Unidad 2:
Gestión de Memoria

Tema 4, Memoria Virtual:

4.1 Comparación entre técnicas:

Gestión de memoria vs Memoria virtual.

4.2 Paginación por demanda:

Reemplazo de páginas, algoritmos de reemplazo, marcos.

4.3 Hiperpaginación y working set.

4.4 Otras consideraciones acerca de la memoria virtual.

4.1 Comparación entre técnicas.

- Introducción:
  - **Gestión de memoria:** *Es necesario que todo el proceso esté cargado en la memoria principal para poder ejecutarse.*
  - **Memoria virtual:** *Permite ejecutar procesos que podrían no estar totalmente cargados en la memoria.*
    - Se utiliza el disco como almacén secundario de procesos.
    - Idea: mantener en memoria principal sólo los fragmentos de cada proceso que se estén utilizando.
  - **Ventaja:** Los programas pueden ser más grandes que la memoria física. (Memoria como gran matriz de almacenaje).
  - **Desventaja:** No es fácil de implementar y podría reducir sustancialmente el rendimiento si no se usa con cuidado.

4.1 Comparación entre técnicas.

- Introducción:
  - **Recordando las técnicas de Gestión de Memoria:**
    - Era necesario colocar todo el espacio de direcciones lógico en la memoria física.
    - Limitación del tamaño de los programas al tamaño de la memoria.
    - Examinando programas reales observamos que:
      - En muchos casos no se necesita todo el programa.
      - Contienen código de error que casi nunca se ejecuta.
      - Asignación inadecuada, en muchos casos se reserva más memoria de la necesaria (vectores, tablas, etc...).
      - Opciones y funciones de muy escaso uso.
      - Posibilidad de superposiciones, no necesitamos todo el programa a un mismo tiempo.

4.1 Comparación entre técnicas.

- Introducción (Ventajas de la memoria virtual):
  - Si consiguiéramos la capacidad de ejecutar programas que no estuvieran todos en la memoria tendríamos ciertas ventajas:
    - **Posibilidad de tener cargados más procesos y de mayor tamaño.**
      - Programas no limitados por el tamaño de la memoria física. Programación para un espacio de direcciones virtuales extremadamente grande.
      - Como cada programa ocupa menos espacio en memoria, se podrían ejecutar más programas simultáneamente; aumentando el rendimiento de la CPU sin aumentar los tiempos de respuesta y retorno.
    - **Mejora del rendimiento** (y la multiprogramación).
    - **Incremento del uso de la CPU y reducción de la E/S.**
    - **Transparencia** de cara al usuario en la utilización de programas.
    - **Facilidad de utilización con paginación.**

4.1 Comparación entre técnicas.

- Trabajando con memoria virtual estamos obligados a:
  - Elegir cuanta memoria se le asigna a cada proceso.
  - Comprobar que la información realmente esté en memoria física.
  - Decidir qué cantidad de información de la memoria virtual se lleva a la memoria física.
  - Decidir qué información de la memoria física se sustituye si la zona de memoria de un proceso está totalmente ocupada.
  - La memoria generalmente se implementa con paginación por demanda, aunque también puede implementarse en un sistema con segmentación, segmentación paginada.

4.2 Paginación por demanda.

- Paginación por demanda:
  - Técnica más habitual y eficaz de implementar MV.
  - Los procesos residen en memoria secundaria.
  - Combina paginación con intercambio:
    - Cuando queremos ejecutar un proceso lo pasamos por intercambio a la memoria.
    - En vez de intercambiar un proceso entero, ahora sólo intercambiamos las páginas necesarias.
    - En lugar de un “intercambiador” hablaremos de un “paginador”, que sabe cuales son las páginas que va a usar.
    - Se evita leer y colocar en la memoria páginas que no se usan, reduciendo de esta forma el tiempo de intercambio y la cantidad de memoria física requerida.

4.2 Paginación por demanda.

- Paginación por demanda:
  - Necesitamos un soporte hardware para distinguir entre las **páginas que están en la memoria (válidas)** y las que **están sólo en disco (no válidas)** ⇒ esquema de *bit de validez/ no validez.*
    - Bit válido ⇒ Página en la memoria.
    - Bit no válido:
      - ⇒ la página no está en el espacio de direcciones lógico.
      - ⇒ es válida pero no está en la memoria.

4.2 Paginación por demanda.

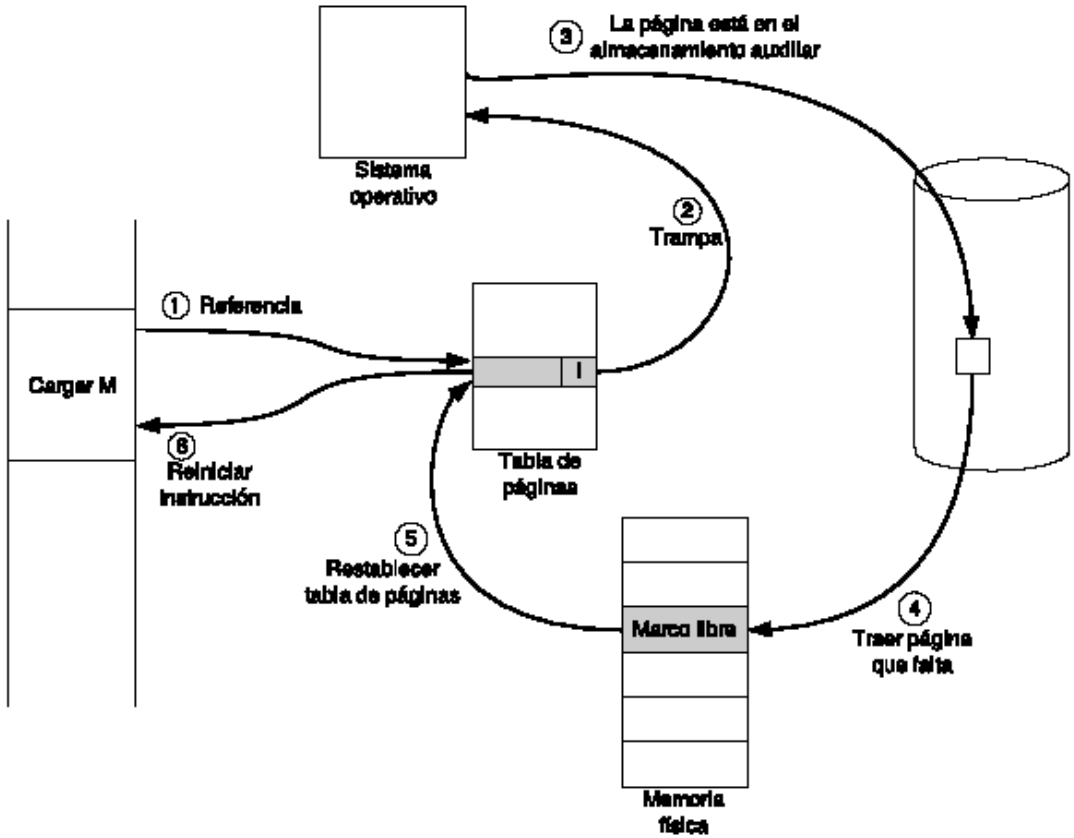
- Paginación por demanda:
  - **Idea fundamental:**
    - Si “adivinamos” y **traemos a la memoria sólo las páginas que se necesitan**, el proceso se ejecutará exactamente igual que si hubiéramos traído todas las páginas.
  - **¿Qué ocurre cuándo un proceso trata de usar una página que no se trajo a la memoria?.**
    - El acceso a una página no válida causa una **trampa de fallo de página**. El hardware de paginación, al traducir la dirección utilizando la tabla de páginas se percata de que el bit de no validez está encendido y transfiere el control al SO.
    - El fallo de página provoca que **el SO recupere del disco la página requerida**. Se actualiza la tabla de páginas y se reintenta la instrucción que ocasionó el fallo.

4.2 Paginación por demanda.

- Paginación por demanda:
  - **Pasos para manejar un fallo de página:**
    - 1. Consultamos una tabla interna que recoge si la referencia fue un acceso válido o no válido a la memoria.
    - 2. Referencia válida ⇒ final del proceso.
      Referencia no válida ⇒ Traemos la página.
    - 3. Encontramos un marco libre.
    - 4. Leemos la página deseada y la colocamos en el marco recién asignado.
    - 5. Final de la lectura ⇒ modificación de la tabla interna y la tdp, de modo que indique que la página ya está en memoria.
    - 6. Reinicio de la instrucción interrumpida por la trampa de dirección no válida. Ahora el proceso puede acceder a la página como si siempre hubiera estado en la memoria.

4.2 Paginación por demanda.

- Paginación por demanda:



4.2 Paginación por demanda.

- Desempeño de la paginación por demanda:
  - Cuando ejecutamos una instrucción:
    - Puede generar más de un fallo de página (una página para la instrucción y muchas para los datos).
    - Problema: **El rendimiento disminuye considerablemente al aumentar el número de fallos de página**, ya que el tiempo de acceso aumenta también frenando la ejecución de los procesos.

    - El tiempo de acceso a la memoria “tam”: (10, 200) ns.
    - **Si no ocurren fallos de página** el tiempo de acceso efectivo “taef”: **taef = tam**.
    - **Si ocurren fallos de página, primero leemos del disco la página requerida y luego accedemos a la información deseada.**

4.2 Paginación por demanda.

- Reemplazo de páginas:
  - Para su implementación es necesario desarrollar:
    - **Algoritmo de asignación de marcos:**
      - Si tenemos varios procesos en memoria es necesario decidir cuantos marcos se asignará a cada uno.
    - **Algoritmo de reemplazo de páginas:**
      - Proceso básico:
        - 1. Atender la interrupción de fallo de página.
        - 2. Traer la página a memoria y colocarla en un marco libre.
        - 3. Reiniciar el proceso.
    - ¿Y si no hay marcos libres? ⇒ Reemplazo de páginas.

4.2 Paginación por demanda.

- Reemplazo de páginas:
  - El reemplazo de páginas actuará del siguiente modo:
    - **Si no hay marcos libres buscamos uno que no se esté usando y lo liberamos** ⇒ el marco liberado puede contener la página por la que ocurrió el fallo.
    - La **secuencia de servicio de fallos de página** será ahora:
      - 1. Encontrar la página deseada en el disco.
      - 2. Hallar un marco libre:
        - A) Si hay un marco libre, usarlo.
        - B) Si no lo hay ⇒ usar un algoritmo de reemplazo para escoger un marco.
        - C) Escribir la página "víctima" en el disco; modificar la tabla de páginas y la tabla de marcos.
      - 3. Leer la página deseada del disco y colocarla en el marco recién liberado; modificar la tdp y la tdm.
      - 4. Finalmente, reiniciar el proceso de usuario.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Objetivo fundamental:** *"Buscar el algoritmo con la frecuencia de fallos de página más baja posible".*
  - **Evaluamos un algoritmo** ejecutándolo con una serie específica de referencias y calculando el número de fallos de página.
  - Serie (cadena) de referencias:
    - Se utilizan para evaluar la calidad de los algoritmos de reemplazo.
    - La serie de referencia se puede obtener:
      - Generador de números aleatorios.
      - Rastreando la ejecución y grabando una traza de ejecución.
  - Para determinar el nº de fallos de página de un algoritmo **es necesario conocer el nº de marcos disponibles.**

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmo FIFO (First In – First Out):**
    - *Sustituimos la página residente que lleve más tiempo en la memoria.*
    - Es fácil de implementar (como una cola FIFO de páginas).
    - Su desempeño no siempre es bueno, puesto que la página sustituida podría contener, por ejemplo, una variable muy utilizada desde el principio y que se usa constantemente.
    - Observamos una **"Anomalía de Belady":** la frecuencia de fallos de página podría aumentar al aumentar el nº de marcos asignados.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmo Óptimo:**
    - También llamado algoritmo mínimo.
    - Tiene la **frecuencia de fallos de página más baja de entre todos los algoritmos** y no presenta Anomalía de Belady.
    - *Escoge como víctima la página que más tardará en ser usada de nuevo.*
    - **Difícil de implementar** (requiere presciencia), es decir, un conocimiento futuro de la serie de referencias.
    - Se utiliza para efectuar comparaciones.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmo LRU (Least Recently Used):**
    - Aproximación implementable del algoritmo óptimo.
    - *Se sustituye la página "menos recientemente usada". Se recuerda el instante en que cada página se usó por última vez, y en caso de reemplazo se escoge la página que tiene más tiempo sin usarse.*
    - **Se utiliza mucho y se considera muy bueno** (alto rendimiento respecto del óptimo).
    - Requiere un hardware adicional para su implementación:
      - Contadores: Reemplazo de la página con un tiempo más largo.
      - Pila: La base de la pila corresponde con la página LRU.
      - Esta implementación resulta costosa, ya que contadores y pilas deben actualizarse en cada referencia a la memoria ⇒ acceso más lento a la memoria.
    - **Sistemas reales: implementan aproximaciones a LRU.**
    - No presentan la Anomalía de Belady.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmos de aproximación al LRU:**
    - **1. Algoritmos con bits de referencia adicionales:**
      - Las técnicas basadas en LRU utilizan un bit de referencia puesto por el hardware.
      - El hardware enciende el bit de referencia (lo pone a 1) de una página cada vez que se hace referencia a ella (lectura o escritura).
      - Examinando este bit no conocemos el orden de uso, pero sí sabemos cuáles páginas se usaron y cuáles no.
      - Es posible obtener información de ordenamiento adicional si registramos los bits de referencia a intervalos adicionales.
      - Byte histórico:
        - Por ej.: 11000100 se usó más recientemente que 01110111.
        - LRU: página con el número más bajo.
      - Si el nº de bits históricos es cero, es decir, dejamos sólo el bit de referencia ⇒ Algoritmo de segunda oportunidad.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmos de aproximación al LRU:**
    - **2. Algoritmo de segunda oportunidad o algoritmo del reloj.**
      - Sencillo y efectivo, *examina la página más antigua como posible víctima.*
      - Comportamiento: FIFO teniendo en cuenta el bit de referencia:
        - Bit de referencia a cero: Reemplazo de página.
        - Bit de referencia a uno: Segunda oportunidad, ponemos el bit de referencia a cero y seleccionamos la siguiente página FIFO.
      - Se puede implementar como una cola circular, donde un puntero indicará cuál es la página a reemplazar a continuación.
      - Cuando se necesita un marco, el puntero avanza hasta encontrar una página cuyo bit de referencia está apagado. Con el avance del puntero los bits de referencia encendidos se van apagando.
      - Una vez hallada una página víctima, se reemplaza y la nueva página se inserta en la cola circular en esa posición.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmos de aproximación al LRU:**
    - **3. Algoritmo de segunda oportunidad mejorado**.
      - Bit de referencia + bit de modificación.
      - Usando estos dos bits tenemos cuatro situaciones posibles:
        - (0, 0): No se ha usado ni modificado recientemente.
        - (0, 1): No se ha usado recientemente, sí se ha modificado.
        - (1, 0): Usada recientemente, no modificada.
        - (1, 1): Usada y modificada recientemente.
      - *Se reemplaza la página que encontremos de clase más baja.*
      - Se da preferencia a las páginas que han sido modificadas a fin de reducir el nº de operaciones de E/S requeridas.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmos de conteo:**
    - Tienen un contador con el nº de referencias que se hacen a cada página.
    - Dos esquemas:
      - **Algoritmo LFU (Least Frequently Used):**
        - *Reemplaza la página menos frecuentemente usada (cuenta más baja).*
        - Problema: páginas que se usaron mucho durante la fase inicial del proceso y luego no se vuelven a usar.
        - Solución: desplazar las cuentas un bit a la derecha a intervalos regulares.
        - Problema serio: páginas traídas recientemente, alta probabilidad de salir (cuenta baja).
      - **Algoritmo MFU (Most Frequently Used):**
        - *Reemplaza la página más frecuentemente usada.*
        - Problema: se pueden mantener páginas viejas a las que no se accede.

4.2 Paginación por demanda.

- Algoritmos de reemplazo de páginas:
  - **Algoritmos de colocación de páginas en buffers:**
    - **Basados en la idea de mantener una reserva de marcos libres.**
    - Cuando ocurre un fallo de página:
      - *La página deseada se coloca en un marco libre de la reserva antes de escribir la víctima en el disco.*
    - Este procedimiento permite al proceso reiniciarse lo más pronto posible, sin esperar a que la página víctima se escriba en el disco.
    - *Cuando la página víctima termina de escribirse en el disco, su marco se añade a la reserva de marcos libres.*

4.2 Paginación por demanda.

- Asignación de marcos:
  - **Objetivo fundamental:** *Tratar de repartir la cantidad de memoria libre entre los distintos procesos.*
  - **Reserva de marcos:**
    - Definir un sistema de reparto de marcos a los procesos en ejecución.
    - Todo proceso debería tener una reserva mínima de marcos.
    - **¿Cómo asignar los marcos a los procesos?**
      - 1. Número mínimo de marcos.
      - 2. Algoritmos de asignación; reparto proporcional (por tamaño, por prioridad).
      - 3. Asignación global o local.

4.2 Paginación por demanda.

- Asignación de marcos:
  - **Número mínimo de marcos:**
    - **A medida que disminuye el nº de marcos asignados a un proceso la frecuencia de fallos de página aumenta**, lo que hace más lenta la ejecución de los procesos.
    - El nº mínimo de marcos que es necesario asignar viene dado por la arquitectura del sistema. Cuando ocurre un fallo de página, antes de haber terminado la ejecución de una instrucción, está deberá reiniciarse. Por lo tanto será necesario tener suficientes marcos para contener todas las páginas a las que una sola instrucción puede hacer referencia.
    - **El máximo nº de marcos está limitado por la cantidad de memoria física disponible.**

4.2 Paginación por demanda.

- Asignación de marcos:
  - **Algoritmos de asignación:**
    - **Reparto equitativo:** La forma más fácil de dividir m marcos entre n procesos es dar a cada uno una porción equitativa, m/n marcos.
    - **Reparto proporcional:** Diferentes procesos requieren diferentes cantidades de memoria ⇒ asignamos memoria disponible a cada proceso según su tamaño.
    - **Reparto por prioridad:** Concedemos más memoria a un proceso de más alta prioridad con el fin de acelerar su ejecución.
    - **Estrategia a seguir:** *esquema de asignación proporcional teniendo en cuenta, el tamaño del proceso y su nivel de prioridad.*

4.2 Paginación por demanda.

- Asignación de marcos:
  - **Asignación global o local:**
    - Varios procesos compiten por los marcos.
    - **Reemplazo global:** Un proceso selecciona un marco de reemplazo de entre el conjunto de todos los marcos (incluso un proceso puede arrebatarle un marco a otro).
    - **Reemplazo local:** Un proceso sólo puede seleccionar un marco de entre los que le fueron asignados.
    - El reemplazo global permite a un proceso prioritario aumentar su asignación de marcos a expensas de procesos de baja prioridad.
    - **El reemplazo global generalmente aumenta el rendimiento de un sistema y es el método más utilizado.**

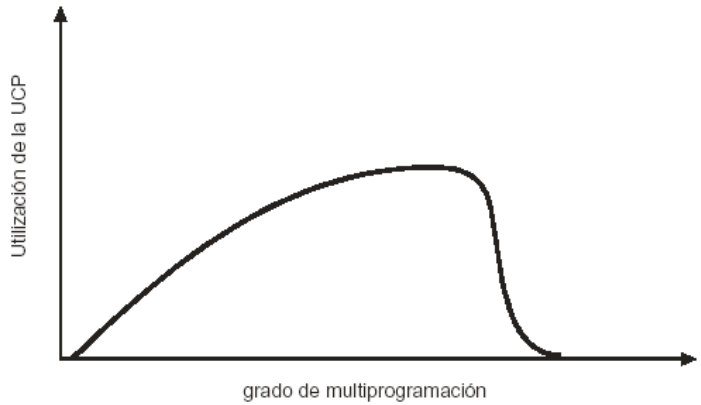
4.3 Hiperpaginación y working set.

- Hiperpaginación:
  - Cuando el **aprovechamiento es bajo se aumenta el grado de multiprogramación.**
  - Es posible reducir el nº de marcos asignados al mínimo, aunque hay cierto nº de páginas que están en servicio activo.
  - **Si el proceso no tiene suficientes marcos para estas páginas, pronto causa un fallo de página.**
  - En ese momento el proceso deberá reemplazar alguna página y, dado que todas sus páginas están en uso activo, deberá reemplazar una que volverá a necesitar de inmediato.

  - **Resultado:** Se genera rápidamente otro fallo de página y otro y otro.
  - **Consecuencia:** Disminuye el aprovechamiento de la CPU y entonces el planificador de CPU aumenta el grado de multiprogramación, agravando aún más el problema.

4.3 Hiperpaginación y working set.

- Hiperpaginación:
  - Si el grado de multiprogramación es excesivo, el sistema puede estar **más tiempo paginando que haciendo trabajo productivo**. Esta actividad se conoce como **hiperpaginación.**
  - ¿Cómo evitarla?:
    - Políticas de reemplazo local.
      - La hiperpaginación de un proceso puede afectar al resto.
    - Concediendo memoria según las necesidades reales (localidades, working set, ...)



4.3 Hiperpaginación y working set.

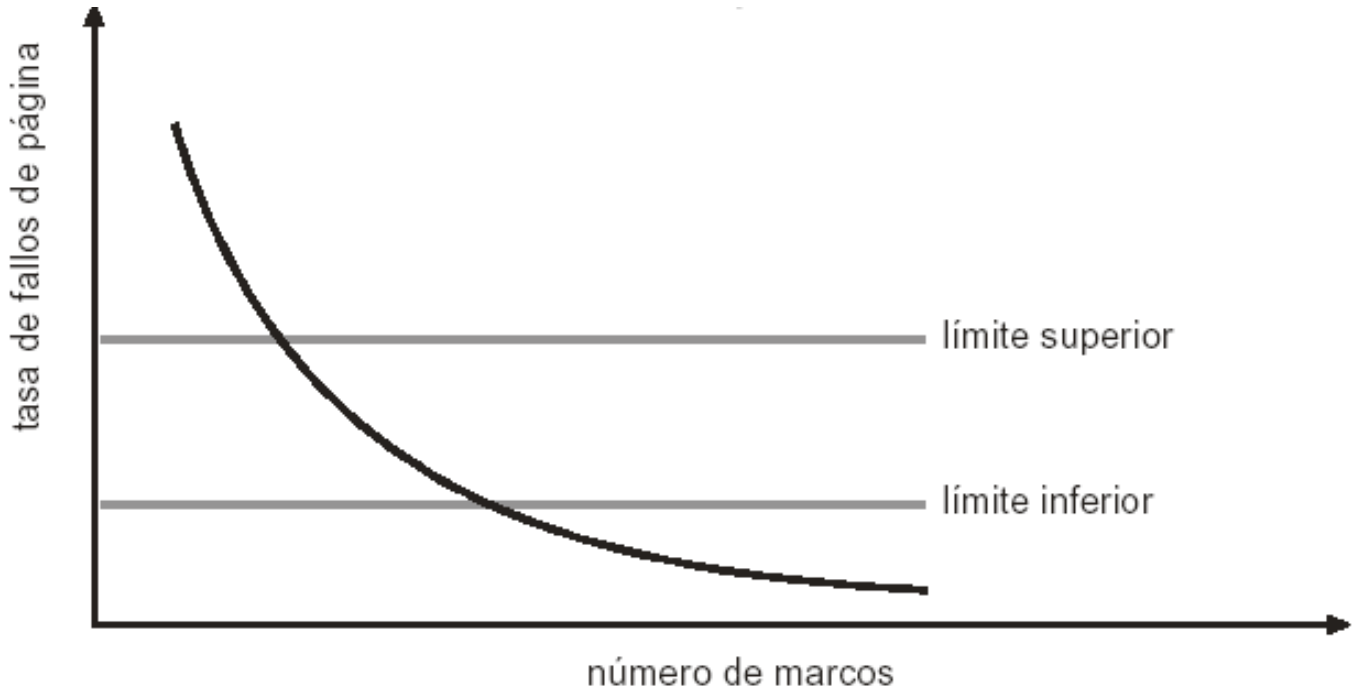
- Modelo del conjunto de trabajo (working set):
  - Se observa que todo proceso trabaja en cada momento con unas zonas de código y datos bien delimitadas: **localidad**.
  - Cuando se salta a otra subrutina, etc., se cambia la localidad.
  - Si un proceso tiene asignada su localidad en MP, no ocasiona fallos de página.
  - Concepto que trata de aproximarse a la localidad actual de un proceso (working set o área activa).
  - **Es el conjunto de páginas con el que ha trabajado un proceso**

4.3 Hiperpaginación y working set.

- Frecuencia de fallos de página:
  - **Hiperpaginación:** *Frecuencia de fallos de página elevada ⇒ nos interesa controlar la frecuencia de fallos de página.*
  - La frecuencia de fallos está muy ligada con el nº de marcos:
    - **Frecuencias altas ⇒ el proceso necesita más marcos.**
    - **Frecuencias bajas ⇒ al proceso le sobran marcos.**
  - **Los marcos de página se repartirán** entre los procesos que tengan fallos de página muy frecuentes.
  - **Podemos establecer límites** superiores e inferiores para la frecuencia de fallos de página deseada.

4.3 Hiperpaginación y working set.

- Frecuencia de fallos de página:



4.4 Otras consideraciones acerca de la MV.

- Prepaginación:
  - *Cuando se reinicia un proceso que se había intercambiado a disco todas sus páginas están en el disco ⇒ habrá un fallo de página para traer a la memoria cada una de ellas.*
  - **La prepaginación trata de evitar este alto nivel de paginación.**
  - **Se trae de una sola vez** a la memoria todas las páginas que se necesitan.
  - **Ventaja:**
    - Si la predicción es buena el tiempo de ejecución de los procesos se reduce notablemente.

4.4 Otras consideraciones acerca de la MV.

- Discusión del tamaño de página:
  - Debemos de definir el tamaño adecuado de página siempre que se diseña una página nueva (512 bytes, 16384 bytes).
  - **A favor de páginas grandes:**
    - Tablas de páginas de los procesos.
    - Minimización de los tiempos de E/S (búsqueda, transferencia,...).
    - Incrementos de capacidades de CPU y memoria por encima de la velocidad del disco (disminuyendo el nº de fallos de página).
  - **A favor de páginas pequeñas:**
    - Aprovechamiento de memoria (menor fragmentación interna).
    - Menos E/S, porque podemos aislar la memoria que realmente se necesita (menos memoria asignada total).

4.4 Otras consideraciones acerca de la MV.

- Estructura del programa:
  - **Las estructuras de datos y de programación:**
    - Pueden aumentar la localidad.
    - Reducen la frecuencia de fallos de página y el nº de páginas del conjunto de trabajo.
    - Una pila tiene una alta localidad frente a una tabla de dispersión que tiene una baja localidad.
    - **El lenguaje de programación también afecta a la paginación.**
      - El abuso de punteros: baja localidad.
      - Pocos punteros (Pascal): Mejor localidad de referencia y se ejecuta más rápidamente.

4.4 Otras consideraciones acerca de la MV.

- Interbloqueo de E/S:
  - *Con paginación por demanda a veces es necesario permitir que se bloqueen o fijen algunas páginas en la memoria.*
  - Se asocia un bit de bloqueo a cada marco.
  - Si el marco está bloqueado no se le podrá escoger para reemplazarlo.
  - Las páginas bloqueadas no podrán reemplazarse.
  - *Una vez que termina la E/S las páginas se desbloquean.*

4.4 Otras consideraciones acerca de la MV.

- Segmentación por demanda:
  - **Aproximación a la paginación por demanda**, requiere menos hardware para implementarla.
  - El SO asigna la memoria en segmentos en lugar de páginas.
  - **Descriptores de segmento:** incluyen información acerca del tamaño, protección y ubicación del segmento.
    - **Bit de validez:** indica si el segmento está o no en la memoria:
      - Si el segmento está en la memoria: acceso.
      - Si no está: fallo de segmento, el SO intercambia el segmento para traer a la memoria el segmento deseado.
    - **Bit de acceso:** determina el segmento que se reemplazará. Funciona de manera similar al bit de referencia de la paginación por demanda.