INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

Memoría:

- La organización y la administración de la "memoria principal" resultan ser uno de los factores MÁS importantes en el diseño de un sistema operativo.
- Todo programa y datos de estos deben estar en el almacenamiento principal (RAM) para:
    - Poder ejecutar dichos programas.
    - Referenciarlos de forma directa.
- El kernel se encarga de administrar los espacios de direcciones de la memoria RAM.
    - Administración eficiente → implica mejor funcionamiento.

Tarea del sistema operativo:

- Debe llevar el registro de las partes de memoria que están en uso y aquellas las cuales no están en uso.
- Debe asignar espacio en memoria principal (RAM) a los procesos cuando estos necesitan el espacio.
- Debe liberar espacio de memoria asignada a procesos que han terminado.
- Se espera que el sistema operativo haga un uso eficiente de la memoría con la finalidad de alojar la mayor cantidad de procesos posible.
- Adicionalmente:
    - El SO debe lograr que el programador se **abstraiga** de la alocación de los programas.
    - Debe brindar **seguridad** entre procesos para que no haya accesos indebidos (un proceso violando el espacio privado de otro).
    - Brindar la posibilidad de **acceso compartido** a determinadas partes de la memoria (librerías, código en común, etc).
    - Garantizar la **performance**.

Administración de memoría

- Hay una división lógica de la memoría física para alojar múltiples procesos:
    - Se garantiza protección.
    - Depende del mecanismo de administración provisto por el Hardware.
    - ¿Cómo va a parar lo lógico a lo físico? → depende del hardware y de un conjunto de estructuras utilizadas por el mismo.

- Asignación eficiente:
    - Contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos.
    - Asignar de manera eficiente la RAM.

Requisitos para la administración de la RAM:

➔ Reubicación
- El programador no debe ocuparse de saber donde será colocado el programa en la RAM. Es decir, el programa es indiferente a la ubicación física.
- Mientras un proceso es ejecutado, puede ser sacado y traído a la memoria (swap), y posiblemente ser colocado en diferentes direcciones → el proceso debe ser independiente de la ubicación.
- Las referencias a la memoría se deben "traducir" según la ubicación actual del proceso.

➔ Protección
- Los procesos NO deben referenciar o acceder a direcciones de memoria de otros procesos. (SALVO si tienen permiso para ello).
- El chequeo se debe realizar durante la ejecución.
    - No es posible anticipar todas las referencias a memoria que un proceso puede realizar (es dinámico), por eso se realiza en run-time.

➔ Compartición
- Consiste en permitir que varios procesos accedan a la misma porción de memoría.
    - Ej: rutinas comunes, librerías, espacios explícitamente compartidos para los procesos, etc.
- Permite un mejor aprovechamiento de la memoria RAM, evitando copias repetidas de instrucciones que son innecesarias.

Abstracción – espacio de direcciones

- El espacio de direcciones es el rango de direcciones a memoria posibles que un proceso puede utilizar para direccionar sus instrucciones y datos.
- El rango de direcciones lógicas esta definida por la arquitectura.
    - Procesador de 32 bits: 0.. 2^{32} -1
    - Procesador de 64 bits: 0.. 2^{64} -1
- El direccionamiento es independiente de la ubicación "real" del proceso en la memoria RAM, **la dirección 100 en el rango puede no ser igual a la dirección 100 en la física**.

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 3: MEMORÍA

Direcciones

- Lógicas (la que usan los procesos)
  - Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria.
  - Representa una dirección en el "Espacio de direcciones del proceso"
- Físicas
  - Referencia a una localidad en la memoria física (RAM).
  - Dirección absoluta.
- **Cuando uso direcciones lógicas, necesito algún tipo de conversión a direcciones físicas para hallar el dato.**

Conversión de direcciones

➔ Cada dirección lógica que se genera se debe convertir a una dirección física.

➔ Una forma simple de hacer esto es utilizando registros auxiliares.
  - Registro base:
    - Dirección de comienzo del espacio de direcciones del proceso en la RAM.
  - Registro límite:
    - Dirección final del proceso o tamaño de su espacio de direcciones.
  - A partir de los valores de esos dos registros, podemos tomar la dirección lógica como un desplazamiento desde la base, y controlar que dicha suma (registro base + dirección lógica=dirección física) no exceda el registro límite. Si la dirección supera el límite → excepción.

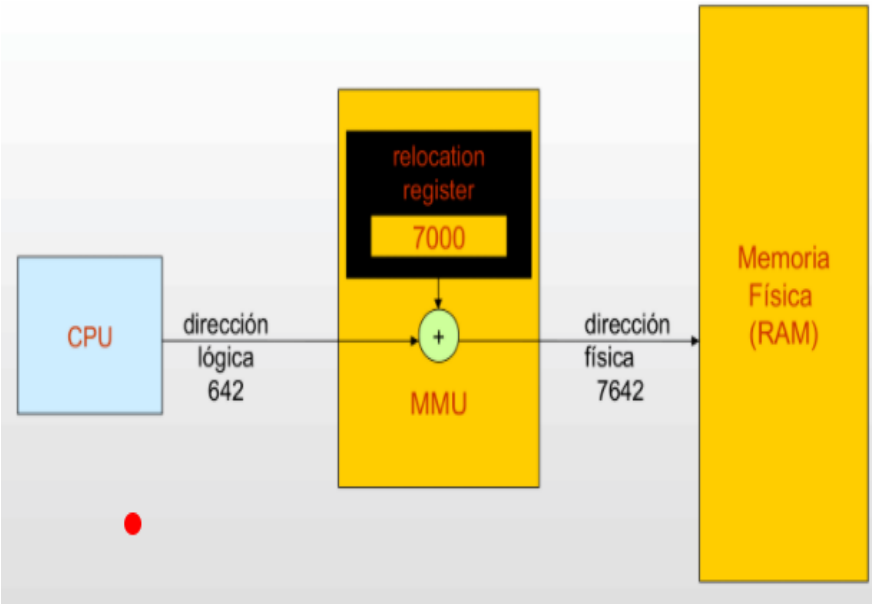
➔ Los valores de ambos registros se fijan cuando el espacio de direcciones del proceso se carga en la memoria.

Direcciones lógicas vs Físicas

- Si la CPU trabaja con direcciones lógicas, para acceder a la memoria principal, se deben transformar en direcciones físicas.
  - Resolución de direcciones (address-binding): transformar la dirección lógica en la dirección física que corresponde.
- Resolución en momento de compilación o en tiempo de carga del proceso (no usado en la actualidad):

- Direcciones lógicas y físicas son idénticas.
- Para reubicar un proceso es necesario recompilarlo o recargarlo.

- Resolución en tiempo de ejecución
  - Direcciones lógicas y físicas son diferentes.
  - Las lógicas son llamadas direcciones virtuales.
  - La reubicación se puede realizar fácilmente.
  - La resolución debe ser rápida en velocidad, a la par del CPU.
  - El mapeo entre "virtuales" y "físicas" es realizado por hardware.
    - Memory Management Unit (MMU).
      - Es un dispositivo de hardware que mapea direcciones virtuales a físicas (convierte.)
        - Es parte del procesador. (Hardware)
        - Re-programar el MMU es una operación privilegiada.
          - SOLO SE REALIZA EN MODO KERNEL.
      - El valor en el "registro de realocación" es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria.
        - Los procesos nunca usan direcciones físicas.



Mecanismos de asignación de memoria:

➔ Particiones fijas: el primer esquema que se implementó.

- La memoria se divide en particiones o regiones de tamaño fijo (todos del mismo tamaño o no).
- Cada partición aloja un proceso.
- Cada proceso se coloca de acuerdo a algún criterio (first fit, best fit, worst fit, next fit) en alguna partición (son algoritmos*).*

- Al ser fijo podría ocurrir fragmentación interna (espacio desperdiciado).

➔ Particiones dinámicas: es la evolución del esquema anterior.

- Las particiones varían en tamaño y en número.
- Cada una aloja un proceso.
- Cada partición es generada de forma dinámica con el tamaño justo que necesita el proceso.
- Generación fragmentación externa.
- Se recomienda peor ajuste, ya que la que me va a quedar libre es probable que sea la mas grande.

➔ Fragmentación (problemas de las particiones)

- Producida cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua.
- Fragmentación interna:
  - Producido en particiones fijas.
  - Porción de partición que queda sin utilizar.
- Fragmentación externa:
  - Producido en particiones dinámicas.
  - Son huecos que van quedando en la memoria a medida que los procesos terminan.
  - Estos huecos si bien son memoria libre, al no estar contiguos entre sí un proceso no puede ser alocado.
  - El problema se soluciona compactando → pero es costosa.

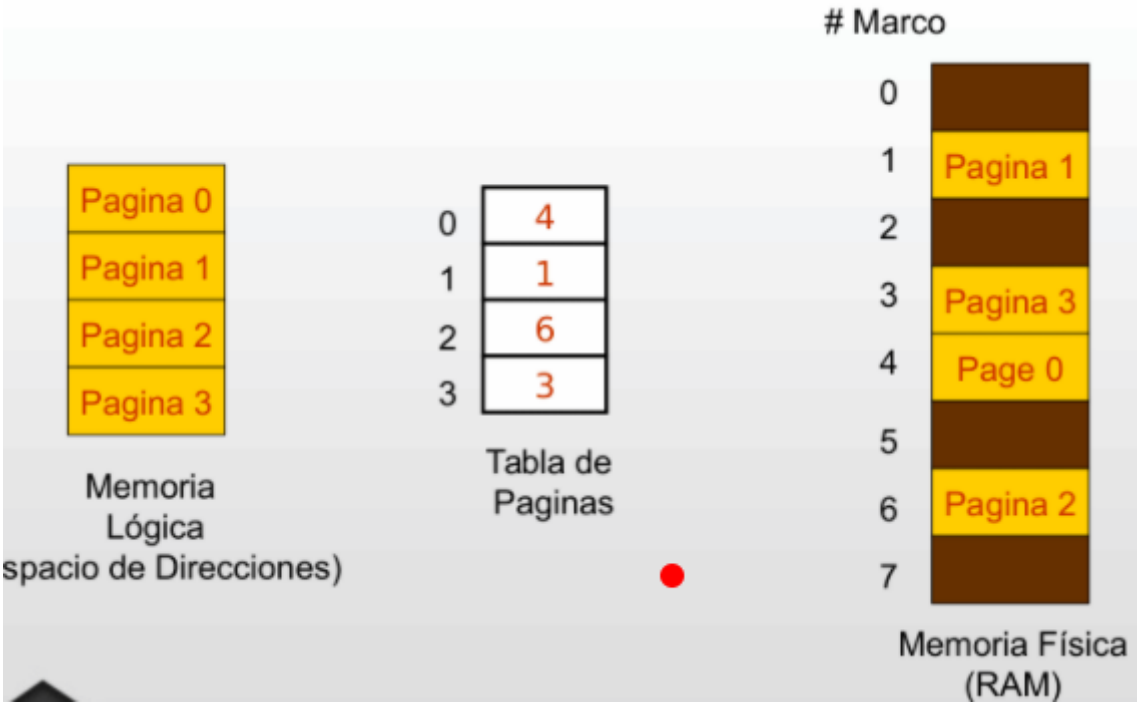
Problemas del esquema registro base y límite

- El esquema de registro base + registro límite presenta problemas:
  - Necesidad de almacenar el espacio de direcciones de forma **continua** en memoria física.
  - Los primeros SO definían particiones fijas de memoria hasta evolucionar en particiones dinámicas.
  - Fragmentación.
  - Mantener partes del proceso que sean innecesarias.
  - Los esquemas de particiones fijas y dinámicas NO son usados hoy en día.
- Solución:
  - Paginación

- Segmentación

Paginación

- ➔ La memoria física es dividida lógicamente en pequeños trozos de igual tamaño (marcos).
- ➔ La memoria lógica (de los procesos) es dividida en trozos de igual tamaño que los marcos (esto genera paginas).
- ➔ El SO debe mantener una tabla de paginas por cada proceso, donde cada entrada de dicha tabla contiene el marco en la que se coloca dicha página. Se guarda la base del marco que se cargo en la PCB. Hay puntero a la tabla.
- ➔ La dirección lógica es interpretada como:
  - Un numero de pagina y un desplazamiento dentro de la misma.



- ➔
- ➔ Como se puede observar, se rompe la continuidad.
- ➔ Puede causar fragmentación interna (menos rompe bolas que la externa).
- ➔ No es muy importante esa fragmentación interna.

Segmentación

- ➔ Esquema que se asemeja a la visión del usuario. El programa es dividido en partes/secciones, donde en cada sección se guardan datos similares.
- ➔ Un programa es una colección de segmentos. Un segmento es una unidad lógica como:
  - Programa Ppal, procedures, funciones, variables locales y globales, stack, etc.

- Cada segmento tiene un registro base y un registro límite.
- Por lo que se genera una tabla de segmentos con esos dos valores para cada segmento por proceso.

➔ Puede causar fragmentación EXTERNA.

➔ Todos los segmentos de un programa pueden NO tener el mismo tamaño (códigos, datos, rutinas). La base y límite del segmento son dinámicos.

➔ Las direcciones lógicas consisten en 2 partes:

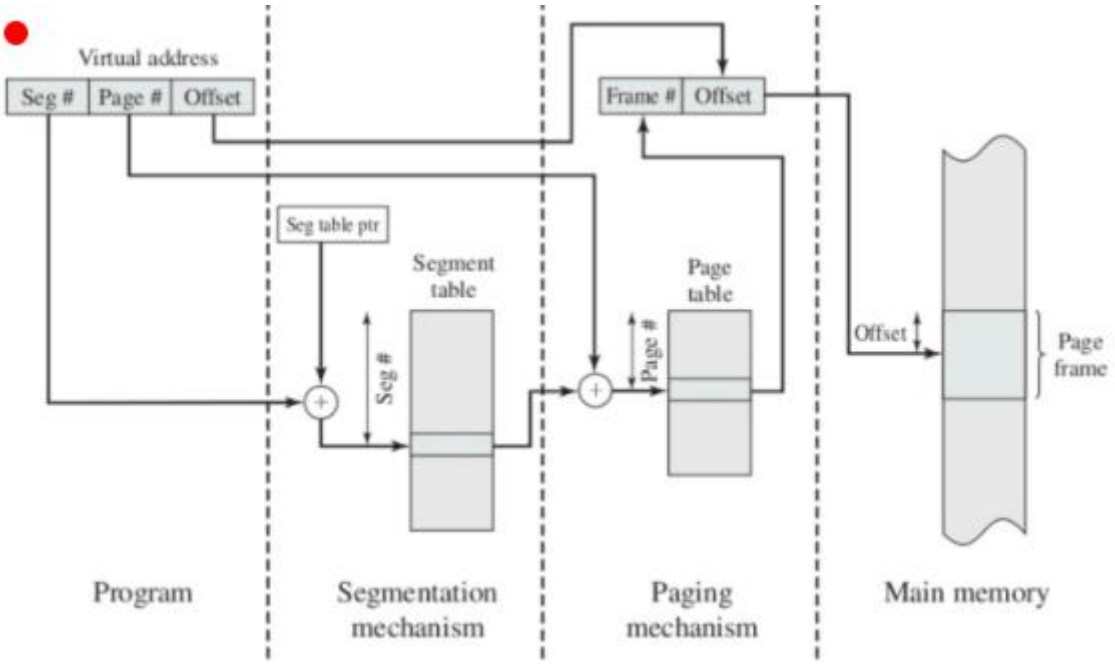
- Selector de segmento.
- Desplazamiento dentro del segmento (sobre registro base y límite).

➔ La segmentación posee ventaja sobre la paginación respecto de: la protección de espacios de memoria y la compartición de bloques de memoria.

➔ Cuando uno compila → el compilador deja huellas del segmento.

Segmentación paginada (mix de las dos anteriores):

- Paginación
  - Transparente al programador.
  - Elimina fragmentación externa.
- Segmentación
  - Visible al programador.
  - Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección.
- Segmentación paginada: cada segmento es dividido en páginas de tamaño fijo.
  - Cada segmento tiene una tabla de páginas que le corresponde.
  - Toma las ventajas de ambas: compartición, protección(de parte de la segmentación), evitar fragmentaciones (de parte de la paginación).
  - Se sigue guardando en paginas
  - La unidad de trabajo para subir o bajar de la RAM es la pagina


Virtual address
Seg #
Page #
Offset
Frame #
Offset
Seg table ptr
Segment table
Page table
Seg #
Page #
Offset
Page frame
Program
Segmentation mechanism
Paging mechanism
Main memory

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

Motivación para Memoria Virtual

- Podemos pensar también que no todo el espacio de direcciones del proceso se necesita en todo momento.
    - Rutinas o librerías que se ejecutan una única vez (o nunca).
    - Partes del programa que no vuelven a ejecutarse.
    - Regiones de memoria alocadas dinámicamente y luego liberadas.
- No hay necesidad que la totalidad de la imagen del proceso sea cargada en memoria.

Como lo podemos trabajar…

- El SO puede traer a memoria las "piezas" de un proceso a medida que éste las necesita.
- Definiremos como "**Conjunto residente"** a la porción del espacio de direcciones del proceso que se encuentra en memoria RAM. (también como working set)
- Con el apoyo del HW:
    - Se detecta cuando se necesita una porción del proceso que no está en su Conjunto Residente.
    - Se debe cargar en memoria dicha porción para continuar con la ejecución.
- Ventajas:
    - Más procesos pueden ser mantenidos en memoria.
        - Solo se cargan algunas secciones de cada proceso.
        - Con más procesos en memoria RAM es más probable que existan más procesos Ready.
    - Un proceso puede ser mas grande que la memoria RAM (la cantidad requerida de RAM para el proceso total se excede).
        - El usuario no se debe preocupar por el tamaño de sus programas.
        - La limitación la impone el HW y el bus de direcciones.

¿Qué se necesita para aplicar la técnica de Memoria Virtual?

- El hardware debe soportar paginación por demanda (y/o segmentación por demanda),es decir que se meten paginas a medida que se necesiten.
- Un dispositivo de memoria secundaria (disco) que dé el apoyo para almacenar las secciones del proceso que no están en memoria principal (área de intercambio - swap).
- El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria.

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

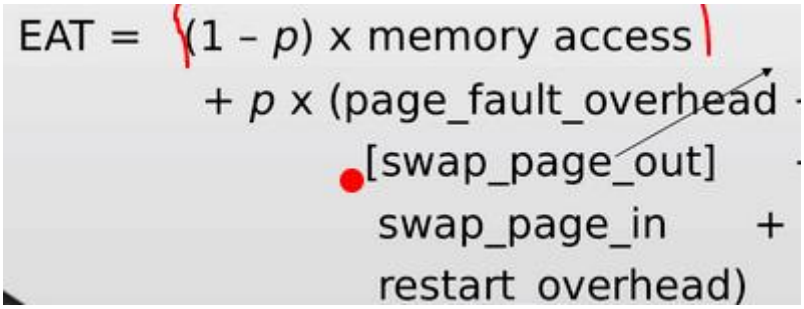
Memoria virtual con paginación

- ➔ Cada proceso tiene su tabla de paginas.
- ➔ Cada entrada en la tabla referencia al frame o marco en el que se encuentra la página en RAM.
- ➔ Cada entrada en la tabla de páginas tiene bits de control:
    - Bit V: indica si la página está en memoria. (cargada), el SO modifica el valor de este bit ya que es el encargado de subir o bajar paginas.
    - Bit M: indica si la página fue modificada. Si se modificó, en algún momento se deben reflejar los cambios en Memoria Secundaria (disco) para mantener una coherencia de datos. El bit modificado lo va cargando el hardware en sí (ya que este sabe sobre operaciones de escritura), el bit lo usa el SO.

Fallo de páginas(page fault)

- Ocurre cuando el proceso intenta usar una dirección que está en una página que no se encuentra en RAM. Bit V = 0.
    - La pagina no se encuentra en su conjunto residente.
- El HW detecta la situación y genera un trap al S.O
- El S.O podrá colocar al proceso en estado de “Blocked” (espera) mientras gestiona que la pagina que se necesite se cargue.
    - La carga consiste en:
        - El S.O busca un marco libre en la memoria y genera una operación de E/S al disco para copiar en dicho marco la página del proceso que se necesita usar.
        - El SO puede asignarle la CPU a otro proceso mientras se completa la E/S
            - La E/S avisará por interrupción cuando finalize.
    - Cuando la E/S finaliza, se notifica al SO y este:
        - Actualiza la tabla de páginas del proceso.
            - Coloca el Bit V en 1 en la pagina correspondiente.
            - Coloca la dirección base del frame donde se colocó la pagina.
        - El proceso que generó el fallo de página vuelve al estado de Ready.
        - Cuando el proceso, se ejecute se volverá a ejecutar la instrucción que provocó el fallo.
- **PERFORMANCE:**
    - Si los page faults son excesivos, la performance del sistema decae.
    - Tasa de page faults entre 0 y 1. $0 <= p <= 1$

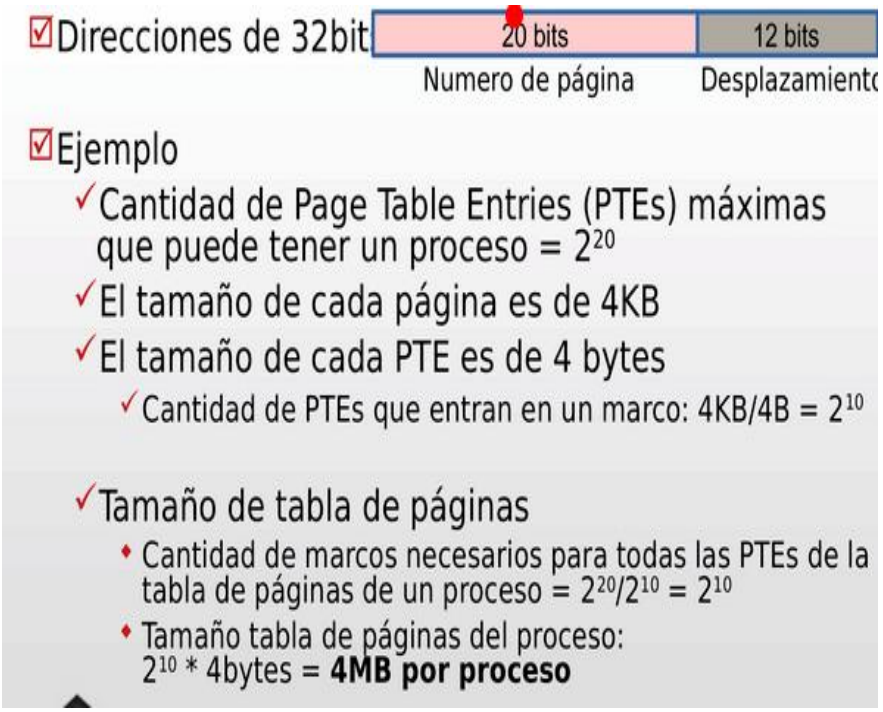
- Si p=0 no hay page faults.
- Si p=1, en cada acceso a memoria hay un page fault (caos).
- P tiene que tender a 0, nunca va a ser 0 por que la técnica se basa en el fallo de pagina.

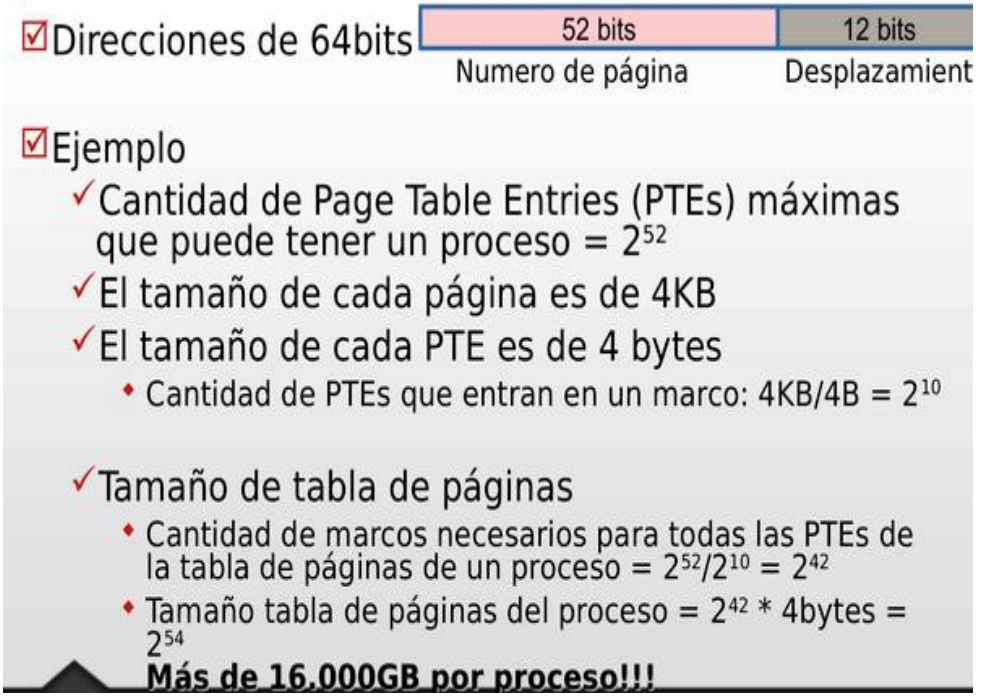
- Effective Access time (EAT) consiste en el tiempo efectivo de acceso:
  - 

  - Si no hay un frame libre, habrá que descargar otro para lograr espacio para la nueva pagina (swap_page_out)

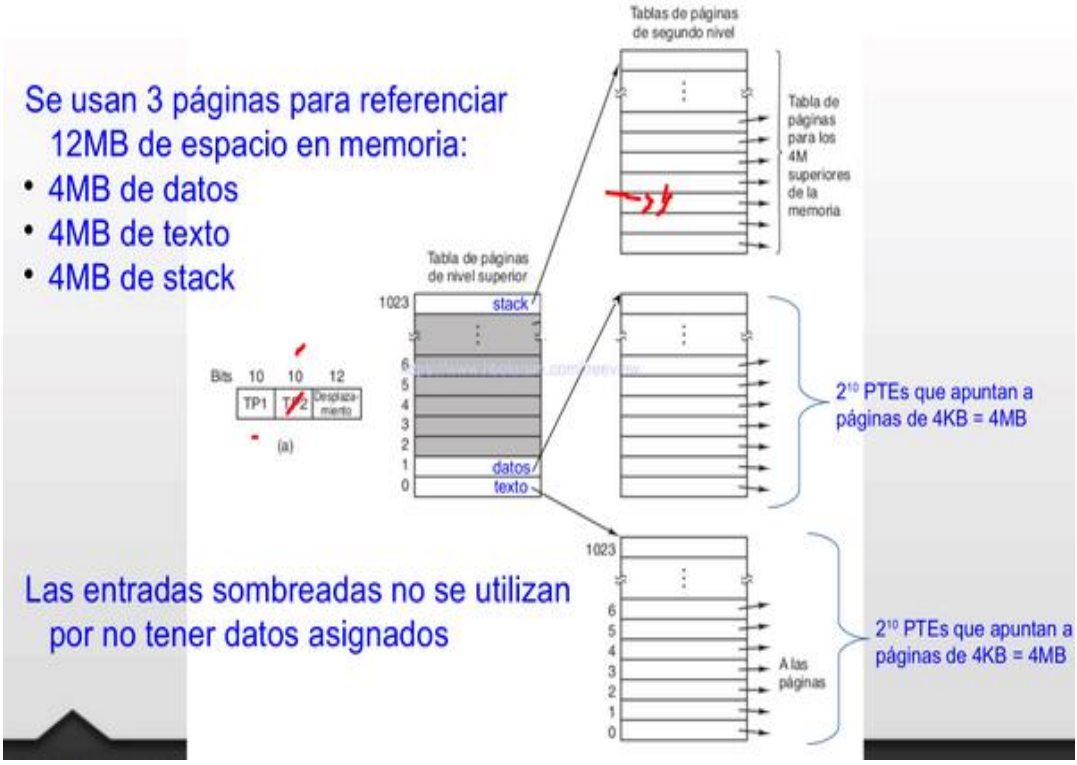
Tabla de paginas

- Cada proceso tiene su tabla de páginas.
- El tamaño de la tabla de páginas depende del espacio de direcciones del proceso.
- Puede alcanzar un tamaño considerable.
- Formas de organizar:
  - Tabla de 1 nivel: tabla única lineal
  - Tabla de 2 niveles: o más, multinivel
  - Tabla invertida: Hashing
- La forma de organizar la tabla de paginas depende del HW subyacente.

- 

- 

  - Eso para tablas de 1 nivel.
  - Existen las tablas de 2 niveles.



-
- El propósito de la tabla de páginas multinivel es dividir la tabla de páginas lineal en múltiples tablas de páginas.
- Cada tabla de paginas suele tener el mismo tamaño pero se busca que tengan un menor numero de paginas por tabla.
- La idea general es que cada tabla sea más pequeña.
- Se busca que la tabla de paginas no ocupe demasiada RAM.
- Además solo se carga una parcialidad de la tabla de paginas (la que se necesite usar).
- Existe un esquema de direccionamientos indirectos.
- Beneficios?
  - Las tablas de segundo nivel (o más) se pueden llevar a memoria secundaria, liberando RAM.
- Desventaja?
  - Más de un acceso a memoria para obtener un dato.
- ----------

- **Tabla invertida**:
  - Utilizada en arquitecturas donde el espacio de direcciones es muy grande.
    - Las tablas de paginas ocuparían muchos niveles y la traducción es costosa.
    - Por esa razón se usa ésta técnica.

        - Por ejemplo, si el espacio de direcciones es de 2^64 bytes, con páginas de 4KB, necesitamos una tabla de páginas con 2^52 entradas.
        - Si cada entrada es de 8 bytes, la tabla es de más de 30 millones de Gigabytes.
    - Hay una entrada por cada marco de página en la memoria real. Es la visión inversa a la que veníamos viendo.
    - Hay una sola tabla para todo el sistema.
    - El espacio de direcciones de la tabla se refiere al espacio físico de la RAM (todo lo que esté cargado), en vez del espacio de direcciones virtuales de un proceso.
    - El numero de página es transformado en un valor de HASH.
    - El HASH se usa como índice de la tabla invertida para encontrar el marco asociado.
    - Se define un mecanismo de encadenamiento para solucionar colisiones (el hash da igual para dos direcciones virtuales).
  - Solo se mantienen las entradas de tablas de pagina (PTE) de páginas presentes en memoria ram.
    - La tabla invertida se organiza como tabla hash en RAM.
      - Se busca indexadamente por número de página virtual.
      - Si está presente en tabla, se extrae el marco de página y sus protecciones.
      - Si no está presente en tabla, corresponde a un page fault.

Tamaño de la pagina

➔ Pequeño
  - Menor fragmentación interna (menos probabilidad de que queden espacios libres que sean muy grandes).
  - Más paginas requeridas por proceso → tablas de paginas más grandes.
  - Más paginas pueden residir en memoria.

➔ Grande
  - Mayor fragmentación interna.
  - La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente → es más rápido mover páginas hacia la memoria ram.

➔ Relación con la E/S

- Vel de transferencia: 2 MB/s
- Latencia: 8 ms
- Búsqueda: 20.
- Búsqueda y latencia tienen que ver con el cabezal del disco.

➔ Pagina de 512 bytes.

- 1 pagina → total: 28,2 ms
- Solo 0,2 ms de transferencia (1%) → transferencia real (sin búsqueda ni latencia).
- 2 paginas → 56,4ms

➔ Pagina de 1024 bytes

- Total: 28,4 ms
- Solo 0,4 ms de transferencia.

➔ Vemos que a mayor tamaño de página se transfiere mejor

Translation Lookasid Buffer (TLB)

- Cada referencia en el espacio virtual puede causar 2 (o más) accesos a la memoría física RAM.
  - Uno (o más) para obtener la entrada en tabla de páginas.
  - Uno para obtener los datos.
- Para solucionar este problema, una memoria cache de alta velocidad es usada para almacenar entradas de páginas.
  - TLB.
- Contiene las entradas de la tabla de páginas que fueron usadas más recientemente.
- Dada una dirección virtual, el procesador examina la TLB.
  - Si la entrada de la tabla de paginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física.
  - Si la entrada no es encontrada en la TLB (miss), el número de página es usado como índice en la tabla de paginas del procesos.
- Se controla si la pagina está en la memoria
  - Si no está, se genera un Page Fault.
- La TLB es actualizada para incluir la nueva entrada.
- El cambio de contexto genera la invalidación de las entradas de la TLB.

Asignación de Marcos:

- ¿Cuántas paginas de un proceso se pueden encontrar en memoria?
  - Tamaño del Conjunto Residente.
- Asignación dinámica
  - El número de marcos para cada proceso es variable, los procesos de alta prioridad pueden usar muchas páginas (y esto puede dejar que procesos con menos prioridad tengan menos páginas disponibles).
- Asignación fija:
  - Número fijo de marcos para cada proceso.
- Asignación equitativa: Ejemplo: si tengo 100 frames y 5 procesos, 20 frames para cada proceso, como desventaja: cada proceso puede requerir mas o menos frames, no es eficiente.
- Asignación proporcional: Asignado acorde al tamaño del proceso.
  - $s_i =$ size of process p_i
  - $S = \sum s_i$
  - $m =$ total number of frames
  - $a_i =$ allocation for $p_i = \frac{s_i}{S} \times m$

  $m = 64$
  $s_i = 10$
  $s_2 = 127$
  $a_1 = \frac{10}{137} \times 64 \approx 5$
  $a_2 = \frac{127}{137} \times 64 \approx 59$
- El SO hace juego entre asignación fija y dinámica.
  - Primero al proceso se le asigna un numero fijo de frames.
  - Si necesita más se le asigna más.

INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS – TEMA 2: MEMORIA

Reemplazo de páginas:

- ¿Qué sucede si ocurre un fallo de página y todos los marcos están ocupados → "Se debe seleccionar una pagina víctima" (la cual será sacrificada xd).
- ¿Cuál sería un reemplazo optimo?
  - Que la pagina a ser removida no sea referenciada en un futuro próximo.
  - Es perfecto, pero sólo teórico, ósea, inaplicable.
- Por eso el sistema operativo trata: la mayoría de los reemplazos predicen el comportamiento futuro mirando el comportamiento pasado.

Alcance del reemplazo

- Reemplazo global
  - El fallo de página de un proceso puede reemplazar la página de **cualquier** proceso.
  - El SO no controla la tasa de page-faults de cada proceso.
  - Puede tomar frames de otro proceso aumentando la cantidad de frames asignados a él.
  - Un proceso de alta prioridad podría tomar los frames de un proceso de menor prioridad.
- Reemplazo local
  - El fallo de página de un proceso solo puede reemplazar sus propias páginas – de su conjunto residente.
  - No cambia la cantidad de frames asignados.
  - El SO puede determinar cual es la tasa de page-faults de cada proceso.
  - Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos.

Algoritmos de reemplazo

- Optimo: es sólo teórico.
- FIFO: el más sencillo
- LRU (Least Recently Used): requiere soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas menos recientemente accedidas.

- 2da chance: un avance del FIFO tradicional que beneficia a las páginas más referenciadas.
- NRU (Non Recently Used):
    - Utiliza Bits R y M.
    - Favorece a las páginas que fueron usadas recientemente.