Juan Menduiña

Trabajo Práctico N° 4:
Paginación por Demanda. Algoritmos de Reemplazos. Memoria Virtual. Entrada-Salida. Dispositivos. Administración de Discos. Sistemas de Archivos. Buffer Caché.

PARTE I: Administración de Memoria Principal.

Ejercicio 1: Memoria Virtual.

(a) *Describir qué beneficios introduce este esquema de administración de la memoria.*

Los beneficios que introduce este esquema de administración de la memoria son:

1. Aumento del espacio de direcciones disponible:
   - Cada proceso percibe un espacio de direcciones lógico mucho mayor que la memoria física real.
   - Permite ejecutar programas más grandes que la RAM disponible ("ilusión" de memoria ilimitada).
2. Protección y aislamiento entre procesos:
   - Cada proceso tiene su propio espacio de direcciones virtual, separado del resto.
   - Un proceso no puede leer ni escribir memoria de otro, aumentando seguridad y estabilidad.
3. Multiprogramación más eficiente:
   - Permite que más procesos estén cargados, parcialmente, en memoria al mismo tiempo.
   - Aumenta la utilización de la CPU y el rendimiento del sistema.
4. Carga diferencial de páginas (*paging on demand*):
   - Sólo se cargan en memoria las partes, realmente, utilizadas del programa.
   - Reduce la cantidad de RAM necesaria para ejecutar procesos.
5. Simplificación para el programador:
   - El programador no necesita gestionar, manualmente, memoria física.
   - Puede trabajar con direcciones continuas, aunque, físicamente, la memoria esté fragmentada.
6. Flexibilidad para compartir memoria:
   - Permite compartir páginas entre procesos (por ejemplo, bibliotecas dinámicas).
   - Ahorra memoria cuando varios procesos usan el mismo código.
7. Soporte para técnicas avanzadas:
   - Memoria compartida entre procesos.
   - *Copy-on-write*.
   - *Swapping*.
   - *Demand paging* y algoritmos de reemplazo.

(b) *¿En qué se debe apoyar el SO para su implementación?*

Para su implementación, el SO se debe apoyar en:

1. Unidad de Gestión de Memoria (MMU - *Memory Management Unit*):
   - Es el *hardware* encargado de traducir direcciones lógicas a físicas.
   - Realiza el mapeo mediante tablas de páginas.
   - Detecta fallos de página (*page faults*).
2. Tablas de páginas:
   - Estructuras que mantienen:
     - El marco de página en RAM (si está presente).
     - Bits de control: presente/ausente, modificado, referencia, protección.
   - Son esenciales para la traducción y para aplicar algoritmos de reemplazo.
3. Mecanismos de interrupción:
   - Interrupciones por fallo de página: Notifican al SO que una página requerida no está en RAM.
   - Permiten que el SO ejecute rutinas de manejo de páginas (traer la página desde disco, elegir víctima, etc.).
4. Memoria secundaria (disco o SSD):
   - Alberga la parte del espacio de direcciones que no cabe en RAM.
   - Incluye:
     - Área de *swap*.
     - Archivos ejecutables mapeados.
   - Sirve como respaldo para almacenar páginas ausentes.
5. Algoritmos de reemplazo de páginas:
   - El SO debe implementar criterios para decidir qué página sacar cuando la RAM está llena (FIFO, LRU, NRU, *Clock* / *Second Chance*, OPT -teórico-).
6. Rutinas del SO para manejo de páginas:
   - El SO debe:
     - Cargar la página desde disco.
     - Actualizar las tablas de páginas.
     - Marcar bits (modificado, referencia).
     - Reanudar el proceso interrumpido.
     - Estas rutinas constituyen el *page fault handler*.

(c) *Al implementar está técnica utilizando paginación por demanda, las tablas de páginas de un proceso deben contar con información adicional además del marco donde se encuentra la página. ¿Cuál es esta información? ¿Por qué es necesaria?*

Cuando se implementa memoria virtual utilizando paginación por demanda, las tablas de páginas de un proceso deben contar con información adicional además del marco donde se encuentra la página. Esto se debe a que muchas páginas no están cargadas en memoria física y el sistema operativo debe poder gestionar qué páginas están presentes, cuáles fueron modificadas, etc.

Esta información es:

1. Bit de presencia/ausencia (*Present bit*):
   - Indica si la página está cargada en RAM.
   - Necesario: Sin este bit, la MMU no puede determinar si debe generar un *page fault* cuando la página aún no está en memoria.
2. Bit de modificado (*Dirty bit*):
   - Indica si la página fue escrita desde que se cargó.
   - Necesario: Si está en 1, la página debe copiarse al disco antes de ser reemplazada, si no, se puede descartar.
3. Bit de referencia (*Referenced/Accessed bit*):
   - Indica si la página fue usada recientemente.
   - Necesario: Es fundamental para los algoritmos de reemplazo (Clock, LRU aproximado) para decidir qué página es candidata a ser expulsada.
4. Bits de protección:
   - Especifican permisos: lectura, escritura, ejecución.
   - Necesario: Previene accesos inválidos o maliciosos; permite que un *page fault* se use también para protección.
5. Bit de validación del contenido (*Valid/Invalid*):
   - Diferencia entre:
     - Página válida, pero no presente (*swap*).
     - Página inválida (dirección que el proceso no debería usar).
   - Necesario: Evita accesos ilegales al espacio del proceso.
6. Información sobre el lugar en disco (parcial o completa):
   - Puede ser:
     - Índice del bloque en el área de *swap*.
     - Identificador en un archivo ejecutable mapeado.
   - Necesario: Para que el SO sepa dónde buscar la página cuando ocurre un *page fault*.

En resumen, utilizando paginación por demanda, cada entrada de tabla de páginas debe contener: Presente/Ausente, Modificado, Referenciado, Permisos, Validez y Ubicación en disco. Toda esta información es necesaria porque permite: generar y manejar *page faults*; decidir qué páginas reemplazar; proteger el acceso a memoria; recuperar o descartar páginas desde/ hacia disco.

Ejercicio 2: Fallos de Página (*Page Faults*).

(a) *¿Cuándo se producen?*

Un fallo de página (*page fault*) se produce cuando un proceso intenta acceder a una página que no se encuentra cargada en la memoria física (RAM). Esto ocurre cuando la entrada de la tabla de páginas para esa página indica que no está presente (bit de presencia= 0). La MMU detecta que la dirección virtual no puede traducirse porque la página no está en RAM. Entonces, genera una interrupción al sistema operativo.

(b) *¿Quién es responsable de detectar un fallo de página?*

El responsable de detectar un fallo de página es la Unidad de Gestión de Memoria (MMU - *Memory Management Unit*).

La MMU intenta traducir la dirección virtual a una dirección física utilizando la tabla de páginas. Si la entrada correspondiente indica que la página no está presente en memoria (bit de presencia= 0) o es inválida, la MMU no puede completar la traducción y, entonces, genera una interrupción de *hardware* o *trap*. Esa interrupción se envía al sistema operativo, que, luego, se encarga de manejar el fallo (traer la página desde disco, elegir víctima, actualizar la tabla de páginas, etc.), pero la detección inicial es, puramente, responsabilidad de la MMU.

(c) *Describir las acciones que emprende el Kernel cuando se produce un fallo de página.*

Las acciones que emprende el *Kernel* cuando se produce un fallo de página son:

1. Verificar el tipo de acceso y la validez de la página:
   - El Kernel consulta la tabla de páginas para determinar:
     - Si la página pertenece al proceso.
     - Si el acceso es válido (lectura, escritura, ejecución).
     - Si la página, simplemente, no está presente o si el acceso es ilegal.
   - Si es un acceso ilegal, se aborta el proceso (*segmentation fault*).
2. Identificar la ubicación de la página en disco:
   - Si el acceso es válido, determina dónde se encuentra la página:
     - En el área de swap.
     - En un archivo ejecutable o de datos mapeado.
     - O si es una página nueva (*heap/stack*) a inicializar.
3. Buscar un marco libre en memoria física:
   - El Kernel necesita un marco para cargar la página.
     - Si hay marcos libres, usa uno.
     - Si no hay marcos libres, debe aplicar un algoritmo de reemplazo de páginas (FIFO, LRU, *Clock*, etc.) para elegir una "víctima".
4. Si la página víctima está modificada, escribirla en disco:
   - Si el *dirty bit*= 1, la página debe escribirse en el área de *swap*.

- Si no está modificada, simplemente, se descarta (no hay que copiar nada).

5. Cargar la nueva página desde disco hacia RAM:
   - La página válida solicitada se copia desde:
     - *Swap*.
     - Archivo ejecutable.
     - O se inicializa si es una página nueva.
6. Actualizar la tabla de páginas del proceso:
   - Se marca la entrada como presente.
   - Se actualiza el número de marco.
   - Se reinician los bits de referencia y modificado.
   - Se realiza cualquier actualización de protección o metadatos.
7. Actualizar la TLB (si es necesario):
   - Si la entrada estaba en la TLB con estado inválido, se invalida y se reemplaza.
   - Puede requerir un TLB *flush* parcial o selectivo.
8. Reanudar la ejecución del proceso:
   - Una vez que la página está disponible:
     - La instrucción interrumpida se reintenta.
     - El proceso continúa como si nada hubiera pasado.

Ejercicio 3.

Suponer que la tabla de páginas para un proceso que se está ejecutando es la que se muestra a continuación:

Página	***Bit V***	***Bit R***	***Bit M***	***Marco***
0	*1*	*1*	*0*	*4*
1	*1*	*1*	*1*	*7*
2	*0*	*0*	*0*	*-*
3	*1*	*0*	*0*	*2*
4	*0*	*0*	*0*	*-*
5	*1*	*0*	*1*	*0*

Asumiendo que:

- *El tamaño de la página es de 512 bytes.*
- *Cada dirección de memoria referencia 1 byte.*
- *Los marcos se encuentran contiguos y en orden en memoria (0, 1, 2, ...) a partir de la dirección física 0.*

¿Qué dirección física, si existe, correspondería a cada una de las siguientes direcciones virtuales? (No gestionar ningún fallo de página en caso de producirse).

(a) *1052.*

Página= 1052 DIV 512
Página= 2.

Desplazamiento= 1052 MOD 512
Desplazamiento= 28.

Por lo tanto, no existe dirección física para esta dirección virtual, ya que la página 2 no está presente en memoria.

(b) *2221.*

Página= 2221 DIV 512
Página= 4.

Desplazamiento= 2221 MOD 512
Desplazamiento= 173.

Por lo tanto, no existe dirección física para esta dirección virtual, ya que la página 4 no está presente en memoria.

(c) *5499.*

Página= 5499 DIV 512
Página= 10.

Desplazamiento= 5499 MOD 512
Desplazamiento= 379.

Por lo tanto, no existe dirección física para esta dirección virtual, ya que la página 10 no existe en la tabla de páginas.

(d) *3101.*

Página= 3101 DIV 512
Página= 6.

Desplazamiento= 3101 MOD 512
Desplazamiento= 29.

Por lo tanto, no existe dirección física para esta dirección virtual, ya que la página 6 no existe en la tabla de páginas.

Ejercicio 4.

Analizar cómo impacta el tamaño de una página (pequeña o grande) en paginación por demanda.

A continuación, se analiza cómo impacta el tamaño de una página en paginación por demanda:

1. Tamaño de página pequeño:
   - Ventajas:
     - Menor fragmentación interna: Como las páginas son chicas, es menos probable que un proceso deje "espacio desperdiciado" dentro de una página.
     - Ajuste más fino entre el programa y las páginas: se cargan sólo las partes del programa necesarias; mejor uso de la memoria RAM.
     - Mayor grado de multiprogramación: Como cada proceso usa menos RAM, entran más procesos simultáneamente.
     - Mejor localización de fallos de página: Las páginas pequeñas permiten manejar mejor patrones de acceso dispersos.
   - Desventajas:
     - Tablas de páginas más grandes: Como hay más páginas por proceso, la tabla crece y consume más memoria.
     - Mayor *overhead* en el TLB: Más entradas necesarias, más *misses*, mayor tiempo de acceso.
     - Más fallos de páginas totales: Porque cada página contiene menos datos, se necesita traer más páginas desde disco.
2. Tamaño de página grande:
   - Ventajas:
     - Tablas de páginas más pequeñas: Menos páginas por proceso, por lo que la tabla es más corta y eficiente.
     - Menos TLB *misses*: Cada entrada cubre más direcciones, por lo que hay mayor probabilidad de aciertos en el TLB.
     - Mayor eficiencia en accesos secuenciales: Para cargas masivas o *streams* grandes, las páginas grandes reducen fallos.
     - Mejor *throughput* del disco: Cargar páginas grandes usa mejor el ancho de banda del disco.
   - Desventajas:
     - Mayor fragmentación interna: Mucho espacio dentro de las páginas queda sin usar.
     - Pérdida de precisión en la asignación: Se cargan grandes bloques aunque se necesite muy poco, por lo que hay desperdicio de RAM.
     - Menor multiprogramación: Cada proceso ocupa más memoria, por lo que entran menos procesos al mismo tiempo.
     - Si la localidad es mala, empeora la *performance*: Se traen páginas enormes desde disco que no se usan.

Ejercicio 5: Asignación de Marcos a un Proceso (Conjunto de Trabajo o *Working Set*).

Con la memoria virtual paginada, no se requiere que todas las páginas de un proceso se encuentren en memoria. El SO debe controlar cuantas páginas de un proceso puede tener en la memoria principal. Existen 2 políticas que se pueden utilizar:

- *Asignación Fija.*
- *Asignación Dinámica.*

(a) *Describir cómo trabajan estas 2 políticas.*

A continuación, se describe cómo trabajan estas 2 políticas:

1. Asignación Fija:
   - El sistema operativo asigna un número fijo de marcos de página a cada proceso desde el momento en que se crea. Ese número no cambia durante toda la ejecución del proceso.
   - Funcionamiento:
     - Cuando el proceso inicia, el SO le asigna una cantidad determinada de marcos (por ejemplo, 10 marcos).
     - El proceso sólo puede usar esa cantidad, aunque necesite más o aunque le sobren.
     - Si el proceso necesita una página nueva y no tiene marcos libres, debe expulsar una de sus propias páginas (*page replacement local*).
     - No puede pedir marcos adicionales.
   - Ventajas:
     - Simplicidad de implementación.
     - Predecible en cuanto a uso de memoria.
     - Evita que un proceso consuma demasiados marcos y afecte a otros.
   - Desventajas:
     - Poco flexible.
     - Algunos procesos podrían quedar con marcos de más y otros con marcos insuficientes.
     - Aumenta el riesgo de *thrashing* si el conjunto de trabajo del proceso crece.
2. Asignación Dinámica:
   - El sistema operativo asigna un número variable de marcos a cada proceso, que puede aumentar o disminuir durante su ejecución.
   - Funcionamiento:
     - El SO monitorea el comportamiento del proceso (frecuencia de fallos de página, *working set*, etc.).
     - Si el proceso muestra muchos fallos de página, se le otorgan más marcos.
     - Si usa pocos marcos o su *working set* se achica, se le quitan marcos y se reasignan a otros procesos.
     - Permite balancear, dinámicamente, la memoria total entre procesos.
   - Ventajas:
     - Mayor eficiencia global.
     - Se adapta a las necesidades reales del proceso.

    - Reduce el riesgo de *thrashing* (cada proceso puede recibir marcos según su *working set*).
- Desventajas:
    - Mayor complejidad del sistema operativo.
    - Requiere monitorear fallos de página y comportamiento del proceso.
    - Si no está bien regulado, un proceso puede crecer demasiado y perjudicar a otros.

(b) *Dada la siguiente tabla de procesos y las páginas que ellos ocupan, y teniéndose 40 marcos en la memoria principal, ¿cuántos marcos le corresponderá a cada proceso si se usa la técnica de Asignación Fija:*

- *Reparto Equitativo?*
- *Reparto Proporcional?*

Proceso	***Total de Páginas Usadas***
1	*15*
2	*20*
3	*20*
4	*8*

Reparto Equitativo: Se reparten los mismos marcos a cada proceso, sin importar su tamaño.

Marcos por proceso= $\frac{40\ marcos}{4\ procesos}$
Marcos por proceso= 10.

Por lo tanto, si se usa la técnica de Asignación Fija con Reparto Equitativo, a cada proceso, le corresponderá 10 marcos.

Reparto Proporcional: Se reparte según la cantidad de páginas que usa cada proceso, respecto al total.

Marcos proceso 1= $40\ \frac{15}{63}$
Marcos proceso 1= 9,52
Marcos proceso 1≅ 9.

Marcos proceso 2= $40\ \frac{20}{63}$
Marcos proceso 2= 12,7.
Marcos proceso 2≅ 13.

Marcos proceso 3= $40\ \frac{20}{63}$
Marcos proceso 3= 12,7.
Marcos proceso 3≅ 13.

Marcos proceso 4= 40 $\frac{8}{63}$
Marcos proceso 4= 5,08
Marcos proceso 4≅ 5.

Por lo tanto, si se usa la técnica de Asignación Fija con Reparto Proporcional, a cada proceso, le corresponderá 9, 13, 13 y 5 marcos, respectivamente.

© *¿Cuál de los 2 repartos usados en (b) resultó más eficiente? Justificar.*

Déficit por proceso= (páginas – marcos) si $>$ 0.

Reparto Equitativo:

Déficit por proceso=

- P1: 15 - 10= 5.
- P2: 20 - 10= 10.
- P3: 20 – 10= 10.
- P4: 8 – 10= -2.

Déficit total= 5 + 10 + 10
Déficit total= 25.

Reparto Proporcional:

Déficit por proceso=

- P1: 15 – 9= 6.
- P2: 20 – 13= 7.
- P3: 20 – 13= 7.
- P4: 8 – 5= 3.

Déficit total= 6 + 7 + 7 + 3
Déficit total= 23.

Por lo tanto, de los 2 repartos usados en (b), resultó ser más eficiente el Reparto Proporcional, ya que es mayor la probabilidad de que el conjunto de trabajo (*working set*) de cada proceso esté cubierto por sus marcos, lo que reduce fallos de página, aumenta el *throughput* y mejora la utilización de CPU.

Ejercicio 6: Reemplazo de Páginas (Selección de una Víctima).

¿Qué sucede cuando todos los marcos en la memoria principal están usados por las páginas de los procesos y se produce un fallo de página? El SO debe seleccionar una de las páginas que se encuentra en memoria principal (en un marco) como víctima, la que será reemplazada por la nueva página que produjo el fallo.

Considerar los siguientes algoritmos de selección de víctimas básicos:

- *LRU.*
- *FIFO.*
- *OPT (Óptimo).*
- *Second Chance.*

(a) *Clasificar estos algoritmos de malo a bueno de acuerdo a la tasa de fallos de página que se obtienen al utilizarlos.*

A continuación, se clasifican estos algoritmos de malo a bueno de acuerdo a la tasa de fallos de página que se obtienen al utilizarlos:

- FIFO (*First-In, First-Out*): El más simple (expulsa la página más antigua), pero puede comportarse mal frente a ciertas secuencias de referencias (sufre la anomalía de Belady) y, por eso, suele dar la peor tasa de fallos entre los listados.
- *Second Chance*: Es una aproximación barata de LRU (usa el bit R para dar una "segunda oportunidad"). Normalmente, tiene rendimiento algo inferior a LRU, pero mucho mejor que FIFO en la mayoría de los casos, y con menor costo de implementación.
- LRU (*Least Recently Used*): Aproxima el comportamiento óptimo al expulsar la página menos recientemente usada; muy buena tasa de fallos en la práctica y es un algoritmo "stack" (no sufre la anomalía de Belady).
- OPT (Óptimo): Conoce el futuro y, por lo tanto, minimiza, exactamente, la cantidad de fallos de página. Es teórico (no implementable en la práctica), pero es la referencia ideal.

(b) *Analizar su funcionamiento. ¿Cómo se podrían implementar?*

1. FIFO (*First-In, First-Out*):
   - Funcionamiento:
     - Reemplaza la página que lleva más tiempo en memoria, sin importar si fue usada recientemente o no.
     - Mantiene las páginas en una cola ordenada por tiempo de carga.
   - Implementación:
     - Cola FIFO: cuando una página entra a memoria, se encola; cuando hay que reemplazar, se desencola la más antigua.
     - Muy fácil de implementar.

    - No usa bits R ni M (sólo ayuda a reducir I/O si M= 1, pero no afecta la elección de víctima).
- Ventajas:
    - Simple, bajo costo.
- Desventajas:
    - Mala tasa de fallos.
    - Sufre anomalía de Belady.

2. *Second Chance*:
    - Funcionamiento:
        - Es una mejora de FIFO usando el bit R.
        - Mantiene una cola como FIFO.
        - Cuando la página más antigua tiene R= 1, no se la reemplaza: se pone R= 0; se mueve al final de la cola (segunda oportunidad).
        - Se sigue avanzando en la cola hasta encontrar una página con R= 0, que será la víctima.
    - Implementación:
        - Cola circular (implementación tipo *Clock*).
        - Requiere un bit R por página y un puntero que recorre los marcos.
    - Ventajas:
        - Mejor que FIFO.
        - Barato de implementar (sólo revisar bits R, no necesita *timestamps*).
    - Desventajas:
        - Peor que LRU.
        - No considera antigüedad real, sólo si fue usada recientemente.
3. LRU (*Least Recently Used*):
    - Funcionamiento:
        - Reemplaza la página que hace más tiempo que no se usa.
        - Imita el comportamiento ideal: Asume que lo que no se usó recientemente es menos probable que vuelva a usarse.
    - Implementación:
        - Difícil en *hardware* puro, por eso se aplican aproximaciones.
        - Implementación exacta (costosa): Guardar un *timestamp* cada vez que se accede a una página. Para elegir la víctima, buscar la página con *timestamp* más viejo.
        - Implementaciones aproximadas: *Clock* mejorado; NRU (*Not Recently Used*); LRU por bits (matriz de nxn bits).
    - Ventajas:
        - Excelente tasa de fallos.
        - No sufre anomalía de Belady.
    - Desventajas:
        - Implementación costosa, salvo aproximaciones.
4. OPT (Óptimo):
    - Funcionamiento:
        - Expulsa la página que será usada más lejos en el futuro.
    - Implementación:
        - No se puede implementar en la vida real porque se necesitan conocer las referencias futuras.
        - Se usa: como referencia teórica para evaluar otros algoritmos; en simuladores de fallos de página.

- Ventajas:
  - Mínima tasa de fallos posible.
- Desventajas:
  - Sólo teórico.

(c) *Se sabe que la página a ser reemplazada puede estar modificada. ¿Cómo detecta el Kernel esta situación? ¿Qué acciones adicionales deben realizarse cuando sucede esta situación?*

Si la página a ser reemplazada está modificada, el *Kernel* detecta esta situación usando el bit M (*Modified*), que está en la entrada de la tabla de páginas correspondiente a esa página.

Las acciones adicionales que deben realizarse cuando sucede esta situación son:

1. Escribir la página modificada al disco (*swap*).
2. Marcar el bit M= 0.
3. Liberar el marco.

Ejercicio 7: Alcance del Reemplazo.

Al momento de tener que seleccionar una página víctima, el Kernel puede optar por 2 políticas a utilizar:

- *Reemplazo Local.*
- *Reemplazo Global.*

(a) *Describir cómo trabajan estas 2 políticas.*

A continuación, se describe cómo trabajan estas 2 políticas:

1. Reemplazo Local:
   - Funcionamiento:
     - Cada proceso tiene asignado un conjunto fijo de marcos.
     - Ante un fallo de página, sólo puede reemplazar una página que le pertenece a él mismo, es decir, dentro de su propio conjunto de marcos.
     - No puede tomar marcos que están asignados a otros procesos.
   - Consecuencias:
     - El desempeño del proceso depende, únicamente, de los marcos que tiene asignados.
     - Si un proceso recibe pocos marcos, puede sufrir muchos fallos de página.
     - Un proceso con baja actividad no libera marcos para otros.
     - Se evita la interferencia entre procesos.
   - Ventajas:
     - Aísla a los procesos: Uno muy demandante no perjudica a los demás.
     - Fácil de controlar.
   - Desventajas:
     - Puede llevar al subaprovechamiento de memoria (si un proceso no usa todos sus marcos).
     - Procesos mal asignados pueden entrar en *thrashing* sin que otros puedan ayudarlos.
2. Reemplazo Global:
   - Funcionamiento:
     - Ante un fallo de página, el proceso puede reemplazar cualquier página del sistema, sin importar a qué proceso pertenezca.
     - La elección de víctima se hace sobre todas las páginas residentes en memoria, no sólo las del proceso actual.
   - Consecuencias:
     - Procesos con alta demanda pueden "robar" marcos a procesos con baja demanda.
     - La memoria se usa de forma más dinámica y eficiente.
     - Pero un proceso puede afectar, gravemente, el rendimiento de otros.
   - Ventajas:
     - Aumenta la utilización global de la memoria.
     - Se adapta, dinámicamente, a los procesos que más memoria requieren.
     - Reduce el riesgo de tener marcos ociosos.
   - Desventajas:

- No aísla procesos: Uno puede perjudicar a los demás quitándoles marcos.
- Puede provocar *thrashing* distribuido entre procesos.
- Más complejo de controlar.

(b) *¿Es posible utilizar la política de "Asignación Fija" de marcos junto con la política de "Reemplazo Global"? Justificar.*

No, no es posible utilizar la política de "Asignación Fija" de marcos junto con la política de "Reemplazo Global", ya que son conceptos contradictorios:

- Asignación Fija impone un límite rígido de marcos para cada proceso.
- Reemplazo Global rompe ese límite porque permite que un proceso use marcos que, originalmente, pertenecen a otro.

Si el sistema permitiera Reemplazo Global mientras dice usar Asignación Fija, dejaría de ser fija, porque un proceso podría perder marcos o ganar marcos por decisiones globales. El Reemplazo Global, por definición, requiere Asignación Dinámica.

En resumen, no es posible porque la Asignación Fija garantiza un conjunto de marcos exclusivo para cada proceso, mientras que el Reemplazo Global permite que un proceso reemplace páginas pertenecientes a otros, violando, así, la exclusividad de la Asignación Fija.

Ejercicio 8.

Considerar la siguiente secuencia de referencias de páginas:

1, 2, 15, 4, 6, 2, 1, 5, 6, 10, 4, 6, 7, 9, 1, 6, 12, 11, 12, 2, 3, 1, 8, 1, 13, 14, 15, 3, 8.

(a) *Si se disponen de 5 marcos, ¿cuántos fallos de página se producirán si se utilizan las siguientes técnicas de selección de víctima? (Considerar una política de Asignación Dinámica y Reemplazo Global).*

- *Second Chance.*
- *FIFO.*
- *LRU.*
- *OPT.*

(b) *Suponiendo que cada atención de un fallo se página requiere de 0,1 seg. Calcular el tiempo consumido por atención a los fallos de páginas para los algoritmos de (a).*

Ejercicio 9.

Ejercicio 10.

Ejercicio 11.

Ejercicio 12.

Ejercicio 13.

Ejercicio 14.

Ejercicio 15.

Ejercicio 16.

Ejercicio 17.

PARTE II: Administración de E/S.

Ejercicio 18.

Ejercicio 19.

Ejercicio 20.

Ejercicio 21.

Ejercicio 22.

Ejercicio 23.

Ejercicio 24.

Ejercicio 25.

Ejercicio 26.

PARTE III: Administración de Discos.

Ejercicio 27.

Ejercicio 28.

Ejercicio 29.

Ejercicio 30.

Ejercicio 31.

Ejercicio 32.

Ejercicio 33.

Ejercicio 34.

Ejercicio 35.

PARTE IV: Administración de Archivos.

Ejercicio 36.

Ejercicio 37.

Ejercicio 38.

Ejercicio 39.

Ejercicio 40.

Ejercicio 41.

Ejercicio 42.

Ejercicio 43.

Ejercicio 44.

Ejercicio 45.

Ejercicio 46.

PARTE V: Caché de Disco.

Ejercicio 47.