UNIVERSIDAD NACIONAL DE UCAYALI

FACULTAD DE INGENIERÍA DE SISTEMAS E

INGENIERÍA CIVIL

ESCUELA PROFESIONAL DE INGENIERÍA DE

SISTEMAS

MEMORIA VIRTUAL

CURSO

Sistemas Operativos

DOCENTE

Mg. Ing. Leo Martín Chumbe Rodriguez.

ALUMNOS

Ascurra Suarez Sergio David.

López Montalván Héctor Andrey.

Marín Rengifo Juan Nicanor.

Suarez Maciel, Susana Isabel.

GRUPO

VII-B

Pucallpa-Perú

2022

Tabla de contenido

MEMORIA VIRTUAL

1. CONCEPTOS BASICOS

1.1. MEMORIA VIRTUAL

La memoria virtual es una técnica que se encarga de gestionar la memoria, esto implica que se utiliza un espacio del disco duro (o disco sólido) para actúe como memoria RAM.

El concepto surgió como un método de ahorro, haciendo posible que equipos que no tuvieran tanta memoria RAM, fueran un poco más rápidos. Con el paso de los años, el concepto quedó implantado en los sistemas operativos modernos, siendo casi que una necesidad.

1.1.1 CARACTERÍSTICAS DE LA MEMORIA VIRTUAL

- Actualmente se encuentra en casi todos los sistemas operativos.
- Al tratarse de un tipo de memoria no física, es más accesible para la CPU.
- Si no se utilizan fragmentos de esta, no se cargan ni se descargan, quedando simplemente allí.
- Normalmente se utiliza cuando la memoria RAM se encuentra llena.
- Su uso y creación fue hecha para optimización de computadoras con baja memoria, además de ser un complemento importante de la memoria RAM.

1.1.2 IMPORTANCIA DE LA MEMORIA VIRTUAL

La importancia de la memoria virtual radica en que, si el ordenador no tiene memoria suficiente, no se podrán ejecutar programas o los que se estén ejecutando tendrán problemas e irán más lento.

Hay que tener presente que una memoria virtual nunca será mejor que una ampliación de RAM, pero sirve como una alternativa para poder ejecutar múltiples programas sin la posibilidad de que la RAM se sobrecargue.

1.1.3 VENTAJAS DE LA MEMORIA VIRTUAL

- Hace que el uso de la memoria principal sea más óptimo.
- Se encarga de procesos pequeños, para así liberar trabajo menor a la memoria RAM.

- Permite al computador hacer uso de más memoria RAM que la tiene en realidad.

1.1.4 DESVENTAJAS DE LA MEMORIA VIRTUAL

- En primer lugar, se debe decir que, aunque el sistema tome una parte del almacenamiento como una porción de memoria, su rendimiento no será igual.
- La RAM tiene un uso particular y específico, algo que la virtual imita en una menor escala.
- En definitiva, no puede ser comparada con la capacidad de un slot que tiene una función específica y única.

1.1.5 LA MEMORIA VIRTUAL EN TÉRMINOS SENCILLOS

Vamos a comenzar con una aplicación hipotética. El código de máquina que conforma esta aplicación tiene un tamaño de 10000 bytes. También requiere otros 5000 bytes para el almacenamiento de datos y para la memoria intermedia de E/S. Esto significa que, para ejecutar la aplicación, deben haber más de 15000 bytes de RAM disponible; un byte menos y la aplicación no será capaz de ejecutarse.

Este requerimiento de 15000 bytes se conoce como el *espacio de direcciones* de la aplicación. Es el número de direcciones únicas necesarias para almacenar la aplicación y sus datos. En las primeras computadoras, la cantidad de RAM disponible tenía que ser mayor que el espacio de direcciones de la aplicación más grande a ejecutar; de lo contrario, la aplicación fallaría con un error de "memoria insuficiente".

Un enfoque posterior conocido como *solapamiento* intentó aliviar el problema permitiendo a los programadores dictar cuales partes de sus aplicaciones necesitaban estar residentes en memoria en cualquier momento dado. De esta forma, el código requerido solamente para propósitos de inicialización podía ser sobreescrito con código a utilizar posteriormente. Mientras que el solapamiento facilitó las limitaciones de memoria, era un proceso muy complejo y susceptible a errores. El solapamiento también fallaba en

solucionar el problema de las limitaciones de memoria globales al sistema en tiempo de ejecución. En otras palabras, un programa con solapamiento requería menos memoria para ejecutarse que un programa sin solapamiento, pero si el sistema no tiene suficiente memoria para el programa solapado, el resultado final es el mismo — un error de falla de memoria.

Con la memoria virtual el concepto del espacio de direcciones de las aplicaciones toma un significado diferente. En vez de concentrarse en cuanta memoria necesita una aplicación para ejecutarse, un sistema operativo con memoria virtual continuamente trata de encontrar una respuesta a la pregunta "¿qué tan poca memoria necesita la aplicación para ejecutarse?".

Aunque inicialmente pareciera que nuestra aplicación hipotética requiere de un total de 15000 bytes para ejecutarse, recuerde que el acceso a memoria tiende a ser secuencial y localizado. Debido a esto, la cantidad de memoria requerida para ejecutar la aplicación en un momento dado es menos que 15000 bytes — usualmente mucho menos. Considere los tipos de accesos de memoria requeridos para ejecutar una instrucción de máquina sencilla:

- La instrucción es leída desde la memoria.
- Se leen desde memoria los datos requeridos por la instrucción.
- Después de completar la instrucción, los resultados de la instrucción son escritos nuevamente en memoria.

El número real de bytes necesarios para cada acceso de memoria varían de acuerdo a la arquitectura del CPU, la instrucción misma y el tipo de dato. Sin embargo, aún si una instrucción requiere de 100 bytes de memoria por cada tipo de acceso de memoria, los 300 bytes requeridos son mucho menos que el espacio de direcciones total de la aplicación de 15000 bytes. Si hubiese una forma de hacer un seguimiento de los requerimientos de memoria de la aplicación a medida que esta se ejecuta, sería posible mantener la aplicación ejecutándose usando menos memoria que lo que indicaría su espacio de direcciones.

Pero esto genera una pregunta:

Si solamente una parte de la aplicación está en memoria en un momento dado, ¿dónde está el resto?

La respuesta corta a esta pregunta es que el resto de la aplicación se mantiene en disco. En otras palabras, el disco actúa como un *almacenamiento de respaldo* para la RAM; un medio más lento y también más grande que actúa como un "respaldo" para un almacenamiento más rápido y más pequeño. Esto puede parecer al principio como un gran problema de rendimiento en su creación — después de todo, las unidades de disco son mucho más lentas que la RAM.

2. TÉCNICAS: PAGINACIÓN Y SEGMENTACIÓN EN MEMORIA VIRTUAL

2.1. PAGINACIÓN EN MEMORIA VIRTUAL

Es una técnica de manejo de memoria, en la cual el espacio de memoria se divide en secciones físicas de igual tamaño, denominadas marcos de página. Los programas se dividen en unidades lógicas denominadas páginas, que tienen el mismo tamaño que los marcos de páginas.

Las páginas sirven como unidad de almacenamiento de información y transferencia a la memoria principal y la memoria secundaria. Las páginas de un programa necesitan estar contiguamente en: memoria.

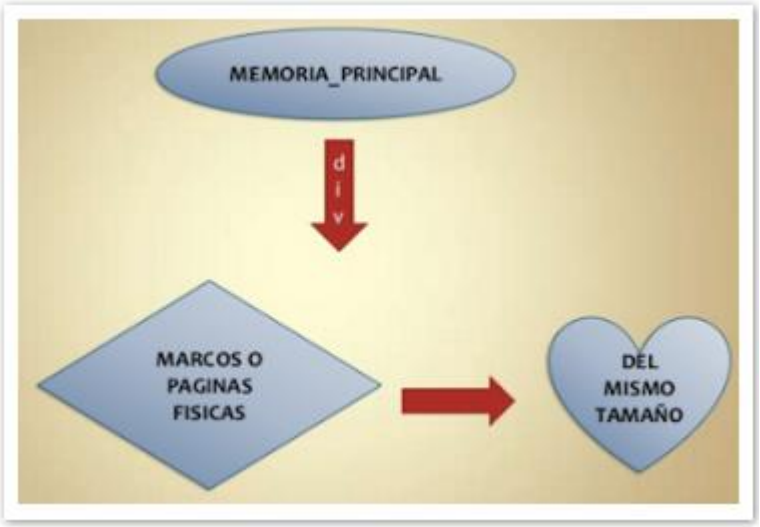


figura 1: Paginación

Existen 2 funciones de la paginación:

1. Llevar a cabo la transformación de una dirección virtual a física.
2. Transferir, cuando haga falta, páginas de la memoria secundaria a la principal, y de la memoria principal a la secundaria cuando ya no sean necesarias.

2.1.1 CARACTERÍSTICAS DE LA PAGINACIÓN

- El espacio de direcciones lógico de un proceso puede ser no contiguo.
- Se divide la memoria física en bloques de tamaño fijo llamados marcos (frames).
- Se divide la memoria en bloques de tamaño llamados páginas.
- Se mantiene información en los marcos libres.
- Para correr un programa de n páginas de tamaño, se necesitan encontrar a n marcos y cargar el programa.
- Se establece una tabla de páginas para trasladar las direcciones lógicas a físicas.
- Se produce fragmentación interna.

2.1.2 VENTAJAS DE LA PAGINACIÓN

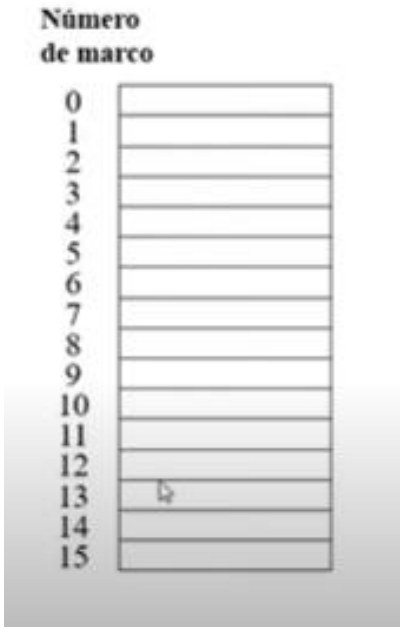
- Es posible comenzar a ejecutar un programa, cargando solo una parte del mismo en memoria, y el resto se cargará bajo la solicitud.
- No es necesario que las páginas estén contiguas en memoria.
- Fácil control de todas las páginas, ya que tienen el mismo tamaño.
- Se elimina el problema de fragmentación externa.
- Se obtiene una alta velocidad de acceso a memoria gracias a la TDP.

3.1.3 DESVENTAJAS DE LA PAGINACIÓN

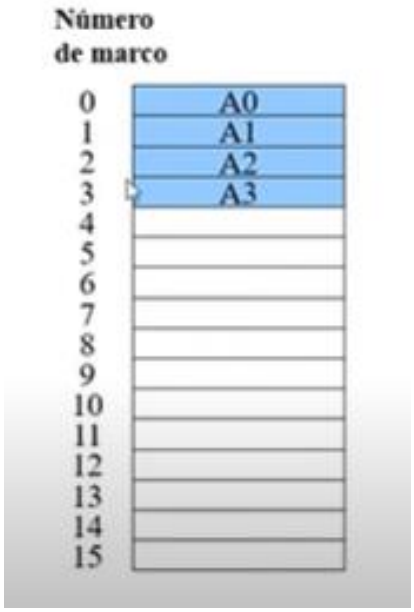
- Problema importante "Superfluity".
- El costo del Hardware y el Software se incrementa.
- Consumen más recursos de memoria.
- Aparece el problema de Fragmentación Interna.

3.1.4 EJEMPLO DE PAGINACIÓN

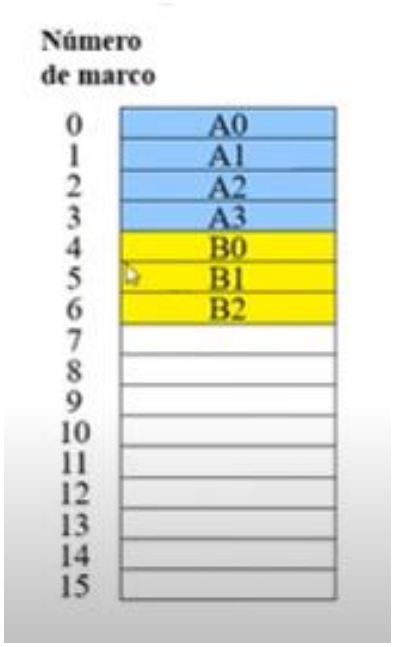
a. Se tienen 16 marcos



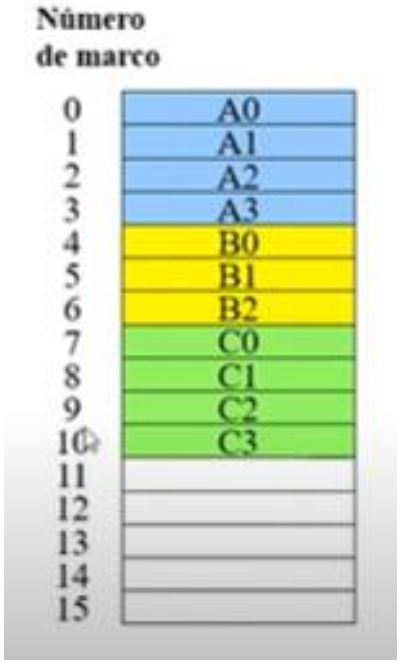
b. El proceso A necesita 4 paginas



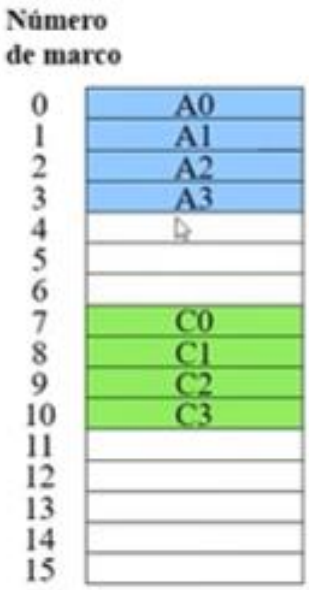
c. El proceso B necesita 3 paginas



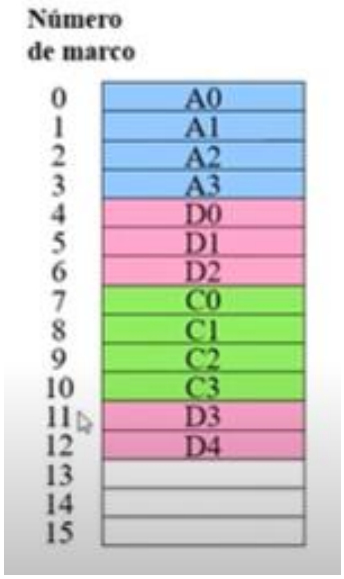
d. El proceso C necesita 4 paginas



e. El proceso B sale de la memoria



f. El proceso D necesita 5 páginas y puede ser ubicado así



g. Las tablas de páginas se reparten así

2.2. SEGMENTACIÓN EN MEMORIA VIRTUAL

La segmentación de memoria es un esquema de manejo de memoria mediante el cual la estructura del programa refleja su división lógica, llevándose a cabo una agrupación lógica de la información en bloques de tamaño variable denominados segmentos, es decir que los segmentos pueden ser de distintos tamaños, incluso de forma dinámica.

La segmentación permite al programador contemplar la memoria como si tuviera varios espacios de direcciones o segmentos. Las referencias a la memoria constan de una dirección de la forma (número de segmento, desplazamiento).

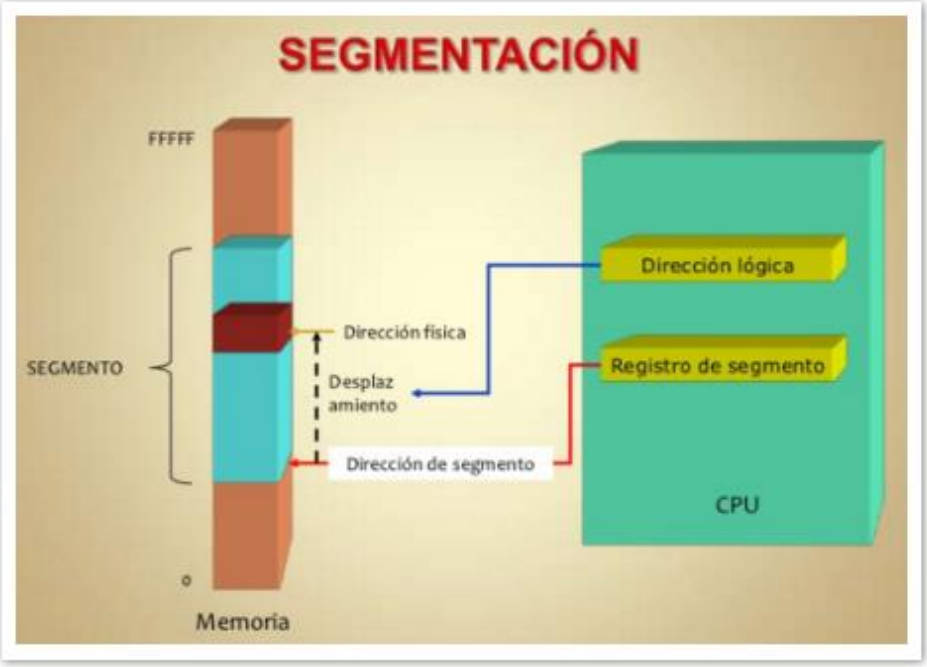


figura 2: Segmentación

2.2.1 VENTAJAS DE LA SEGMENTACIÓN

- El programador conoce las unidades lógicas de su programa.
- Es posible compilar módulos separados como segmentos.
- Facilidad de modificación de los módulos.
- El cambio dentro de un módulo no afecta al resto de los módulos.

2.2.2 DESVENTAJAS DE LA SEGMENTACIÓN

- Se complica el manejo de memoria virtual.
- El costo del Hardware y el Software se incrementa.

- Mayor consumo de recursos: memoria, tiempo de CPU, etc.
- Aparece el problema de Fragmentación Externa.

3. TRADUCCIÓN DE DIRECCIONES

La memoria "física" de un ordenador es una secuencia de bytes, que empieza en el 0 y termina en....por ejemplo 64 megas. Es decir, un "array" de bytes.

Cada byte tiene una dirección: el número que ocupa posicionalmente en la memoria. Esta dirección es conocida como "dirección física". (quiero "olvidarme, que además en los procesadores 386 y superiores de Intel, existe la llamada "segmentación", pero en principio vamos a odiar esto..... a propósito).

Las direcciones que va a utilizar un programa (en modo protegido), no son esas. Siempre son traducidas, o mapeadas, en direcciones de memoria física por un mecanismo de "traducción de direcciones".

Este mecanismo, nos introduce en el concepto de "dirección virtual". Se llama de esta manera porque no corresponde directamente a posiciones de memoria "física", sino que a través de una "función de mapeado" equivale a una dirección de memoria física.

Es decir, este mecanismo, al "apuntar" a una dirección de memoria, lo que hace es buscar, por ejemplo, en unas tablas internas, la correspondencia entre esta dirección y la dirección real "física" del dato.

La traducción de dirección "virtual" a "física" también proporciona protección de memoria, ya que podría disponerse, por ejemplo, que ciertas direcciones físicas de memoria no se mapeen desde ninguna dirección virtual.

Además de esta protección, en la función de traducción de direcciones, se pueden identificar ciertas direcciones virtuales como "no validas". Esto amplia el mecanismo de protección. Para no tener que generar una dirección física cuando se presenta una dirección virtual no valida, el mecanismo de traducción de direcciones informa de una "excepción", de forma que el software del sistema operativo, puede tomar la acción que considere oportuna.

El espacio de direcciones virtuales de un proceso es el conjunto de direcciones de memoria virtual que puede usar. El espacio de direcciones de cada proceso es privado y no se puede acceder a él por otros procesos a menos que se comparta.

Una dirección virtual no representa la ubicación física real de un objeto en memoria; en su lugar, el sistema mantiene una tabla de páginas para cada proceso, que es una estructura de datos interna que se usa para traducir direcciones virtuales en sus direcciones físicas correspondientes. Cada vez que un subproceso hace referencia a una dirección, el sistema traduce la dirección virtual a una dirección física.

El espacio de direcciones virtuales de Windows de 32 bits tiene un tamaño de 4 gigabytes (GB) y se divide en dos particiones: una para su uso por el proceso y la otra reservada para su uso por el sistema.

3.1 EJEMPLO DE TRADUCCIÓN DE DIRECCIONES

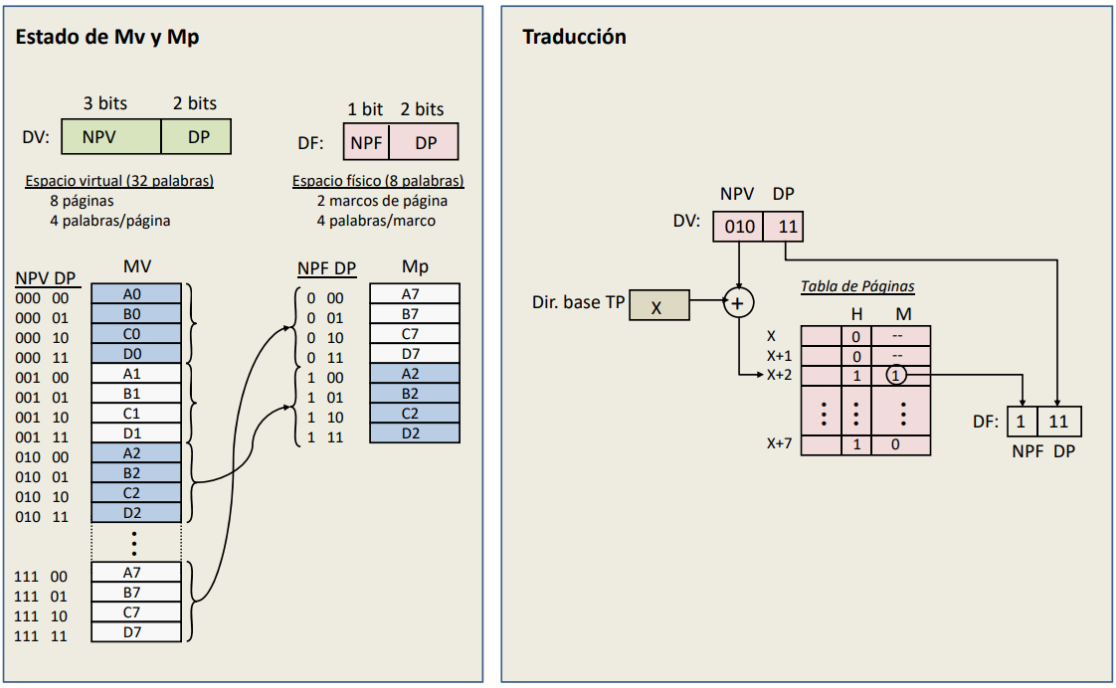


figura 3: Ejemplo de Traducción de direcciones.

4. ALGORITMOS O ESTRATEGIAS DE REEMPLAZO DE PAGINACIÓN

¿Qué sucede si se desea acceder a una página y esta no está en memoria?

Lo que sucede es lo que conocemos como fallo de página. Un fallo de página es cuando algún proceso que está en ejecución intenta acceder a datos o código que está en su espacio de direcciones, pero que no está actualmente ubicado en la

RAM del sistema. El sistema operativo debe manejar los fallos de página haciendo residentes en memoria los datos accedidos, permitiendo de esta manera que el programa continúe la operación como si el fallo de página nunca hubiera ocurrido.

Ahora bien, ¿Qué ocurre en un fallo de página?

- Se intenta acceder a la página.
- Si la página está en memoria ir al paso 8.
- Se genera un "trap" para el SO.
- Encontrar un marco libre para colocar la página que se leerá del disco.

Si la memoria está llena, el sistema operativo tiene que ejecutar un **algoritmo de reemplazo** depáginas para liberar un marco.

- Leer la página de disco y cargarla en marco de página en memoria real.
- Actualizar la tabla de páginas.
- Finalizar "trap".
- Acceder a la página.

Se puede decir que un fallo de página dispara una política de reemplazo. El objetivo de los algoritmos de reemplazo es minimizar la tasa de fallos de página.

Las estrategias de reemplazo se pueden clasificar en dos categorías: reemplazo global y reemplazo local. Con una estrategia de reemplazo global, se puede seleccionar, para satisfacer el fallo de página de un proceso, un marco que actualmente tenga asociada una página de otro proceso. Esto es, un proceso puede quitarle un marco de página a otro. La estrategia de reemplazo local requiere que, para servir el fallo de página de un proceso, solo pueden usarse marcos de páginas libres o marcos ya asociados al proceso.

A continuación, se describirán los algoritmos de reemplazo más típicos. Todos estos algoritmos pueden utilizarse tanto para estrategias globales como locales. Cuando se aplica un algoritmo determinado utilizando una estrategiaglobal, el criterio de evaluación del algoritmo aplicará a todas las páginas en memoria principal. En el caso de unaestrategia local, el criterio de evaluación del algoritmo se aplica sólo a las páginas en memoria principal que pertenecen al proceso que causó el fallo de página. Se describirán los algoritmos sin distinguir entre los dos tipos de estrategias.

1. Algoritmo de reemplazo óptimo

En sistemas operativos que utilizan paginación para el manejo de memoria, los algoritmos de reemplazo de páginas son usados para decidir qué páginas pueden ser sacadas de memoria cuando se necesita cargar una nueva y ya no hay espacios, minimizando la tasa de fallos en página.

Este algoritmo debe de tener el menor índice de fallos de página de todos los algoritmos. En teoría, este algoritmo debe de reemplazar la página que no va a ser usada por el periodo más largo de tiempo, por ejemplo, si hay una página A que será usada dentro de 10000 instrucciones, y una página B que será usada dentro de 2800 instrucciones, se debería eliminar la de la memoria la página A. Desafortunadamente, el algoritmo de reemplazo óptimo es fácil en teoría, pero prácticamente imposible de implementar, dado que requiere conocer afuturo las necesidades del sistema. Tal algoritmo existe y ha sido llamado OPT o MIN, pero se usa únicamente para estudios de comparaciones.

Tabla 1. Ejemplo con tres marcos

- En el instante 3 se llena la memoria real.
- Se sigue a la página A, pero como la memoria está llena se debe decidir a quién quitar.
- La política óptima nos dice: De los que se tienen en memoria real hay que fijarse quién de ellos está másdistante en el futuro. Se observa que es la B.
- En el instante 4 se hizo el reemplazo de página.
- En 5 D ya está en memoria por tanto permanece igual, no hay fallo.
- En 9 como E la voy a utilizar en el futuro y C y D no aparecen escojo por política FIFO.

Tiempo	1	2	3	4	5	6	7	8	9	10	11	12
Referencia	**D**	**C**	**B**	**A**	**D**	**C**	**E**	**D**	**C**	**B**	**A**	**E**
Marco 0	D	D	D	D	D	D	D	D	D	B	B	B
Marco 1		C	C	C	C	C	C	C	C	C	A	A
Marco 2			B	A	A	A	E	E	E	E	E	E
¿Fallo?	X	X	X	X			X			X	X	

Fallos=7/12

Rendimiento=42 (Ver 4. Rendimiento de estrategias de reemplazo).

2. Algoritmo FIFO (First Input - First Output)

En este algoritmo se trata a los marcos asignados a un proceso como un buffer circular y las páginas se suprimen de memoria según la técnica de espera circular (round-robin). Todo lo que se necesita es un puntero que circule a través de los marcos del proceso. Esta es, por tanto, una de las políticas de reemplazo más sencillas de implementar.

La lógica que hay detrás de esta elección, además de su sencillez, es reemplazar la página que ha estado más tiempo en memoria: Una página introducida en memoria hace mucho tiempo puede haber caído en desuso.Este razonamiento será a menudo incorrecto, porque habrá regiones de programa o de datos que son muy usadas a lo largo de la vida de un programa. Con el algoritmo FIFO, estas páginas se cargarán y expulsarán repetidas veces.

Tabla 2. Ejemplo de funcionamiento algoritmo

Tiempo	1	2	3	4	5	6	7	8	9	10	11	12
Referencia	**D**	**C**	**B**	**A**	**D**	**C**	**E**	**D**	**C**	**B**	**A**	**E**
Marco 0	D	D	D	A	A	A	E	E	E	E	E	E
Marco 1		C	C	C	D	D	D	D	D	B	B	B
Marco 2			B	B	B	C	C	C	C	C	A	A
¿Fallo?	X	X	X	X	X	X	X			X	X	

- D
- D C
- D C B -- Memoria llena, Se reemplaza la página en la primera posición, en este caso la D.
- C B A – C sería la candidata a reemplazar y A que acabo de entrar se va al final de la lista.
- B A D – Entró la D y queda al final.
- A D C
- D C E
- D C E
- D C E
- C E B
- E B A
- E B A

Fallos = 9/12

Rendimiento=25

3. Algoritmo LRU (Least Recently Used)

El algoritmo LRU está basado en el principio de proximidad temporal de referencias: si es probable que se vuelvan a referenciar las páginas accedidas recientemente, la página que se debe reemplazar es la que no se ha referenciado desde hace más tiempo.

El algoritmo LRU no sufre la anomalía de Belady. Pertenece a una clase de algoritmos denominados algoritmos de pila. La propiedad de estos algoritmos es que las páginas residentes en memoria para un sistema con marcos de página son siempre un subconjunto de las que habría en un sistema con n+1 marcos. Esta propiedad asegura que un algoritmo de este tipo nunca sufrirá la anomalía de Belady. Hay un aspecto sutil en este algoritmo cuando se considera su versión global. A la hora de seleccionar una página no habría que tener en cuenta el tiempo de acceso real, sino el tiempo lógico de cada proceso, es decir, habría que seleccionar la página que haya sido menos recientemente usada teniendo en cuenta el tiempo lógico de este proceso.

La estrategia LRU se puede realizar con una estructura de listas que contenga una entrada por cada marco de página ocupado. Cada vez que se hace referencia a un marco de página, la entrada correspondiente a esta página se coloca al principio de la lista, y las entradas más antiguas se llevan al final de la lista. Cuando hay que reemplazar una página para dejar espacio a otra entrante, se selecciona la entrada al final de la lista, se libera el marco de página correspondiente, se coloca la página entrante en el marco de página y la entrada correspondiente a ese marco de página se coloca al principio de la lista, porque esa página es ahora la que ha sido utilizada más recientemente.

A pesar de que el algoritmo LRU es realizable y proporciona un rendimiento bastante bueno, su implementación eficiente es difícil y requiere un considerable apoyo hardware. Una implementación del algoritmo podría basarse en utilizar un contador que se incremente por cada referencia a memoria. Cada posición de la tabla de páginas ha de tener un campo de tamaño suficiente para que quepa el contador. Cuando se referencia a una página, el valor actual del contador se copia por hardware a la posición de la tabla correspondiente a esa página. Cuando se produce un fallo de página, el sistema operativo examina los contadores de todas las páginas residentes en memoria y selecciona como víctima aquella que tiene el

valor menor. Esta implementación es factible, aunque requiere un hardware complejo y muy específico.

Tabla 3. Ejemplo algoritmo LRU

Tiempo	1	2	3	4	5	6	7	8	9	10	11	12
Referencia	**D**	**C**	**B**	**A**	**D**	**C**	**E**	**D**	**C**	**B**	**A**	**E**
Marco 0	D	D	D	A	A	A	E	E	E	B	B	B
Marco 1		C	C	C	D	D	D	D	D	D	A	A
Marco 2			B	B	B	C	C	C	C	C	C	E
¿Fallo?	X	X	X	X	X	X	X			X	X	X

- D
- D C
- D C B
- C B A – Busco la que esté más lejana en el tiempo y reemplazo, en este caso es la D.
- B A D
- A D C
- D C E
- C E D – Se puede ver la diferencia con FIFO, como acabo de acceder a D, está pasa al final.
- E D C
- D C B
- C B S
- B S E

Fallos = 10/12

Rendimiento=16

4. Segunda oportunidad

Este algoritmo es una modificación sencilla de FIFO que evita el problema de desalojar una página que se usa mucho y consiste en examinar el bit R de la página más antigua. Si es 0, la página es antigua y no utilizada,por lo que se reemplaza de manera inmediata. Si el bit es 1, R se pone a cero, la página se coloca al final de la lista de páginas, como si hubiera llegado en ese momento a la memoria. Después continúa la búsqueda siguiendo la lista.

Supongamos que ocurre un fallo de página en el instante 20. La página más antigua es A, que llego en elinstante 0, al iniciar el proceso. Si A tiene el bit R

igual 0, se retira de la memoria, ya sea mediante su escritura en el disco (se tiene nueva información) o solo se abandona (en caso contrario). Por otro lado, si Rigual 1, A se coloca al final de la lista y su tiempo de carga cambia al tiempo activo (20). Se limpia entonces el bit R. La búsqueda de una página adecuada prosigue con B.

Lo que hace la segunda oportunidad es buscar una página antigua sin referencias durante el anterior intervalo de tiempo. Si todas las páginas tienen alguna referencia, el algoritmo de la segunda oportunidad deriva de un simple FIFO.

5. Algoritmo de reloj

Es una variante del algoritmo de la segunda oportunidad y del algoritmo LRU. En este algoritmo se hace uso de los dos bits de estado que están asociados a cada página. Estos bits son el bit R, el cual se activa cuando se hace referencia (lectura/escritura) a la página asociada; y M, que se activa cuando la página asociada es modificada (escritura).

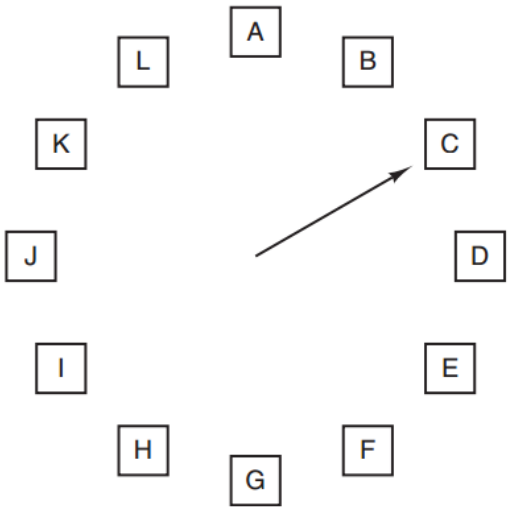
En este algoritmo se ordena las páginas reales en una lista circular y la recorre en el sentido horario, una manecilla apunta a la página más antigua.

Al presentarse un fallo de página, se revisa la página a la que apunta la manecilla, si R es igual a cero, se desaloja, la nueva página se inserta en su lugar y la manecilla se incrementa en un lugar. Si R es igual a uno secambia a cero y la manecilla se adelanta a la siguiente página. Se repite el proceso hasta que haya una página con R igual a cero.

6. Buffering de páginas

Una situación que intentan evitar la mayoría de os sistemas es la que se produce cuando la página seleccionadapara reemplazar esta modificada. En este caso, el tratamiento del fallo de página implica dos operaciones al disco, aumentando considerablemente el tiempo de servicio del fallo.

Una solución a lo anterior es utilizar el buffering de páginas. Esto consiste en mantener un conjunto de marcosde páginas libres. Cuando se produce un fallo de



Cuando ocurre un fallo de página, la página a la que apunta la manecilla se inspecciona. La acción a realizar depende del bit R:
R = 0: Desaloja la página
R = 1: Desactiva R y avanza la manecilla

página, se usa un marco de página libre, pero no se aplica el algoritmo de reemplazo, esto quiere decir que se consume un marco de página, pero no se libera otro. Cuandoel sistema operativo detecta que el número de marcos de página disminuye por debajo de un cierto umbral, aplica repetitivamente el algoritmo de reemplazo hasta que el número de marcos libres sea suficiente. Las páginas liberadas que no están modificadas pasan a la lista de marcos libres. Las páginas que han sido modificadas pasan a la lista de modificadas. Las páginas que están en cualquiera de las dos listas pueden recuperarse si vuelven a referenciarse.

En este caso la rutina de fallo de página recupera la página directamentede la lista y actualiza la entrada correspondiente de la tabla de páginas para conectarla. Cabe destacar que este fallo no implicaría operaciones de entrada/salida.

Esta estrategia puede mejorar el rendimiento de algoritmos de reemplazo que no sean muy efectivos. Así m si el algoritmo de reemplazo decide revocar una página que en realidad está siendo usada por un proceso, se producirá inmediatamente un fallo de página que la recuperará de las listas.

RENDIMIENTO DE ESTRATEGIAS DE REEMPLAZO

El rendimiento se define como: $R=1-F$

donde F = $\frac{\textit{Número de fallos de página}}{\textit{Número total de referencias}}$

5. EJEMPLO DE SISTEMA DE MEMORIA VIRTUAL: PROCESADOR PENTIUM II

El Pentium II dispone de un sistema de gestión de memoria virtual con posibilidad de segmentación y paginación. Los dos mecanismos se pueden activar o desactivar con independencia, dando pues lugar a cuatro formas de funcionamiento del sistema de memoria:

a. **Memoria no segmentada no paginada:** la dirección virtual coincide con la dirección física. Esta alternativa resulta útil cuando el procesador se utiliza como controlador de sistemas empotrados.
b. **Memoria paginada no segmentada:** la memoria constituye un espacio lineal de direcciones paginado. La protección y la gestión de memoria se realizan a través de la paginación.
c. **Memoria segmentada no paginada:** la memoria constituye un conjunto de espacios de direcciones virtuales (lógicas). Esta alternativa presenta la ventaja frente a la paginación en que proporciona, si es necesario, mecanismos de protección a nivel de byte. Además, garantiza que la tabla de segmentos se encuentra ubicada en el procesador cuando el segmento está en memoria. Por ello, la segmentación sin páginas da lugar a tiempos de acceso predecibles.

d. **Memoria segmentada paginada:** se utilizan simultáneamente los dos mecanismos, la segmentación para definir particiones lógicas de memoria en el control de acceso, y la paginación u para gestionar la asignación de memoria dentro de las particiones.

CONCLUSIONES

El uso de la memoria virtual ayuda enormemente a la gestión de procesos y archivos dentro de un sistema operativo, el que podría verse demasiado restringido si solo utilizara memoria física para trabajar. Los algoritmos de segmentación o paginación como vimos en esta oportunidad nos dan una forma más eficiente dependiendo de cuál se utilice para realizar los cambios de páginas en la memoria, de esta forma obtenemos mayor eficiencia y velocidad a la hora de ejecutar los procesos. Para el caso del algoritmo de la segunda oportunidad nos damos cuenta que mejora el algoritmo FIFO. Por otro lado, Si se incrementa memoria real (más marcos) los fallos de páginas son menores. LRU con 4 marcos,rendimiento 33% en FIFO 10 fallos a diferencia que con 3 marcos 9 fallos (Anomalía de Belady).