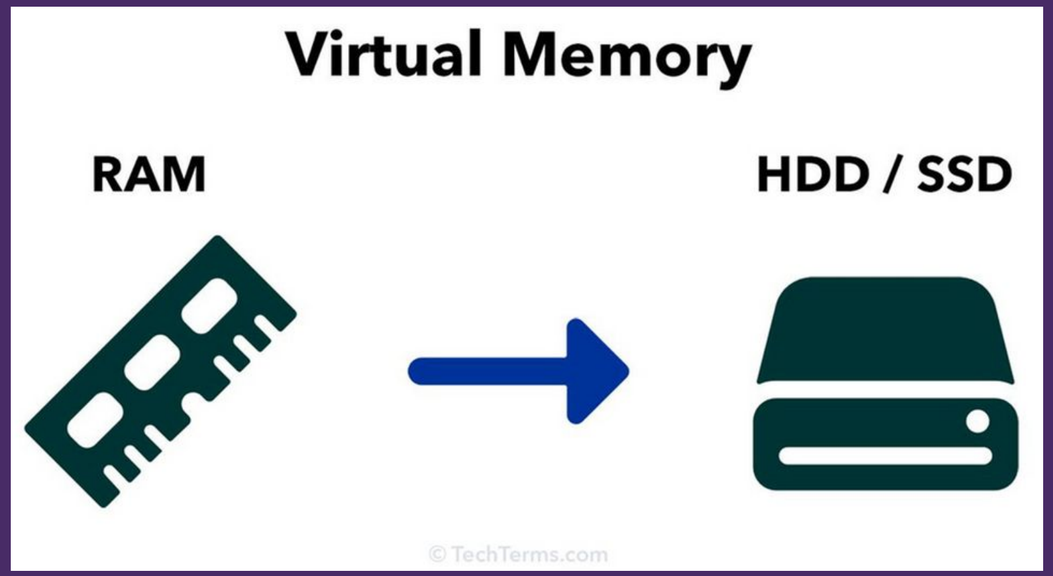
LABORATORIO SISTEMAS OPERATIVOS 2

MEMORIA VIRTUAL

MEMORIA VIRTUAL

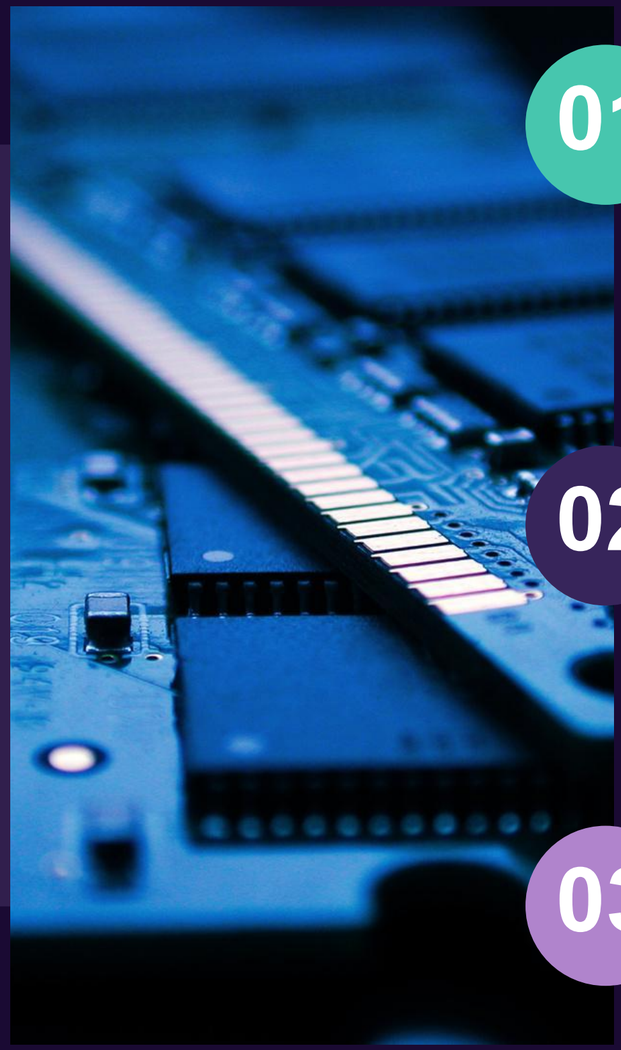


La memoria virtual es una técnica que permite la ejecución de procesos que no están completamente en memoria. Una ventaja importante de este esquema es que los programas pueden ser más grandes que la memoria física.

Además, la memoria virtual abstrae la memoria principal en una matriz de almacenamiento uniforme y extremadamente grande, separando la memoria lógica vista por el usuario de la memoria física. Esta técnica libera a los programadores de las preocupaciones sobre las limitaciones de almacenamiento de memoria.

El requisito de que las instrucciones deban estar en la memoria física para ser ejecutadas parece necesario y razonable; pero también es desafortunado, ya que limita el tamaño de un programa al tamaño de la memoria física.

De hecho, un analisis de programas reales nos muestra que, en muchos casos, no se necesita el programa completo. Por ejemplo:



01

Un programa ya no estaría limitado por la cantidad de memoria física disponible. Los usuarios podrían escribir programas para un espacio de direcciones virtuales extremadamente grande, simplificando la tarea de programación.

02

Debido a que cada programa de usuario podría ocupar menos memoria física, más programas podrían ejecutarse al mismo tiempo, con un aumento correspondiente en la utilización y el rendimiento de la CPU, pero sin aumento en el tiempo de respuesta.

03

Se necesitaría menos E/S para cargar o intercambiar cada programa de usuario en la memoria, por lo que cada programa de usuario se ejecutaría más rápido.

La capacidad de ejecutar un programa que esté sólo parcialmente en la memoria le otorgaría muchos beneficios

01

Un programa ya no estaría limitado por la cantidad de memoria física disponible. Los usuarios podrían escribir programas para un espacio de direcciones virtuales extremadamente grande, simplificando la tarea de programación.

02

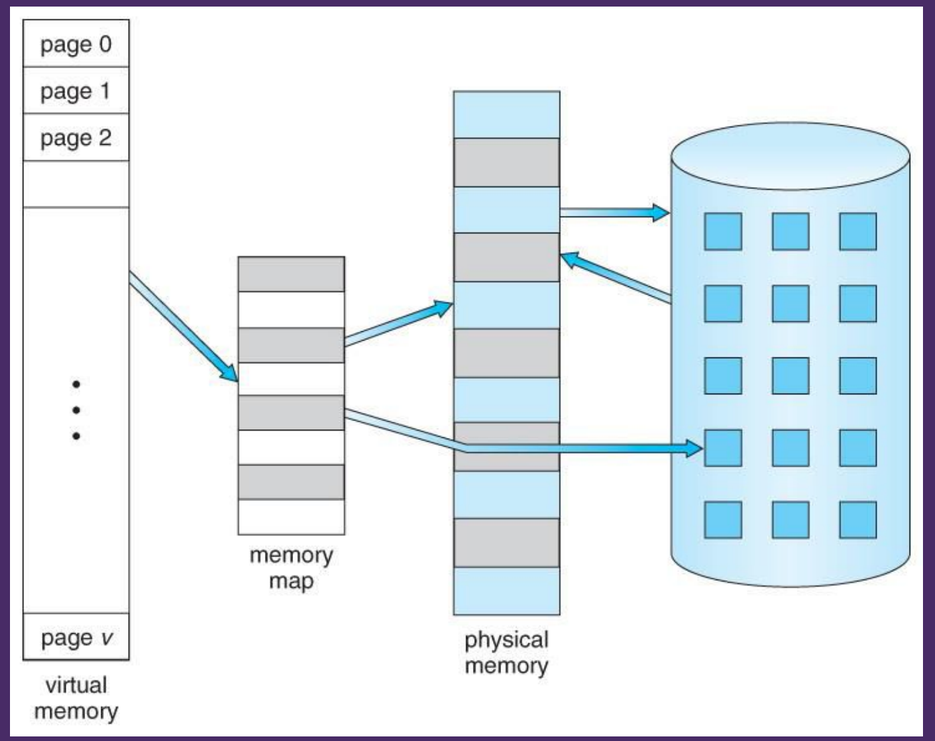
Debido a que cada programa de usuario podría ocupar menos memoria física, más programas podrían ejecutarse al mismo tiempo, con un aumento correspondiente en la utilización y el rendimiento de la CPU, pero sin aumento en el tiempo de respuesta.

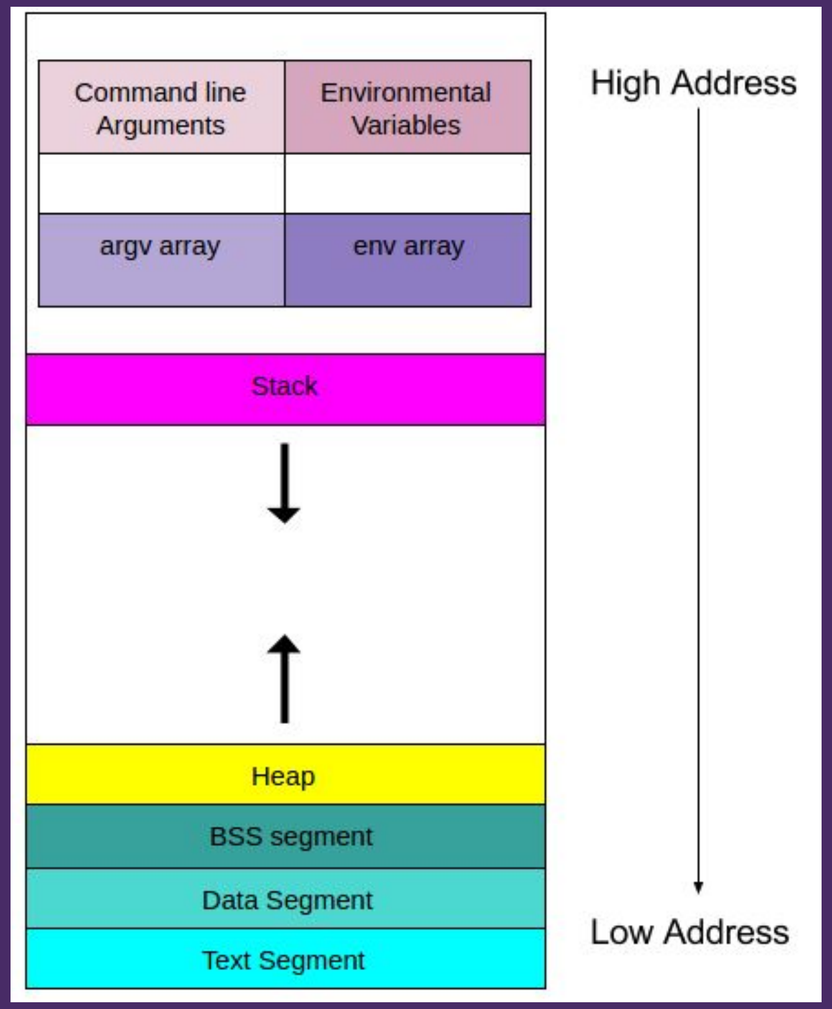
03

Se necesitaría menos E/S para cargar o intercambiar cada programa de usuario en la memoria, por lo que cada programa de usuario se ejecutaría más rápido.

La memoria virtual implica la separación de la memoria lógica, tal como la perciben los usuarios, de la memoria física. Esta separación permite proporcionar una memoria virtual extremadamente grande a los programadores cuando solo hay disponible una memoria física más pequeña.

La memoria virtual facilita mucho la tarea de programación, porque el programador ya no necesita preocuparse por la cantidad de memoria física disponible; en cambio, puede concentrarse en el problema que debe programarse.

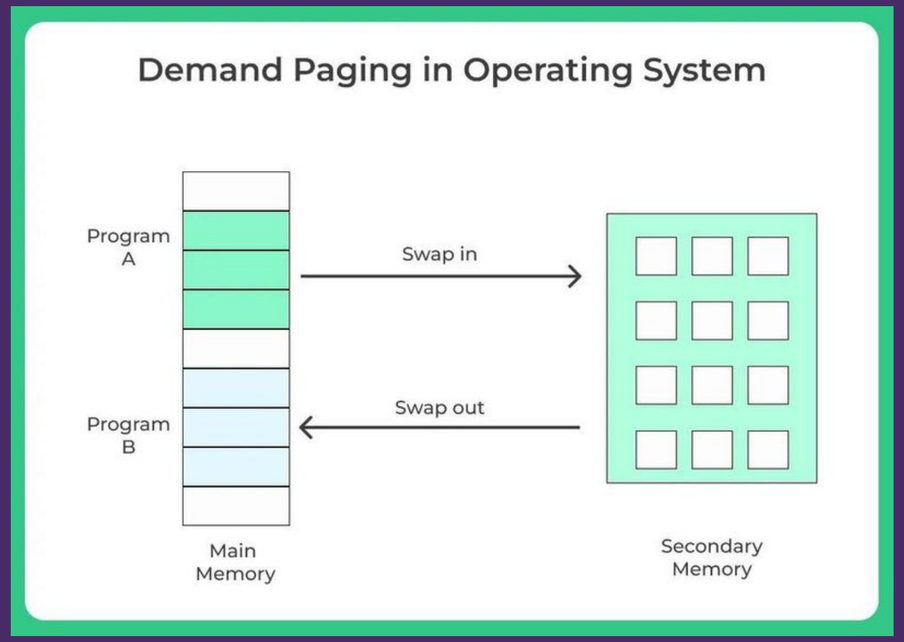




El espacio de direcciones virtuales de un proceso se refiere a la vista lógica (o virtual) de cómo se almacena un proceso en la memoria. Normalmente, esta visión es que un proceso comienza en una determinada dirección lógica (por ejemplo, la dirección 0) y existe en una memoria contigua.

La memoria física puede estar organizada en marcos de página y los marcos de página físicos asignados a un proceso pueden no ser contiguos. Depende de la unidad de administración de memoria (MMU) asignar páginas lógicas a marcos de páginas físicas en la memoria.

PAGINACIÓN POR DEMANDA

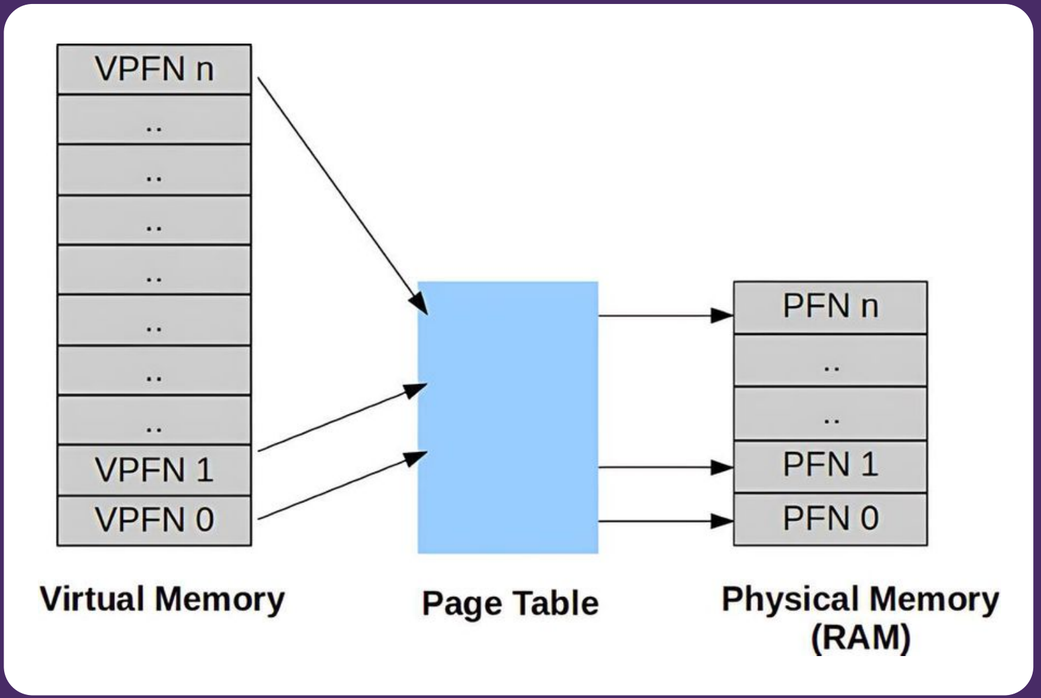


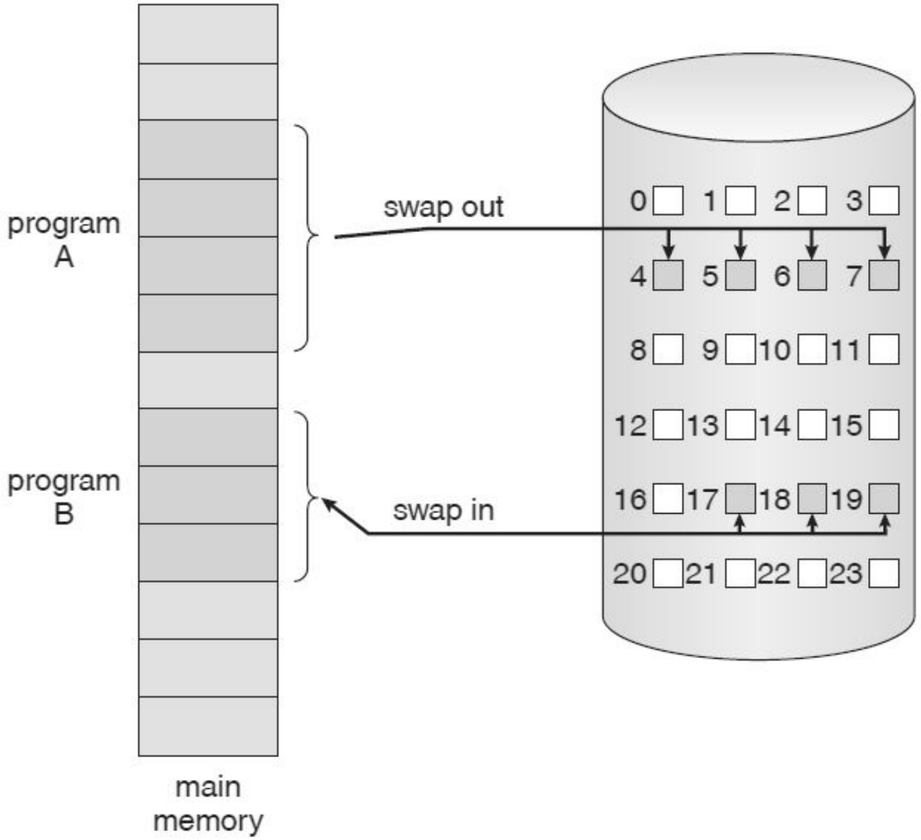
Considere un programa que comienza con una lista de opciones disponibles entre las que el usuario debe seleccionar. Cargar todo el programa en la memoria da como resultado la carga del código ejecutable para todas las opciones, independientemente de si el usuario finalmente selecciona una opción o no.

Una estrategia alternativa es cargar inicialmente las páginas sólo cuando sean necesarias. Esta técnica se conoce como paginación por demanda y se usa comúnmente en sistemas de memoria virtual.

Con la memoria virtual paginada bajo demanda, las páginas sólo se cargan cuando se solicitan durante la ejecución del programa; Por lo tanto, las páginas a las que nunca se accede nunca se cargan en la memoria física.

Un sistema de paginación bajo demanda es similar a un sistema de paginación con intercambio donde los procesos residen en la memoria secundaria (generalmente un disco). Cuando queremos ejecutar un proceso, lo intercambiamos en la memoria.





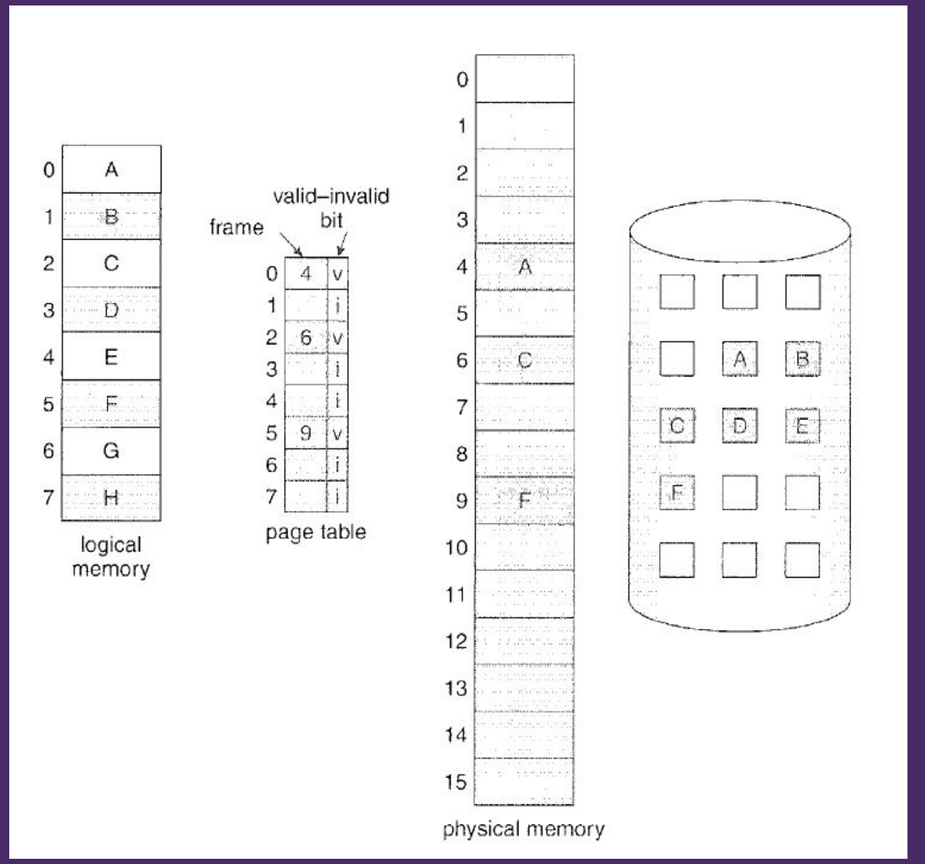
Cuando se va a intercambiar un proceso, el paginador “adivina” qué páginas se utilizarán antes de que se vuelva a intercambiar el proceso. En lugar de intercambiar un proceso completo, el paginador trae a memoria sólo las páginas necesarias.

Por lo tanto, evita leer en la memoria páginas que no se utilizarán de todos modos, disminuyendo el tiempo de intercambio y la cantidad de memoria física necesaria.

Con este esquema, necesitamos algún tipo de soporte de hardware para distinguir entre las páginas que están en la memoria y las páginas que están en el disco.

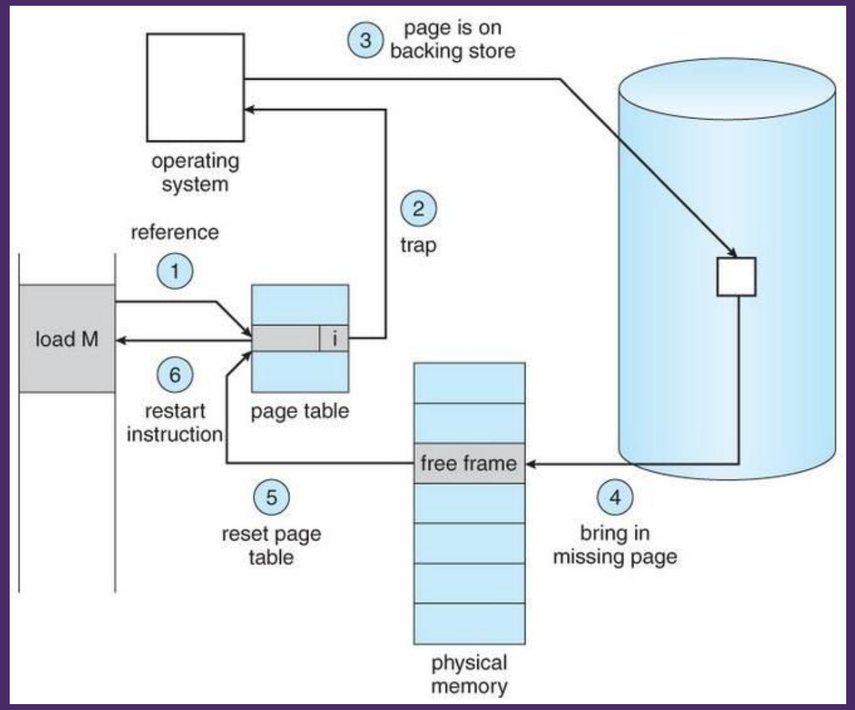
Para este fin se puede utilizar el esquema de bits válido-inválido. Esta vez, sin embargo, cuando este bit se establece en "válido", la página asociada es legal y está en la memoria. Si el bit se establece en "no válido", la página no es válida (es decir, no está en el espacio de direcciones lógicas del proceso) o es válida pero se encuentra actualmente en el disco.

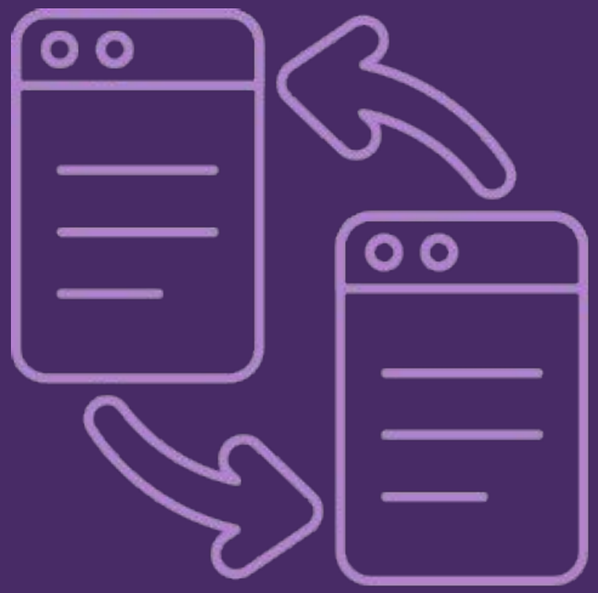
La entrada de la tabla de páginas para una página que se trae a la memoria se establece como habitual, pero la entrada de la tabla de páginas para una página que no está actualmente en la memoria simplemente se marca como no válida o contiene la dirección de la página en el disco.



El procedimiento para manejar un fallo de página es sencillo:

1. Verificamos una tabla interna (generalmente mantenida con el bloque de control de proceso) de este proceso para determinar si la referencia fue un acceso a memoria válido o no válido.
2. Si la referencia no fuera válida, damos por finalizado el proceso. Si era válido, pero aún no hemos ingresado esa página, ahora la ingresamos.
3. Encontramos un marco libre (tomando uno de la lista de marcos libres, por ejemplo).
4. Programamos una operación de disco para leer la página deseada en el marco recién asignado.
5. Cuando se completa la lectura del disco, modificamos la tabla interna mantenida con el proceso y la tabla de páginas para indicar que la página ahora está en la memoria.
6. Reiniciamos la instrucción que fue interrumpida por la trampa. El proceso ahora puede acceder a la página como si siempre hubiera estado en la memoria.



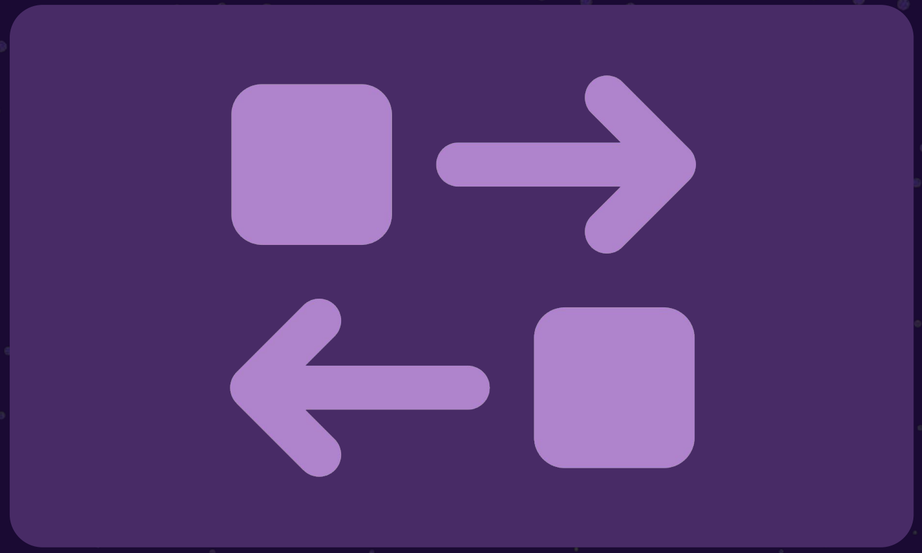


Pero, ¿qué sucede si la página deseada reside en el disco pero no hay marcos libres?

El sistema operativo tiene varias opciones en este punto. Podría finalizar el proceso del usuario. Sin embargo, la paginación bajo demanda es el intento del sistema operativo de mejorar la utilización y el rendimiento del sistema informático. Los usuarios no deben ser conscientes de que sus procesos se ejecutan en un sistema paginado; la paginación debe ser lógicamente transparente para el usuario. Entonces esta opción no es la mejor opción.

En este caso, la solución más común es el reemplazo de páginas.

REEMPLAZO DE PAGINAS



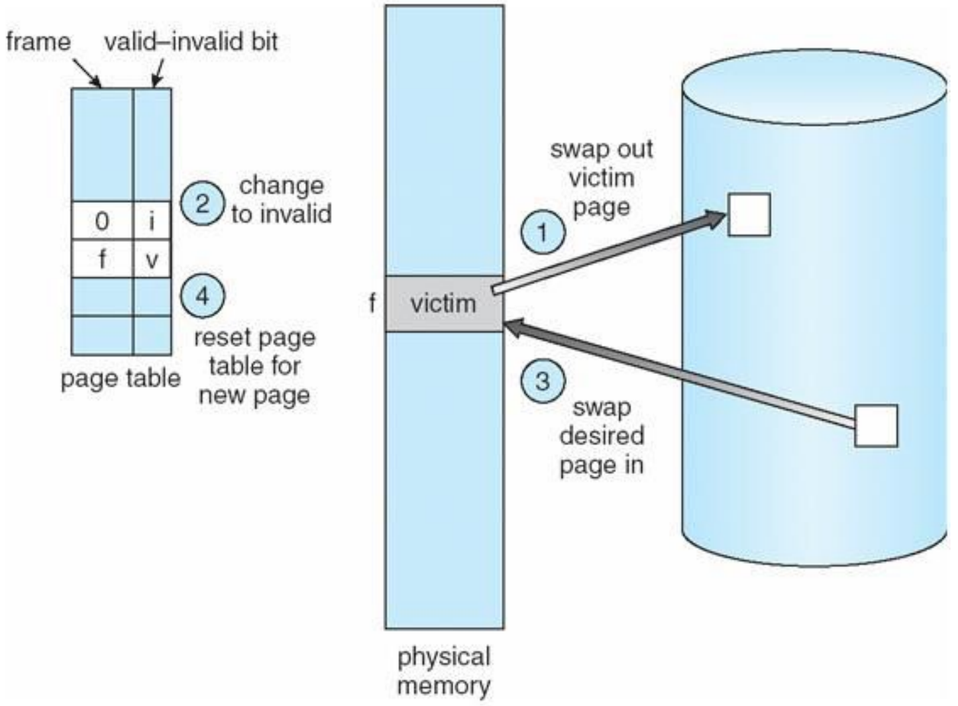
El reemplazo de página adopta el siguiente enfoque. Si no hay ningún marco libre, buscamos uno que no esté siendo utilizado actualmente y lo liberamos.

Podemos liberar un marco escribiendo su contenido en el espacio de intercambio y cambiando la tabla de páginas (y todas las demás tablas) para indicar que la página ya no está en la memoria. Ahora podemos usar el marco liberado para contener la página en la que falló el proceso.

Modificamos la rutina de servicio de fallo de página para incluir el reemplazo de página:

1. Buscar la ubicación de la página deseada en el disco.
2. Encpmtrar un marco libre:
    - Si hay un marco libre, se usa este.
    - Si no hay ningún marco libre, se utiliza un algoritmo de reemplazo de página por el marco de la víctima.

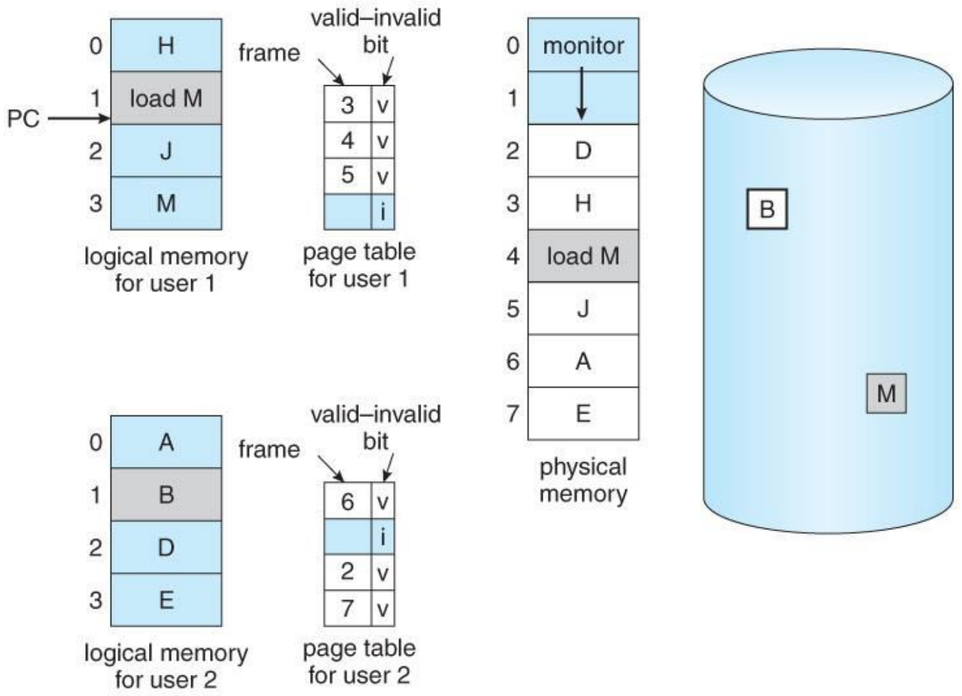
3. Se escribe la víctima en el disco; se cambian las tablas de páginas y marcos en consecuencia.
4. Se lee la página deseada en el marco recién liberado; se cambian las tablas de páginas y marcos.
5. Se reinicia el proceso de usuario.



El reemplazo de página es básico para paginación por demanda. Completa la separación entre memoria lógica y memoria física. Con este mecanismo se puede proporcionar una enorme memoria virtual a los programadores en una memoria física más pequeña.

Sin demanda, las direcciones de paginación/usuario se asignan a direcciones físicas, por lo que los dos conjuntos de direcciones pueden ser diferentes.

Todas las páginas de un proceso aún deben estar en la memoria física; sin embargo, con la paginación bajo demanda, el tamaño del espacio de direcciones lógicas ya no está limitado por la memoria física.



Debemos resolver dos problemas principales para implementar la paginación bajo demanda: debemos desarrollar un algoritmo de asignación de marcos y un algoritmo de reemplazo de páginas.

Si tenemos múltiples procesos en memoria, debemos decidir cuántos marcos deseamos asignar a cada proceso. Además, cuando sea necesario reemplazar la página, debemos seleccionar los marcos que se van a reemplazar. Diseñar algoritmos apropiados para resolver estos problemas es una tarea importante porque la E/S de disco es muy costosa.

Existen muchos algoritmos diferentes de reemplazo de páginas. Probablemente cada sistema operativo tenga su propio esquema de reemplazo. ¿Cómo seleccionamos un algoritmo de reemplazo particular? En general, queremos el que tenga la tasa de error de página más baja.

THRASHING

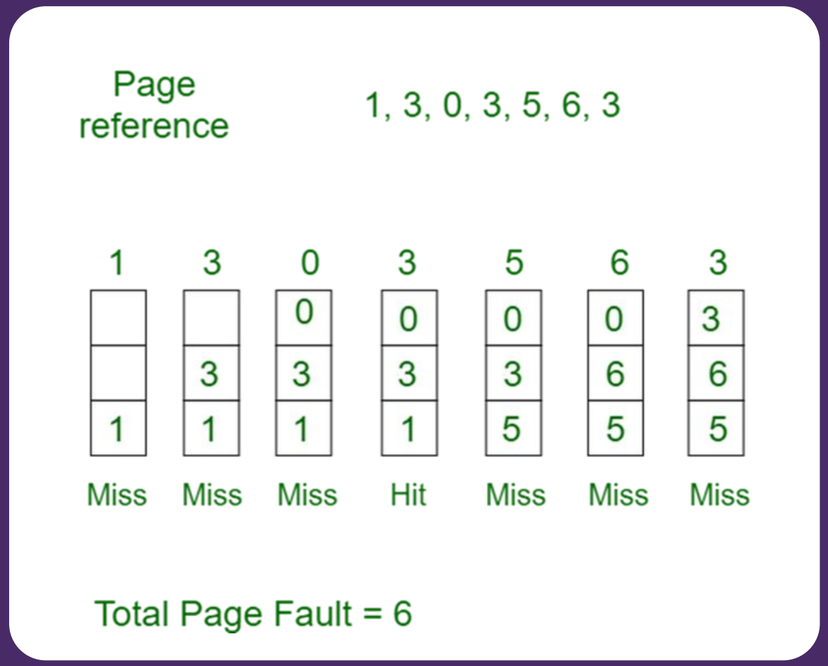
¿Qué pasa si el algoritmo que seleccionamos no funciona correctamente?

Pensemos en un proceso que no tiene la cantidad de marcos que necesita para admitir nuevas páginas, este rápidamente generará un fallo de página, punto donde deberá reemplazar alguna página activa.

Sin embargo, dado que todas sus páginas están en uso activo, se debe reemplazar una página que será necesaria de forma inmediata. En consecuencia, rápidamente falla una y otra vez, reemplazando páginas que el paginador deberá recuperar inmediatamente.

Esta alta actividad de paginación se llama thrasing, que ocurre si el sistema esta dedicando más tiempo a paginar que a ejecutar.

REEMPLAZO DE PÁGINA FIFO



El algoritmo de reemplazo de páginas más simple es el primero en entrar, primero en salir (FIFO). Un algoritmo de reemplazo FIFO asocia con cada página el momento en que esa página se trajo a la memoria.

Cuando es necesario reemplazar una página, se elige la página más antigua. Tenga en cuenta que no es estrictamente necesario registrar la hora en que se ingresa una página.

Podemos crear una cola FIFO para contener todas las páginas en la memoria.
Reemplazamos la página al principio de la cola.
Cuando una página llega a la memoria, la insertamos al final de la cola.

ANOMALÍA DE BELADY

	0	1	2	3	0	1	4	0	1	2	3	4
M1	0	0	0	3	3	3	4	4	4	4	4	4
M2		1	1	1	0	0	0	0	0	2	2	2
M3			2	2	2	1	1	1	1	1	3	3
	F	F	F	F	F	F	F			F	F	

Total de fallos de página: 9

	0	1	2	3	0	1	4	0	1	2	3	4
M1	0	0	0	0	0	0	4	4	4	4	3	3
M2		1	1	1	1	1	1	0	0	0	0	4
M3			2	2	2	2	2	2	1	1	1	1
M4				3	3	3	3	3	3	2	2	2
	F	F	F	F			F	F	F	F	F	F

Total de fallos de página: 10

Es de esperar que dar más memoria a un proceso mejore su rendimiento. En algunas investigaciones iniciales, los investigadores notaron que esta suposición no siempre era cierta.

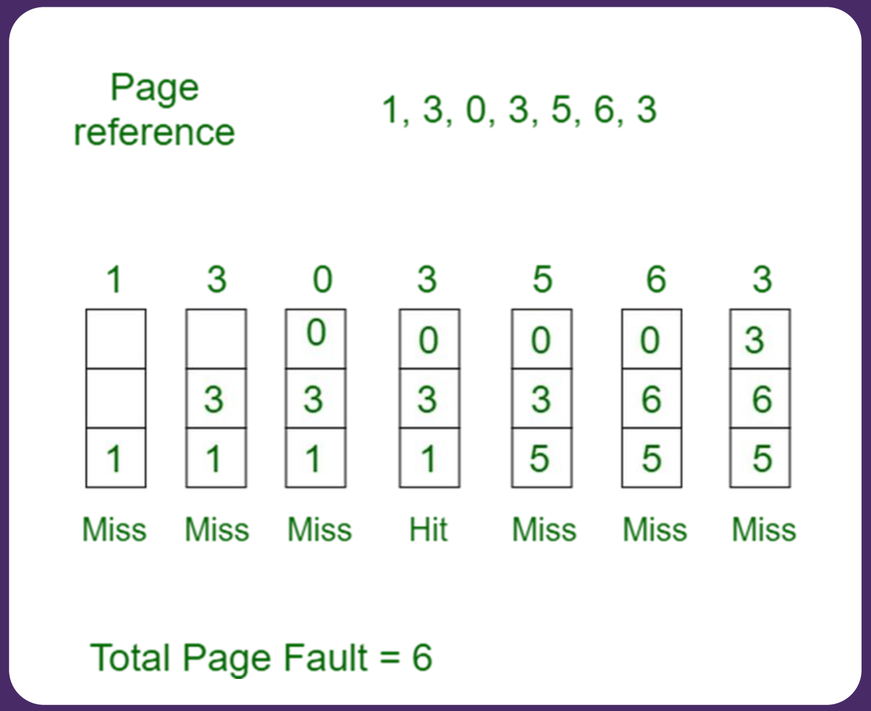
La anomalía de Belady es un efecto donde es posible tener más fallos de página al aumentar el número de marcos en la memoria física utilizando el método FIFO como algoritmo de reemplazo.

REEMPLAZO DE PÁGINA ÓPTIMO

Uno de los resultados del descubrimiento de la anomalía de Belady fue la búsqueda de un algoritmo óptimo de sustitución de páginas. Un algoritmo óptimo de reemplazo de páginas tiene la tasa de fallas de página más baja de todos los algoritmos y nunca sufrirá la anomalía de Belady. Dicho algoritmo consiste en reemplazar la página que no se utilizará durante un período de tiempo más largo.

Obviamente este algoritmo no puede existir, ya que es imposible conocer los requisitos futuros de un proceso, por lo que se suele utilizar para para comprobar el rendimiento de los siguientes algoritmos.

REEMPLAZO DE LA PÁGINA MENOS USADA RECIENTEMENTE



El reemplazo de la menos usada recientemente (Least Recently Used o LRU) asocia con cada página la hora del último uso de esa página.

Cuando se debe reemplazar una página, LRU elige la página que no se ha utilizado durante más tiempo.

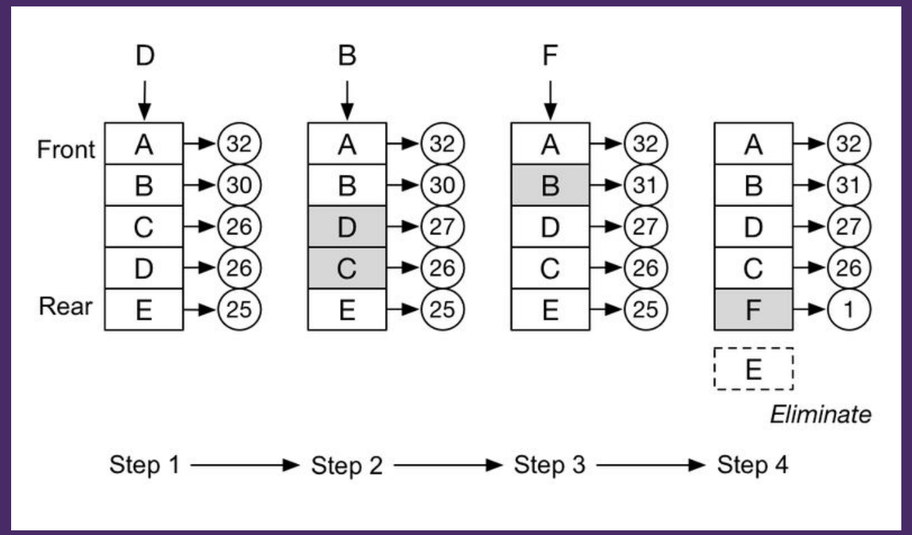
Podemos pensar en esta estrategia como el algoritmo óptimo de reemplazo de páginas mirando hacia atrás en el tiempo, en lugar de hacia adelante.

La política LRU se utiliza a menudo como algoritmo de reemplazo de páginas y se considera buena. El principal problema es cómo implementarlo sin requerir asistencia de hardware sustancial.

En el caso más sencillo, asociaremos con cada entrada de la tabla de páginas un campo de tiempo y agregaremos a la CPU un contador lógico. El reloj se incrementa para cada referencia de memoria. Cada vez que una referencia a una página se realiza, el contenido del registro de reloj se copia en el campo de tiempo.

Otro enfoque para implementar el reemplazo de LRU es mantener una pila de números de página. Cada vez que se hace referencia a una página, se elimina de la pila y se coloca en la parte superior. De esta manera, la página utilizada más recientemente siempre está en la parte superior de la pila

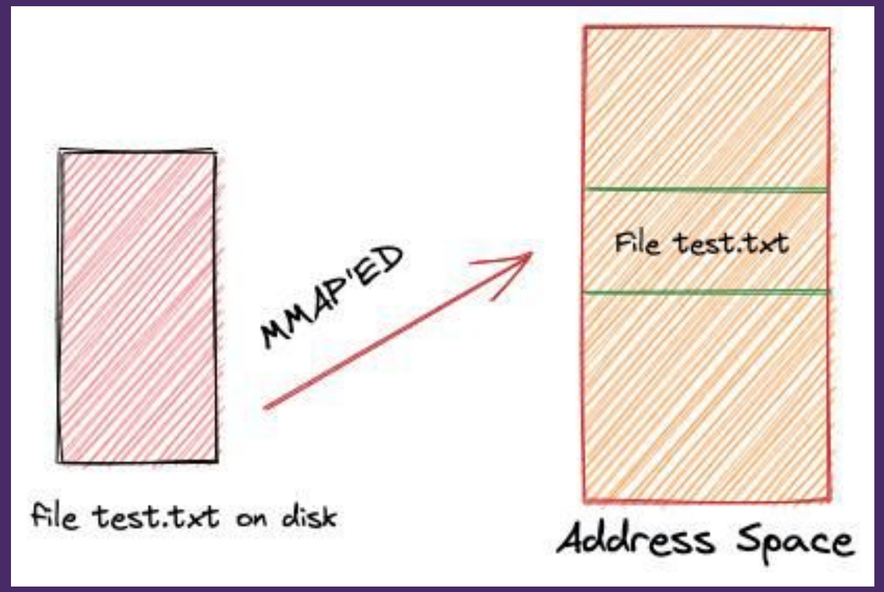
REEMPLAZO DE LA PÁGINA UTILIZADA CON MENOS FRECUENCIA (LFU)



Aquí podemos llevar un contador del número de referencias que se han hecho a cada página, luego se reemplazará la página con el menor recuento. El motivo de esta selección es que una página utilizada activamente debe tener un gran número de referencias.

Sin embargo, surge un problema cuando una página se utiliza mucho durante la fase inicial de un proceso pero luego nunca se vuelve a utilizar. Como se usó mucho, tiene un gran número y permanece en la mención aunque ya no sea necesario. Una solución es desplazar los recuentos 1 bit a la derecha a intervalos regulares, formando un recuento de uso promedio que decae exponencialmente.

ARCHIVOS MAPEADOS EN MEMORIA



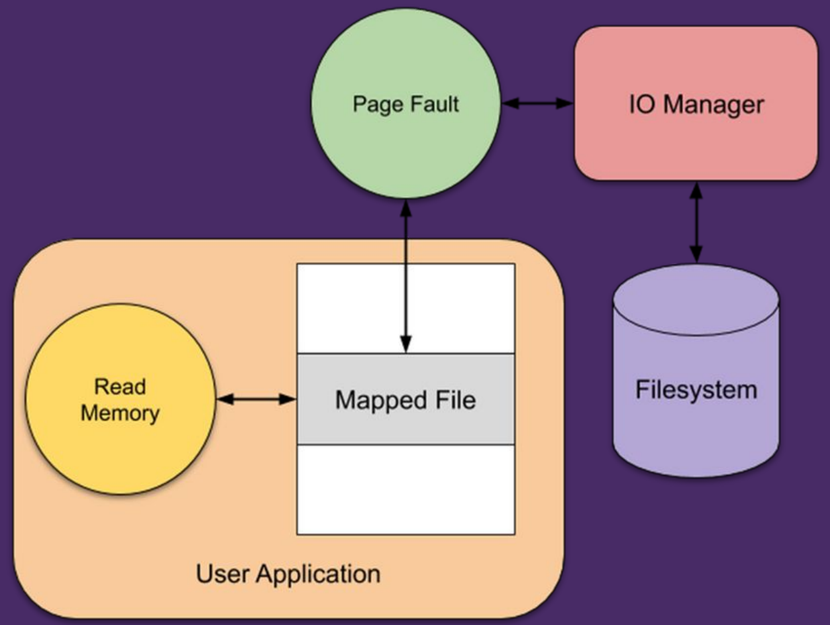
Otro uso de la memoria virtual es el mapeo de archivos en memoria.

Considere una lectura secuencial de un archivo en el disco utilizando las llamadas estándar al sistema open(), read() y write(). Cada acceso a archivos requiere una llamada al sistema y acceso al disco.

Alternativamente, podemos utilizar las técnicas de memoria virtual analizadas hasta ahora para tratar la E/S de archivos como accesos rutinarios a la memoria. Este enfoque, conocido como mapeo de un archivo en memoria, permite que una parte del espacio de direcciones virtuales se asocie lógicamente con el archivo.

Mapear un archivo en memoria se logra asignando un bloque de disco a una página (o páginas) en la memoria. El acceso inicial al archivo se realiza mediante paginación de demanda ordinaria, lo que genera un error de página. Sin embargo, una porción del archivo del tamaño de una página se lee desde el sistema de archivos a una página física (algunos sistemas pueden optar por leer más de una porción de memoria del tamaño de una página a la vez).

Las lecturas y escrituras posteriores del archivo se manejan como accesos rutinarios a la memoria, lo que simplifica el acceso y el uso del archivo al permitir que el sistema manipule archivos a través de la memoria en lugar de incurrir en la sobrecarga de usar las llamadas al sistema read() y write().



El esquema general para crear una región de memoria compartida utilizando archivos asignados en memoria implica primero crear una asignación de archivos para el archivo que se va a asignar y luego establecer una vista del archivo asignado en el espacio de direcciones virtuales de un proceso.

Luego, un segundo proceso puede abrir y crear una vista del archivo mapeado en su espacio de direcciones virtuales. El archivo mapeado representa el objeto de memoria compartida que permitirá que se produzca la comunicación entre los procesos.

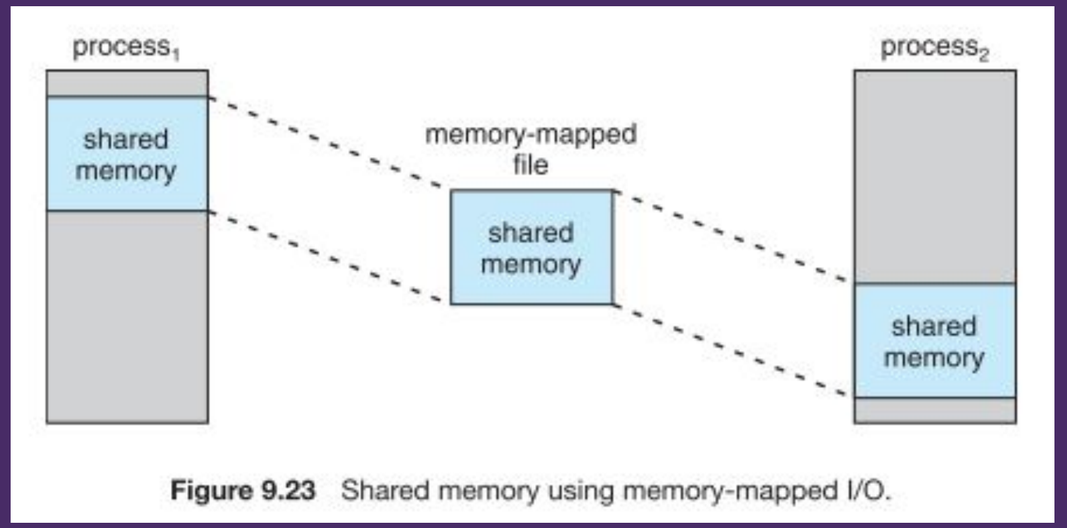


Figure 9.23 Shared memory using memory-mapped I/O.

¡GRACIAS POR LA ATENCIÓN!

¿Dudas?