardware disponible. Hay también dos comentarios adicionales sobre estas dos primeras opciones: primero, con la excepción de los sistemas operativos de algunas plataformas y de otros sistemas de carácter especializado, todos los sistemas operativos importantes proporcionan memoria virtual

En cada caso, el aspecto central es el rendimiento: Se tratará de minimizar la tasa de ocurrencia de fallos de página, porque los fallos de página causan una considerable sobrecarga sobre el software. Como mínimo, esta sobrecarga incluye la decisión de qué página o páginas residentes se van a reemplazar, y la E/S del intercambio o swap de dichas páginas. También, el sistema operativo debe planificar la ejecución de otro proceso durante la operación de E/S de la página, causando un cambio de contexto. Como se verá, la tarea de gestión de la memoria en un entorno de paginación es endiabladamente compleja. Adicionalmente, el rendimiento de un conjunto de políticas en particular depende del tamaño de la memoria, de la velocidad relativa de la memoria principal y secundaria, del tamaño y del número de procesos que están compitiendo por los recursos, y del comportamiento en ejecución de los diferentes programas de forma individual.

POLÍTICA DE RECUPERACIÓN

La política de recuperación determina cuándo una página se trae a la memoria principal. Las dos alternativas habituales son bajo demanda y paginación adelantada (prepaging). 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.

Con paginación adelantada (prepaging), se traen a memoria también otras páginas, diferentes de la que ha causado el fallo de página. La paginación adelantada tiene en cuenta las características que tienen la mayoría de dispositivos de memoria secundaria, tales como los discos, que tienen tiempos de búsqueda y latencia de rotación. Si las páginas de un proceso se encuentran almacenadas en la memoria secundaria de forma contigua, es mucho más eficiente traer a la memoria un número de páginas contiguas de una vez, en lugar de traerlas una a una a lo largo de un periodo de tiempo más amplio. Por supuesto, esta política es ineficiente si la mayoría de las páginas que se han traído no se referencian a posteriori.

La política de paginación adelantada puede emplearse bien cuando el proceso se arranca, en cuyo caso el programador tiene que designar de alguna forma las páginas necesarias, o cada vez que ocurra un fallo de página. Este último caso es el más apropiado porque resulta completamente invisible al programador.

La paginación adelantada no se debe confundir con el swapping. Cuando un proceso se saca de la memoria y se le coloca en estado suspendido, todas sus páginas residentes se expulsan de la memoria. Cuando el proceso se recupera, todas las páginas que estaban previamente en la memoria principal retornan a ella.

POLÍTICA DE UBICACIÓN

La política de ubicación determina en qué parte de la memoria real van a residir las porciones de la memoria de un proceso. En los sistemas de segmentación puros, la política de ubicación es un aspecto de diseño muy importante.

POLÍTICA DE REEMPLAZO

Trata de la selección de una página en la memoria principal como candidata para reemplazarse cuando se va traer una nueva página.
Nos referimos a los dos primeros conceptos como la gestión del conjunto residente, que se trata en la siguiente subsección, y se ha reservado el término política de reemplazo para el tercer concepto, que se discutirá en esta misma subsección.
Cuando todos los marcos de la memoria principal están ocupados y es necesario traer una nueva página para resolver un fallo de página, la política de reemplazo determina qué página de las que actualmente están en memoria va a reemplazarse. Todas las políticas tienen como objetivo que la página que va a eliminarse sea aquella que tiene menos posibilidades de volver a tener una referencia en un futuro próximo. Así, la mayoría de políticas tratan de predecir el comportamiento futuro en base al comportamiento pasado. En contraprestación, se debe considerar que cuanto más elaborada y sofisticada es una política de reemplazo, mayor va a ser la sobrecarga a nivel software y hardware para implementarla
Algoritmos básicos. Independientemente de la estrategia de gestión del conjunto residente (que se discutirá en la siguiente subsección), existen ciertos algoritmos básicos que se utilizan para la selección de la página a reemplazar. Los algoritmos de reemplazo que se han desarrollando a lo largo de la literatura son:
• Óptimo.
• Usado menos recientemente (least recently used-LRU).
• FIFO (first-in-first-out).
• Reloj.
La política óptima de selección tomará como reemplazo la página para la cuál el instante de la siguiente referencia se encuentra más lejos. Se puede ver que para esta política los resultados son el menor número de posibles fallos de página [BELA66]. Evidentemente, 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 apartir 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. Y, de hecho, la política LRU proporciona unos resultados casi tan buenos como la política óptima. El problema con esta alternativa es la dificultad en su implementación. Una opción sería etiquetar cada página con el instante de tiempo de su última referencia; esto podría ser en cada una de las referencias a la memoria, bien instrucciones o datos. Incluso en el caso de que el hardware diera soporte a dicho esquema, la sobrecarga sería tremenda. De forma alternativa se puede mantener una pila de referencias a páginas, que igualmente es una opción costosa
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.
Mientras que la política LRU alcanza unos resultados similares a la política óptima, es difícil de implementar e impone una sobrecarga significativa. Por otro lado, la política FIFO es muy sencilla de implementar pero su rendimiento es relativamente pobre. A lo largo de los años, los diseñadores de sistemas operativos han intentado un gran número de algoritmos diferentes para aproximarse a los resultados obtenidos por LRU e intentando imponer una

sobrecarga más reducida. Muchos de estos algoritmos son variantes del esquema denominado política del reloj.
En su forma más sencilla la política del reloj requiere la inclusión de un bit adicional en cada uno de los marcos de página, denominado bit de usado. Cuando una página se trae por primera vez a la memoria, el bit de usado de dicho marco se pone a 1. En cualquier momento que la página vuelva a utilizarse (después de la referencia generada con el fallo de página inicial) su bit de usado se pone a 1.Para el algoritmo de reemplazo de páginas, el conjunto de todas las páginas que son candidatas para reemplazo (de este proceso: ámbito local; toda la memoria principal: ámbito global6) se disponen como si se tratase de un buffer circular, al cual se asocia un puntero. Cuando se reemplaza una página, el puntero indica el siguiente marco del buffer justo después del marco que acaba de actualizarse. Cuando llega el momento de reemplazar una página, el sistema operativo recorre el buffer para encontrar un marco con su bit de usado a 0. Cada vez que encuentra un marco con el bit de usado a 1, se reinicia este bit a 0 y se continúa. Si alguno de los marcos del buffer tiene el bit de usado a 0 al comienzo de este proceso, el primero de estos marcos que se encuentre se seleccionará para reemplazo. Si todos los marcos tienen el bit a 1, el puntero va a completar un ciclo completo a lo largo del buffer, poniendo todo los bits de usado a 0, parándose en la posición original, reemplazando la página en dicho marco. Véase que esta política es similar a FIFO, excepto que, en la política del reloj, el algoritmo saltará todo marco con el bit de usado a 1. La política se domina política del reloj debido a que se pueden visualizar los marcos de página como si estuviesen distribuidos a lo largo del círculo.
Las diferencias entre las cuatro políticas son más palpables cuando el número de marcos reservados es pequeño, estando FIFO por encima en un factor de 2 ve- ces peor que el óptimo. Con intención de ejecutar de forma eficiente, sería deseable encontrarse en el lado derecho de la curva (con una tasa de fallos de página pequeña) mientras que al mismo tiempo se mantiene una necesidad de reservar relativamente pocos marcos (hacia el lado izquierdo de la curva). Estas dos restricciones indican que el modo deseable de operación estaría aproximadamente en la mitad de la curva.
El algoritmo del reloj puede hacerse más potente incrementando el número de bits que utiliza7. En todos los procesadores que soportan paginación, se asocia un bit de modificado a cada una de las páginas de la memoria principal y por tanto con cada marco de la memoria principal. Este bit es necesario debido a que, cuando una página se ha modificado, no se la puede reemplazar hasta que se haya escrito de nuevo a la memoria secundaria. Podemos sacar provecho de este bit en el algoritmo del reloj de la siguiente manera. Si tenemos en cuenta los bits de usado y modificado, cada marco de página cae en una de las cuatro categorías siguientes:
• No se ha accedido recientemente, no modificada (u = 0; m = 0)
• Accedida recientemente, no modificada (u = 1; m = 0)
• No se ha accedido recientemente, modificada (u = 0; m = 1)
• Accedida recientemente, modificada (u = 1; m = 1)
Con esta clasificación, el algoritmo del reloj puede actuar de la siguiente manera:
1. Comenzando por la posición actual del puntero, recorremos el buffer de marcos. Durante el re- corrido, no se hace ningún cambio en el bit de usado. El primer marco que se encuentre con (u= 0; m = 0) se selecciona para reemplazo.
2. Si el paso 1 falla, se recorre el buffer de nuevo, buscando un marco con (u = 0; m = 1). El primer marco que se encuentre se seleccionará para reemplazo. Durante el recorrido, se pondrá el bit de usado a 0 en cada uno de los marcos que se vayan saltando.
3. Si el paso 2 también falla, el puntero debe haber vuelto a la posición original y todo los marcos en el conjunto tendrán el bit de usado a 0. Se repite el paso 1 y, si resulta necesario el paso 2. Esta vez, se encontrará un marco para reemplazo.
En resumen, el algoritmo de reemplazo de páginas 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. Incluso aunque esta página tenga que escribirse antes de ser remplazada, debido al principio de proximidad de referencia, puede no necesitarse de nuevo en el futuro próximo. Si esta segunda pasada falla, todos los marcos en el buffer se encuentran marcados como si no hubiesen sido accedidos recientemente y se realiza una ter- cera pasada.
La ventaja de este algoritmo sobre el algoritmo del reloj básico es que se les otorga preferencia para el reemplazo a las páginas que no se han modificado. Debido a que la página que se ha modificado debe escribirse antes del reemplazo, hay un ahorro de tiempo inmediato.

Buffering páginas. A pesar de que las políticas LRU y del reloj son superiores a FIFO, ambas incluyen una complejidad y una sobrecarga que FIFO no sufre. Adicionalmente, existe el aspecto relativo a que el coste de reemplazo de una página que se ha modificado es superior al de una que no lo ha sido, debido a que la primera debe escribirse en la memoria secundaria.

GESTIÓN DEL CONJUNTO RESIDENTE
Tamaño del conjunto residente. Con la memoria virtual paginada, no es necesario, y en algunos casos no es ni siquiera posible, traer todas las páginas de un proceso a la memoria principal para pre- parar su ejecución. Debido a que el sistema operativo debería saber cuántas páginas traerse, esto es, cuánta memoria principal debería reservar para un proceso en particular. Diferentes factores entran en juego:
• Cuanto menor es la cantidad de memoria reservada para un proceso, mayor es el número de procesos que pueden residir en la memoria principal a la vez.
• Si el conjunto de páginas de un proceso que están en memoria es relativamente pequeño, entonces, en virtud del principio de proximidad de referencia, la posibilidad de un fallo de página es mayor.
• Más allá de un determinado tamaño, la reserva de más memoria principal para un determinado proceso no tendrá un efecto apreciable sobre la tasa de fallos de página de dicho proceso, debido al principio de proximidad de referencia.
Teniendo en cuenta estos factores, se pueden encontrar dos tipos de políticas existentes en los sistemas operativos contemporáneos. La política de asignación fija proporciona un número fijo de marcos de memoria principal disponibles para ejecución. Este número se decide en el momento de la carga inicial de proceso (instante de creación del proceso) y se puede determinar en base al tipo de proceso (interactivo, por lotes, tipo de aplicación) o se puede basar en las guías proporcionadas por el programador o el administrador del sistema. Con la política de asignación fija, siempre que se produzca un fallo de página del proceso en ejecución, la página que se necesite reemplazará una de las páginas del proceso.
Una política de asignación variable permite que se reserven un número de marcos por proceso que puede variar a lo largo del tiempo de vida del mismo. De forma ideal, a un proceso que esté causando una tasa de fallos de página relativamente alta de forma continua, indicativo de que el principio proximidad de referencia sólo se aplica de una forma relativamente débil para este proceso, se le otorgarán marcos de página adicionales para reducir esta tasa de fallos; mientras tanto, a un proceso con una tasa de fallos de páginas excepcionalmente baja, indicativo de que el proceso sigue un comportamiento bien ajustado al principio de proximidad de referencia, se reducirán los marcos reserva- dos, con esperanza de que esto no incremente de forma apreciable la tasa de fallos. El uso de políticas de asignación variable se basa en el concepto de ámbito de reemplazo, como se explicará en la siguiente subsección.
Ámbito de reemplazo. La estrategia del ámbito de reemplazo se puede clasificar en global y local. Ambos tipos de políticas se activan por medio de un fallo de página cuando no existen

marcos de página libres. Una política de reemplazo local selecciona únicamente entre las páginas residentes del proceso que ha generado el fallo de página. Para la identificación de la página a reemplazar en una política de reemplazo global se consideran todas las páginas en la memoria principal que no se encuentren bloqueadas como candidatos para el reemplazo, independientemente de a qué proceso pertenezca cada página en particular.
Existe una correlación entre el ámbito de reemplazo y el tamaño del conjunto residente (Tabla 8.4). Un conjunto residente fijo implica automáticamente una política de reemplazo local: para mantener el tamaño de conjunto residente, al reemplazar una página se debe eliminar de la memoria principal otra del mismo proceso. Una política de asignación variable puede emplear claramente la política de reemplazo global: el reemplazo de una página de un proceso en la memoria principal por la de otro causa que la asignación de memoria para un proceso crezca en una página mientras que disminuye la del otro.
Asignación fija, ámbito local. En este caso, se parte de un proceso que se encuentra en ejecución en la memoria principal con un número de marcos fijo. Cuando se da un fallo de página, el sistema operativo debe elegir una página entre las residentes del proceso actual para realizar el reemplazo. Se utilizarían los algoritmos de reemplazo que se han visto en la sección precedente.
Con la política de asignación fija, es necesario decidir por adelantado la cantidad de espacio reservado para un proceso determinado. Esto se puede hacer en base al tipo de aplicación y al tamaño del programa. Las desventajas de esta estrategia son de dos tipos: si las reservas resultan ser demasiado pequeñas, va a haber una alta tasa de fallos de página, haciendo que el sistema multiprogramado completo se ralentice. Si las reservas, por contra, resultan demasiado grandes, habrá muy pocos pro- gramas en la memoria principal y habrá mucho tiempo del procesador ocioso o mucho tiempo perdido en swapping.
Asignación variable, ámbito global. Esta combinación es, probablemente, la más sencilla de implementar y ha sido adoptada por un gran número de sistemas operativos. En un momento determina- do, existen un número de procesos determinado en la memoria principal, cada uno de los cuales tiene una serie de marcos asignados. Normalmente, el sistema operativo también mantiene una lista de marcos libres. Cuando sucede un fallo de página, se añade un marco libre al conjunto residente de un proceso y se trae la página a dicho marco. De esta forma, un proceso que sufra diversos fallos de página crecerá gradualmente en tamaño, lo cual debería reducir la tasa de fallos de página global del sistema.
La dificultad de esta estrategia se encuentra en la elección de los reemplazos cuando no existen marcos libres disponibles, el sistema operativo debe elegir una página que actualmente se encuentra en la memoria para reemplazarla. Esta selección se lleva a cabo entre todos los marcos que se encuentran en la memoria principal, a excepción de los marcos bloqueados como son los usados por el núcleo. Utilizando cualquiera de las políticas vistas en la sección anterior, se tiene que la página seleccionada para reemplazo puede pertenecer a cualquiera de los procesos residentes; no existe ninguna disciplina predeterminada que indique qué proceso debe perder una página de su conjunto residente. Así pues, el proceso que sufre la reducción del tamaño de su conjunto residente no tiene porqué ser el óptimo.
Asignación variable, ámbito local. La asignación variable con reemplazo de ámbito local intenta resolver los problemas de la estrategia de ámbito global. Se puede resumir en lo siguiente:
1. Cuando se carga un nuevo proceso en la memoria principal, se le asignan un cierto número de marcos de página a su conjunto residente, basando en el tipo de aplicación, solicitudes del programa, u otros criterios. Para cubrir esta reserva se utilizará la paginación adelantada o la paginación por demanda.
2. Cuando ocurra un fallo página, la página que se seleccionará para reemplazar pertenecerá al conjunto residente del proceso que causó el fallo.
3. De vez en cuando, se reevaluará la asignación proporcionada a cada proceso, incrementándose o reduciéndose para mejorar al rendimiento

Los elementos clave en la estrategia de asignación variable con ámbito local son los criterios que se utilizan para determinar el tamaño del conjunto residente y la periodicidad de estos cambios. Una estrategia específica que ha atraído mucha atención en la literatura es la denominada estrategia del conjunto de trabajo. A pesar de que la estrategia del conjunto de trabajo pura sería difícil implementar, es muy útil examinarla como referencia para las comparativas

El conjunto de trabajo con parámetro para un proceso en el tiempo virtual t, W(t,) es el conjunto de páginas del proceso a las que se ha referenciado en las últimas unidades de tiempo virtual.

El tiempo virtual se define de la siguiente manera. Considérese la secuencia de referencias a memoria, r(1), r(2), ... , en las cuales r(i) es la página que contiene la i-esima dirección virtual generada por dicho proceso. El tiempo se mide en referencias a memoria; así t=1,2,3... mide el tiempo virtual interno del proceso.

Se considera que cada una de las dos variables de W. La variable es la ventana de tiempo virtual a través de la cual se observa al proceso. El tamaño del conjunto trabajo será una función nunca decreciente del tamaño de ventana. El resultado que se muestra en la Figura 8.19 (en base a [BACH86]), demuestra la secuencia de referencias a páginas para un proceso. Los puntos indican las unidades de tiempo en las cuales el conjunto de trabajo no cambia. Nótese que para mayor tamaño ventana, el tamaño del conjunto trabajo también es mayor. Esto se puede expresar en la siguiente relación: W(t, +1) W(t,)

El conjunto trabajo es también una función del tiempo. Si un proceso ejecuta durante unidades de tiempo, y terminando el tiempo utiliza una única página, entonces |W(t,)|=1. Un conjunto traba- jo también puede crecer hasta llegar a las N páginas del proceso si se accede rápidamente a muchas páginas diferentes y si el tamaño de la ventana lo permite. De esta forma, 1 |W(t,)| min(,N)

Este concepto del conjunto de trabajo se puede usar para crear la estrategia del tamaño conjunto residente:

1. Monitorizando el conjunto de trabajo de cada proceso.
2. Eliminando periódicamente del conjunto residente aquellas páginas que no se encuentran en el conjunto de trabajo, esto en esencia es una política LRU.
3. Un proceso puede ejecutar sólo si su conjunto trabajo se encuentra en la memoria principal (por ejemplo, si su conjunto residente incluye su conjunto de trabajo).

Esta estrategia funciona debido a que parte de un principio aceptado, el principio de proximidad de referencia, y lo explota para conseguir una estrategia de la gestión de la memoria que minimice los fallos de página. Desgraciadamente, persisten varios problemas en la estrategia del conjunto trabajo:

1. El pasado no siempre predice el futuro. Tanto el tamaño como la pertenencia al conjunto de trabajo cambian a lo largo del tiempo (por ejemplo, véase la Figura 8.20).
2. Una medición verdadera del conjunto de trabajo para cada proceso no es practicable. Sería necesario, por cada proceso, asignar un sello de tiempo a cada referencia a una página que asigne el tiempo virtual del proceso y que mantenga una lista ordenada por tiempo de las páginas de cada uno de ellos.
3. El valor óptimo de es desconocido y en cualquier caso puede variar.

La forma de hacer esto es centrarse no en las referencias exactas a páginas y sí en la tasa de fallos de página. La tasa de fallos de páginas cae a medida que se incrementa el tamaño del conjunto residente de un proceso. En lugar de monitorizar el tamaño del conjunto trabajo de forma directa se pueden alcanzar resultados comparables monitorizando la tasa de fallos de página. Si la tasa de fallos de páginas de un proceso está por debajo de un determinado límite, el sistema de forma global se puede beneficiar de asignar un tamaño de conjunto residente menor para este proceso sin dañar al proceso en cuestión

Un algoritmo que sigue esta estrategia es el algoritmo de frecuencia de fallos de página]. El algoritmo necesita un bit de usado que se encuentre asociado a cada página de memoria. Este

bit se pondrá a 1 cuando se haya accedido a la página. Cuando se produce un fallo de página, el sistema operativo anotará el tiempo virtual desde el último fallo de página para dicho proceso; esto se puede realizar manteniendo un contador de las referencias a páginas. Se fija un umbral F. Si la diferencia de tiempo con el último fallo de página es menor que éste, entonces se añade una página al conjunto residente del proceso. En otro caso, se descartan todas los páginas con el bit de usado a 0, y se reduce el tamaño del conjunto residente de forma acorde. Mientras tanto, se pone a 0 el bit de usado del resto de las páginas.
El tiempo entre fallos de página es recíproco a la tasa de fallos de página.
Si una estrategia de este estilo se complementa con el buffering de páginas, se puede conseguir como resultado un rendimiento bastante bueno.
Sin embargo, existe un fallo grave en la estrategia adoptada por PFF, que hace que su comporta- miento no sea bueno durante los periodos transitorios cuando se produce un desplazamiento hacia una nueva región de referencia. Con PFF ninguna página sale del conjunto residente antes de que hayan pasado F unidades de tiempo virtual desde su última referencia. Durante estos periodos entre dos regiones de referencia, la rápida sucesión de fallos de página hace que el conjunto residente del pro- ceso crezca antes de que las páginas de la antigua región de referencia se expulsen; los súbitos picos en las solicitudes de memoria pueden producir desactivaciones y reactivaciones de procesos innecesarias, que se corresponden con cambios de proceso y sobrecargas de swapping no deseables.

SISTEMAS DE ARCHIVOS

Sistema de archivos une la idea nuestra de usuarios recpecto con lo que pasa en el SO, provee mecanismos de almacenamiento y acceso tanto de datos y de programas del SO como de los usuarios del sistema de computación. Es el aspecto mas visible de un SO, implementa una abstracción que conocemos como archivos, organiza los archivos en forma lógica(carpeta o directorio). Estas dos cosas son abstracciones, lo que en realidad tenemos son bloques de datos que van a corresponder a archivos. Permite compartir datos entre procesos, gente y maquinas. Protege os datos de acceso no autorizados.
El archivo es una entidad abstracta, lo mas próximo que percibe el usuario, a bajo nivel es solo una secuencia de bytes. se guarda de forma persistente, duradera.
Las operaciones sobre los archivos las logramos a través de llamadas al sistema. Cada una requiere la búsqueda de la entrada en el directorio asociado con el archivo..
Se tiene una tabla de archivos abiertos, tiene una entrada por cada uno de los archivos que los usuarios abren. Cuando se cierra el archivo se elimina de la tabla de archivos abiertos.
En los accesos secuenciales, la info se procesa en secuencia, se comienza en el bloque 1 del archivo y se accede de forma gradual.es de los mas simples. Los usan editores de C, los editores de texto,e tc.
Accesos directos, para que se puedan dar los archivos tienen que estar conformados por registros lógicos de tamaño fijo. Para poder acceder, como los registros son de tamaño fijo, se sabe cuanto va a ocupar un registro, el registro va estar en el tamaño de registro por la ubicación que queremos. Se trata como una secuencia numerada de bloques o registros.
La diferencia entre estos es que el secuencial tengo que barrer toda la info y en el directo hago saltos de una dirección a otra, esto permite acceder a grandes cantidades de info.
En el acceso indexada, mantienen una tabla de índices, el registro 1 en tal lado, en 2 en tal lado, etc. La diferencia con el directo es que el indexado es mas genérico que el directo, se usa cuando n tenemos los registros lógicos del mismo tamaño.
Para organizar los archivos surge en concepto de directorio, para los usuarios provee una forma estructurada de organizar los archivos y para el SA provee una interface de nombramiento adecuada que permite la implementación de una organización de archivos lógica, diferente de la ubicación física del archivo en el disco. Están almacenados en el disco, pero también se puede tener en la ram para acceder mas rápido a la info. La mayoría soporta directorios multinivel.

Directorio es un archivo con características especiales, en un directorio se tiene un listado de los archivos que están dentro de ese directorio. También se puede ver como una tabla con tantas entradas como archivos tiene ese directorio, cada entrada tiene un campo y el nombre del archivo.
Implementación del sistema de archivos
Implementando archivos: Desde el punto de vista del almacenamiento secundario, hay que tener en cuenta la asignación de espacios de almacenamiento secundario a archivos y manejo del espacio disponible para asignar. Cuando se crea un archivo se puede asignar con:
Pre asignación significa que el so debe conocer al momento de asignar un archivo el tamaño máximo que tendrá el archivo
Asignación dinámica es ir asignando bloques a medida que ele archivo crece en tamaño
Metodos de asignación de archivos:
Esta explicado en una pregunta arriba.

Administración Entrada y Salida

Dispositivos: almacenamiento, transmisión e interfaces con las personas. Son muy diferentes, se diferencian por:
• la unidad de trasferencia: los datos se pueden transferir por flujo de bytes o caracteres(mouse, teclado) o en bloques grandes(disco).
• Velocidad de trasferencia.
• Representación de los datos
• Uso que se le va a dar a un dispositivo influye sobre el software y las políticas de so y las utilidades de soporte.
• Complejidad de control.
• Condiciones de error
• Operaciones
Clasificación:

- Dependiendo del flujo de caracteres o de bloques
- Accesos secuenciales o aleatorios
- Síncronos o asíncronos
- Compatible o dedicado
- Velocidad de operación
- Lectura-escritura, escritura o lectura.

Hay dispositivos que no se ajustan a la clasificación 1, por ejemplo los relojes.
Los dispositivos de E/S tiene dos componentes, el mecánico y el electrónico. Se usan para proveer un diseño mas general y modular.
Teneos los puertos para que se comunique el sistema, el bus de comunicación que conecta la memoria principal con el procesador y los dispositivos de E/S.
El soft de E/S, esta dentro del kernel el objetivo es independencia del dispositivo, si el so fuera dependiente del dispositivo seria muy complejo el desarrollo del so y la posibilidad de evolucionar. Esta es la característica fundamental. El so nombra y accede a todos los dispositivo de manera uniforme, el manejo de errores también es general.
Surge el driver que encapsula los detalles y particularidades de diferentes dispositivos.