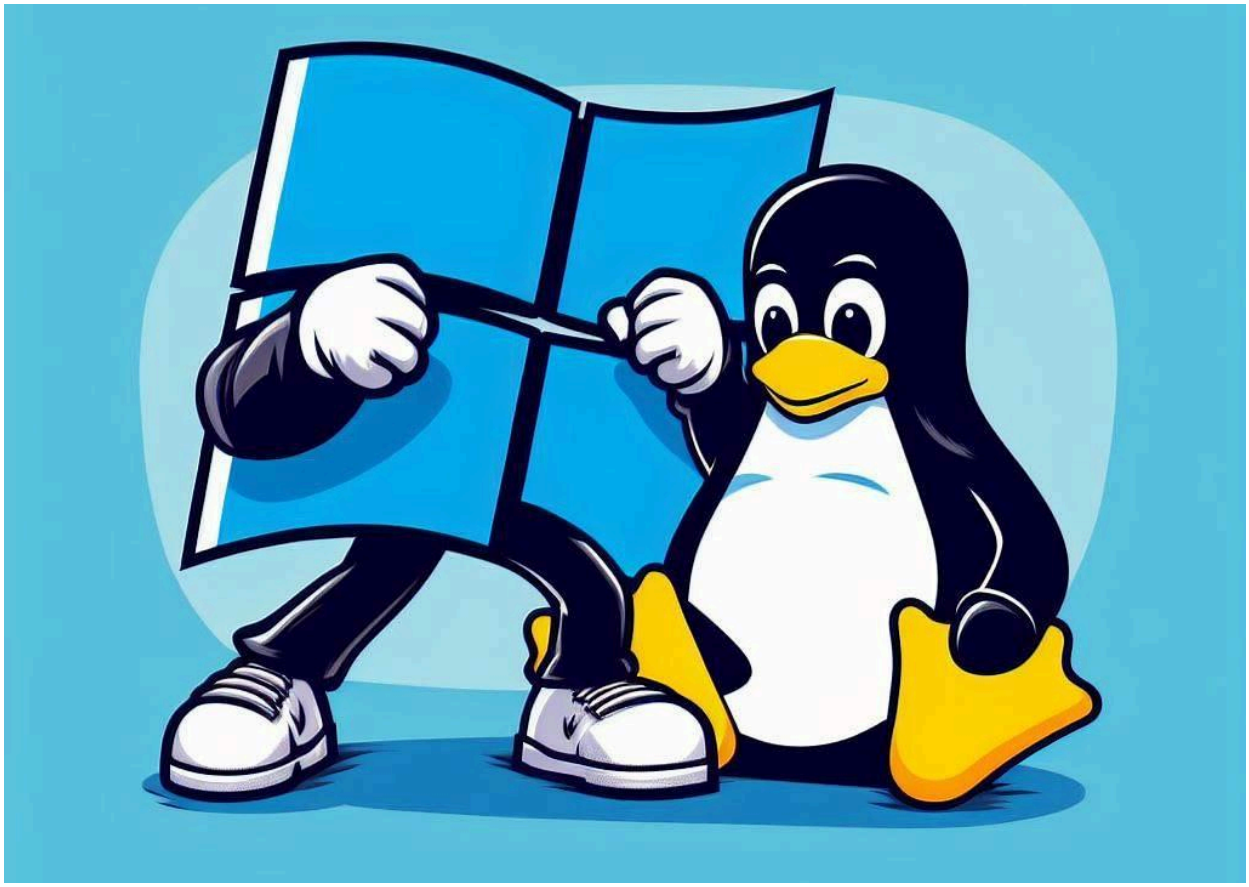
INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS

MEMORIA

Resumen teoría



Caporal Nicolás

Facultad de Informática

UNLP

2023

ÍNDICE

Clase 1

MEMORIA

La organización y administración de la memoria principal es uno de los factores más importantes en el diseño de los Sistemas Operativos.

Los programas y datos deben estar en memoria principal para:

- Poderlos ejecutar
- Referenciarlos directamente

TAREAS DEL SISTEMA OPERATIVO

En lo referido a memoria, el Sistema Operativo debe:

- Llevar un registro de las partes de memoria que se están utilizando y de aquellas que no. Y conocer el contenido (Si es código, si son datos, etc).
- Asignar espacio de memoria principal a procesos cuando estos la necesitan.
- Liberar el espacio de memoria asignada a los procesos que han terminado.
- Lograr que el programador se abstraiga de la alocación de programas.
- Brindar seguridad entre los procesos para que no accedan a secciones privadas de otros.
- Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria. (Librerías, código común, etc).

  Por ejemplo, si tengo 2 instancias abiertas de Chrome, en lugar de tener el código el código répetido 2 veces, una para cada proceso, tenerlo una vez para ambos, y así hacer un uso más eficiente del sistema.
- Garantizar la performance del sistema.

Se espera de un Sistema Operativo un uso eficiente de la memoria con el fin de alojar el mayor número de procesos (tener un alto grado de multiprogramación, maximizando el tiempo productivo de CPU).

ADMINISTRACIÓN DE MEMORIA

Se realiza una división lógica de la Memoria Física (real) para alojar multiples procesos, y se debe garantizar la protección (que los procesos no se “pisen” entre ellos).

Esto depende de un mecanismo provisto por el **Hardware**. El diseño de los SO se acomoda para funcionar con la forma de trabajar del HW, y no al revés.

Se debe realizar una asignación eficiente (intentar contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismos)

REQUISITOS PARA LA ADMINISTRACIÓN DE MEMORIA PRINCIPAL

REUBICACIÓN:

Referido a la independencia del programa respecto de la administración de la memoria física. Se logra gracias al Hardware.

- El programador no debe ocuparse de conocer donde será colocado en la Memoria RAM
- Mientras un proceso se ejecuta, puede ser sacado y traído a la memoria (swap) y, posiblemente, colocarse en diferentes direcciones.
- Las referencias a la memoria se deben “traducir” según ubicación actual del proceso.

PROTECCIÓN:

También se logra gracias al hardware (Registro base y registro límite).

- Los procesos NO deben referenciar (acceder) a direcciones de memoria de otros procesos
  Salvo que tengan permiso
- El chequeo se debe realizar durante la ejecución
  NO es posible anticipar todas las referencias a memoria que un proceso puede realizar, ya que el proceso es dinámico

COMPARTICIÓN:

Ligado al manejo eficiente (no repetir datos) así como a los procesos cooperativos que deseen compartir datos entre sí para lograr un objetivo común.

- Permitir que varios procesos accedan a la misma porción de memoria.
- Permite un mejor uso/aprovechamiento de la memoria RAM, evitando copias innecesarias (repetidas) de instrucciones.

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 durante su ejecución.

El tamaño del espacio de direcciones depende de la Arquitectura del Procesador

- Con 32 bits: 0 .. 2^{32} - 1
- Con 64 bits: 0 .. 2^{64} – 1

Todo esto pensado desde el lado lógico.

Es independiente de la ubicación “real” del proceso en la Memoria RAM
(La dirección 100 del espacio de direcciones del proceso 1, no necesariamente va a la dirección física 100)

DIRECCIONES (LÓGICAS VS FÍSICA)

DIRECCIONES LÓGICAS:

Los procesos trabajan y generan direcciones lógicas.

- Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria real (dirección física).
- Representa una dirección en el “Espacio de Direcciones del Proceso”

En caso de usar direcciones Lógicas, es necesaria algún tipo de conversión a direcciones Físicas.

DIRECCIONES FÍSICAS

- Referencia una localidad en la Memoria Física (RAM) (Dirección absoluta)

CONVERSIÓN DE DIRECCIONES

Cada dirección lógica que se genera, debe ser convertida a la dirección física.

Una forma simple de realizarlo, es considerar que el espacio de direcciones (ED) de un proceso está en memoria RAM de forma contigua.

Para marcarlo se utilizan 2 registros auxiliares de la CPU, el Registro Base (dirección de comienzo del ED) y el Registro Límite (dirección final del ED).
Entonces, la dirección física se calcula haciendo Registro Base + un desplazamiento. Y se revisa que no este fuera del registro limite.

Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado en memoria, y el kernel se lo indica a la CPU en sus 2 registros (siendo la CPU quien se encarga de controlar en tiempo de ejecución que el proceso no se salga de los límites).

El contenido de estos registros varía entre procesos, por lo que es otra cosa que el SO debe cambiar ante un Context Switch.

DIR. LÓGICAS VS FÍSICAS

La CPU trabaja con direcciones **lógicas** (recordemos que en los registros base y límite se carga según el espacio de direcciones, y el espacio de direcciones es una abstracción).
Por tanto, para acceder a memoria principal, las debe transformar en direcciones físicas.

Esa resolución se realiza normalmente en tiempo de ejecución:

- Direcciones físicas y Lógicas son diferentes
- El mapeo entre Virtuales y Físicas es realizado por Hardware (el MMU de la CPU)

MEMORY MANAGEMENT UNIT (MMU)

La MMU es un dispositivo de hardware (ligado a la CPU, o incluso dentro de la CPU) que se encarga de mapear (traducir) direcciones virtuales a físicas.

Re-programar el MMU es una operación privilegiada que solo se puede realizar en modo Kernel.

Cada vez que hay un Context Switch, recibe el Registro Base y el Registro Límite para hacer el cálculo. El valor en el “registro de realocación” (contiene el mismo valor del Registro Base) es sumado a cada dirección lógica generada por el proceso de usuario al momento de acceder a la memoria.

MECANISMOS DE ASIGNACIÓN DE MEMORIA (PARTICIONES)

Este modelo planteado donde un proceso se almacena de manera contigua a partir de una dirección y hasta un límite, se denomina Particiones.

Cuando se utilizaba este modelo (ya no se utiliza) existían dos formas de particionar la memoria:

PARTICIONES FIJAS:

De antemano, el SO realiza una división lógica de la memoria RAM. Y cada espacio de direcciones (cada proceso) entra justo en una partición,

- La memoria se divide en particiones de tamaño fijo (pueden ser todas del mismo tamaño o variar)
- Alojan a un proceso cada una
- Cada proceso se coloca de acuerdo a un criterio en alguna partición.
- Genera fragmentación interna (espacio desperdiciado).

PARTICIONES DINÁMICAS:

No existe una división previa de la RAM. A medida que los procesos van apareciendo, la memoria se va particionando de acorde al tamaño de los espacio de direcciones que van llegando

- Las particiones varían en tamaño y en número
- Alojan un proceso cada una
- Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso
- Genera fragmentación externa.

FRAGMENTACIÓN

La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua.

FRAGMENTACIÓN INTERNA:

- Se produce en el esquema de particiones Fijas y en Paginación
- Es la porción de la partición que queda sin utilizar
- No tiene solución

FRAGMENTACIÓN EXTERNA:

- Se produce en el esquema de Particiones Dinámicas y en la tecnica de Segmentación
- Son huecos que van quedando en la memoria a medida que los procesos finalizan
- Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos utilizar
- Para solucionar el problema se puede acudir a la compactación, pero es muy costosa en cuanto a tiempo de Memoria Secundaria (HDD)

PROBLEMAS DEL ESQUEMA REGISTRO BASE + LÍMITE

El esquema de Registro Base + Limite se utilizaba ya que los SO operativos estaban limitados por el HW. Recordemos que el diseño de los SO se acomodó para funcionar con la forma de trabajar del HW, y no al revés. Este esquema presentaba problemas:

- Necesidad de almacenar el Espacio de Direcciones de forma continua en la Memoria Física
- Los primeros SO definían particiones fijas de memoria, luego evolucionaron a particiones dinámicas
- Genera fragmentación
- Mantiene “partes” del proceso que no son necesarias
- Los esquemas de particiones fijas y dinámicas no se usan hoy en día

Solución: Ante la evolución del hardware, surgieron 2 técnicas de administración

- Paginación
- Segmentación

PAGINACIÓN

La Memoria Física es dividida lógicamente en pequeños trozos de igual tamaño llamados “Marcos”

La Memoria Lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos, llamadas “Páginas”

En esta técnica, cualquier página del espacio de direcciones, puede ir a cualquier marco.
De esta manera se rompe la continuidad, ya no es necesario que el espacio de direcciones vaya de forma continua.

Puede generar fragmentación interna solo en la última página.

El SO debe mantener una **tabla de páginas** por cada proceso, donde cada entrada contiene el Marco a dónde fue a parar esa página.

Ahora el MMU utiliza la tabla de páginas (en vez del RB y RL) para realizar la conversión.
La dirección física se calcula como: un número de página + un desplazamiento dentro de la misma.

SEGMENTACIÓN

Esquema que se asemeja a la “visión del usuario”.

El programa se divide en partes/secciones. Un programa es una colección de segmentos.

Un segmento es una unidad lógica como: Programa Principal, Procedimientos y Funciones, variables locales y globales, stack, etc.
Y cada uno de estos segmentos se pueden cargar de manera no continua al espacio de direcciones.

Todos los segmentos de un programa pueden no tener el mismo tamaño (código, datos, rutinas). Por lo tanto, la Segmentación puede causar **Fragmentación Externa**. Al los segmentos de programa de distintos tamaños, quedan espacios pequeños en el medio sin utilizar.

Las direcciones Lógicas consisten en 2 partes:

> Número de Segmento + Desplazamiento dentro del segmento

El SO debe mantener una **Tabla de Segmentos** que permite mapear la dirección lógica en física.

Cada entrada contiene:

- Base: Dirección física de comienzo del segmento
- Limit: Longitud del Segmento
- Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos.
- Segment-table length register (STLR) : cantidad de segmentos de un programa

SEGMENTACIÓN VS PAGINACIÓN

Se utiliza más la paginación por sobre la Segmentación, ya que la fragmentación es menor. Y la Externa es costosa de resolver.

Sin embargo, la ventaja de Segmentación es que es más efectiva para Compartir y Proteger que la Paginación. Si tengo dos instancias de Chrome, el código es el mismo, por tanto el **segmento** que contiene el código se puede compartir facilmente, con una sola entrada en la tabla de segmentos (colocando la misma dirección base).

SEGMENTACIÓN PAGINADA

Es la combinación de las dos anteriores.

Cada segmento es dividido en páginas de tamaño fijo.

El cálculo de la memoria física, se obtiene con Segmentación, obteniendo las ventajas ya mencionadas en cuanto a compartir y proteger.

Para subir los datos a memoria principal, se utiliza Paginación. Se suben **páginas** a la RAM, eliminando la fragmentación Externa y sus problemas.

Para cada **segmento** hay una **tabla de página** que contiene a qué dirección de la memoria física fue el segmento.

La paginación

- Transparente al programador
- Elimina Fragmentación externa.

Segmentación

- Es visible al programador
- Facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección

Clase 2

Como ya vimos, en paginación el espacio de direcciones no debe estar físicamente contiguo en memoria para poder ejecutarse. El hardware traduce las direcciones lógicas a físicas utilizando la tabla de páginas del SO.

Evidentemente no es necesario que esté todo el proceso entero en memoria para ejecutarlo. Entonces ahí entra el concepto de **Memoria Virtual**.

MEMORIA VIRTUAL

La memoria virtual es una **técnica de gestión** de la memoria que se encarga de que el sistema operativo disponga, tanto para el software de usuario como para sí mismo, de **mayor cantidad de memoria** que esté disponible físicamente.

¿CÓMO TRABAJA LA MEMORIA VIRTUAL?

El SO trae a memoria las “piezas” de un proceso a medida que las necesita.

La parte de un proceso que el SO tiene en memoria principal en un momento determinado se denomina “**Conjunto residente**” (o Working Set).

Para poder determinar qué porción del proceso que aún no forman parte del conjunto residente son las que se requieren a continuación, el SO se apoya del **hardware**. Cuando se detecta una situación así, se debe cargar en memoria dicha porción para continuar con la ejecución.

VENTAJAS DE LA MEMORIA VIRTUAL

- **Más procesos pueden ser mantenidos en memoria.**

  Sólo son cargadas algunas secciones de cada proceso. Con más procesos en memoria principal es más probable que existan más procesos Ready aumentando el grado de multiprogramación

- **Un proceso puede ser más grande que la memoria Principal**

  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 MEMORIA VIRTUAL?

- Soporte del hardware: El hardware debe soportar paginación por demanda (y/o segmentación por demanda). Para ser eficiente, la memoria virtual debe tener soporte de hardware, ya que en cada referencia a memoria se deben actualizar los **bits de referencia y de modificación**

- **Area de Swap:** Se necesita un dispositivo de memoria secundaria (disco) que dé el apoyo para almacenar y mantener las secciones del proceso que no están en Memoria Principal.

- **Lógica programada del SO:** El SO debe ser capaz de manejar el movimiento de las páginas (o segmentos) entre la memoria principal y la secundaria.

¿QUÉ SE NECESITA PARA MEMORIA VIRTUAL CON PAGINACIÓN?

- Cada proceso debe tener su tabla de páginas
- Cada entrada de la tabla referencia al marco en el que se encuentra la página en memoria principal
- Cada entrada en la tabla de páginas tiene:
    - **Bit V:** Indica si la página está en memoria, para saber si el MMU puede traducir de lógico a físico o no (lo activa el SO, lo usa el HW)
    - **Bit M:** Indica si la página fue modificada. Si se modificó, en algún momento, se deben reflejar los cambios en memoria secundaria. (lo activa el HW, 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 la memoria principal. (Bit V = 0)

- 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 página que se necesite se cargue. Ya que el proceso para poder continuar depende de que esa página se cargue a memoria y esa operación de E/S es lenta, se pone en Espera al proceso para utilizar la CPU en otros mientras tanto, maximizando la eficiencia del sistema.
- El S.O. busca un “Frame o Marco Libre” en la memoria y genera una operación de E/S al disco para copiar en dicho Frame la página del proceso que se necesita utilizar.
- El SO puede asignarle la CPU a otro proceso mientras se completa la E/S La E/S se realizará y avisará mediante interrupción su finalización.
- Cuando la operación de E/S finaliza, se notifica al SO y este:
    - Actualiza la tabla de páginas del proceso Coloca el Bit V en 1 en la página en cuestión
    - Coloca la dirección base del Frame donde se colocó la página
- El proceso que generó el Fallo de Página vuelve a estado de Ready (listo)
- Cuando el proceso se ejecute, se volverá a ejecutar la instrucción que antes generó el fallo de página

La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles. Cada vez que hay que alocar una página en un marco, se produce un fallo de página ¿Qué sucede si es necesario alocar una página y ya no hay marcos disponibles? Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos (FIFO, Óptimo, LRU, etc.) ¿Cuál es el mejor algoritmo?: El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro

PERFORMANCE

Si los page faults son excesivos, la performance del sistema decae.

Tasa de Page Faults: 0 <= p <= 1

- Si p = 0 no hay page faults
- Si p = 1 , todos los accesos a memoria ram generan un fallo de página

Los Page Faults idealmente deben tender a 0. (Aunque **nunca** va a ser 0, es paginación por demanda, alguna vez falla **seguro**)

El tiempo efectivo de acceso a memoria (EAT Effective Access Time) depende del tiempo de acceso del hardware en sí, y de el tiempo de acceso con un fallo de página con todo lo que ello requiere. Por tanto a mayor tasa de fallo, mayor tiempo de acceso efectivo

SOBRE LA TABLA DE PÁGINAS

- Recordemos que cada proceso tiene su tabla de páginas
- La tabla tiene tantas entradas como potenciales páginas pueda tener un proceso
- Esa tabla de página idealmente debe estar en memoria principal para que la MMU la pueda traducir
- El tamaño de la tabla de páginas depende del espacio de direcciones del proceso
- Puede alcanzar un tamaño considerable

ORGANIZACIÓN DE LA TABLA DE PÁGINAS

Formas de organizar:

- Tabla de 1 nivel: Tabla única lineal
- Tabla de 2 niveles (o más, multinivel): Se hace una indirección
- Tabla invertida: Hashing

La forma de organizarla depende del HW subyacente

TABLA 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 páginas suele tener el mismo tamaño pero se busca que tengan un menor número de páginas por tabla
- La idea general es que cada tabla sea más pequeña
- Se busca que la tabla de páginas no ocupe demasida memoria RAM
- Además solo se carga una parcialidad de la tabla de páginas (solo lo que se necesite resolver)
- Existe un esquema de direccionamientos indirectos

La ventaja es que las tablas “inferiores” se pueden paginar: Es decir, se pueden sacar de memoria principal.

La desventaja es que para traducir una dirección de memoria, la MMU debe realizar 2 (o más, según la cantidad de memorias) accesos a memoria. Podría pasar, por ejemplo en un estructura de 4 niveles, que cada vez que el proceso quiere acceder a memoria la MMU debe acceder otras veces a memoria para traducir.

TABLA INVERTIDA

Utilizada en Arquitecturas donde el espacio de direcciones es muy grande donde las tablas de páginas ocuparían muchos niveles y la traducción sería costosa. Por esta razón se adopta esta técnica, subsanando la desventaja de la técnica anterior.

En esta técnica sólo se mantienen los PTEs (Entrada de tabla de página) de páginas presentes en memoria física. (Al contrario de la técnica anterior). Es decir, si una página no está memoria, tampoco estará su entrada en la tabla de páginas. Hay una entrada por cada marco de página en la memoria real. El espacio de direcciones de la tabla se refiera al espacio físico de la RAM, en vez del espacio de direcciones virtuales de un proceso.

Hay una sola tabla para todo el sistema

El número 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 solucinar colisiones (cuando el hash da igual para 2 direcciones virtuales). El fallo de página se produce si al intentar acceder a una entrada

TAMAÑO DE LA PÁGINA

El tamaño de página está basado en la cantidad de bits de desplazamiento que se utiliza en una dirección. Está definido por la arquitectura del HW, el SO debe adaptarse

Si es pequeño:

- Menor Fragmentación Interna. Al ser la página más chica, hay menor probabilidad de que queden espacios libres grandes.
- Más páginas requeridas por proceso
- Tablas de páginas más grandes. Porque al ser el tamaño de página chico, se necesitan menos bits para desplazamiento o direccionamiento, por lo tanto la tabla de página es más grande
- Más páginas pueden residir en memoria.

Si es grande:

- Mayor Fragmentación interna
- La memoria secundaria está diseñada para transferir grandes bloques de datos más eficientemente por lo tanto si tenemos tamaño de página grande, es más rápido mover páginas hacia la memoria principal.

TRANSLATION LOOKASIDE BUFFER (TLB)

El TLB o Buffer de Traducción Adelantada es una memoria caché que está en la CPU, que contiene las **entradas** de la tabla de páginas que fueron usadas más recientemente.

Es una memoria de alta velocidad, por tanto es un gran soporte que presta el hardware al SO disminuyendo los accesos a memoria principal.

Dada una dirección virtual, el procesador examina el TLB:

- Si la entrada de la tabla de páginas 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 páginas del proceso.

Ante cada Context Switch, todo lo que estaba en el TLB no sirve más.

Por tanto, con un Quantum chico (que genera muchos cambios de contexto) pierde mucha efectividad y sentido el TLB

ASIGNACIÓN DE MARCOS

Asignación fija:

Se le asigna un número fijo de marcos a cada proceso, y ese es su límite máximo. Este tamaño se asigna proporcionalmente al tamaño de cada proceso. Si necesita una nueva página, debe elegir una victima para liberar un marco. (No puede ir con reemplazo global)

Asignación dinámica:

El número de marcos de cada proceso varía.

A medida que el proceso necesita más marcos, el SO se los asigna.

Estas dos politicas, son extremos teoricos, que no sirven en la realidad. Los sistemas operativos reales buscan un punto intermedio

REEMPLAZO DE PÁGINAS

Los marcos podrían estar todos ocupadosen asignación dinamica o el proceso podría llegar a su límite máximo en asignación fija

¿Qué sucede si ocurre un fallo de página y no hay más marcos? Se debe seleccionar una página víctima.

¿Cuál sería Reemplazo Optimo? Que la página a ser removida no sea referenciada en un futuro próximo ya que evita generar un futuro fallo de página.

La mayoría de los reemplazos predicen el comportamiento futuro mirando el comportamiento pasado.

ALCANCE DEL REEMPLAZO

A la hora de seleccionar una página victima ¿puede ser cualquiera de toda la memoria o solo las páginas que corresponden al proceso que está generando el page fault?

Reemplazo Global: (no puede ir con asignación fija)

- 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 (en general se usa esta técnica)**:**

- 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 cuál 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 solo teórico

FIFO: Es el más sencillo

LRU (Least Recently Used): Requiere soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas más recientemente accedidas (Windows y Linux usan este)

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

Todos estos algoritmos trabajan con reemplazo local, porque si se trabajase con global tendrian que considerar como posible victima a un montón de páginas, lo cual es inviable en cuanto a eficiencia. Todos estos algoritmos consideran las páginas de un solo proceso

Clase 3

TRASHING (Hiperpaginación)

Concepto: decimos que un sistema está en **thrashing** cuando la CPU pasa más tiempo resolviendo fallos de página (paginando) que en ejecutar los procesos en sí. Como consecuencia, hay una baja importante de performance en el sistema. (En la técnica de paginación por demanda).

CICLO DEL THRASHING

1) El kernel monitorea el uso de la CPU.

2) Si hay baja utilización aumenta el grado de multiprogramación (aceptar más procesos a CPU).

3) Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos. (Los nuevos procesos aceptados)

4) Un proceso necesita más frames. Comienzan los page-faults y robo de frames a otros procesos.

5) Por swapping de páginas, y encolamiento en dispositivos, baja el uso de la CPU.

6) Vuelve a 1)

CONTROL DE TRASHING

- Se puede limitar el thrashing usando algoritmos de reemplazo local.

  Con este algoritmo, si un proceso entra en thrashing no roba frames a otros procesos.

  Si bien perjudica la performance del sistema, es controlable.

- Se puede bajar el grado de multiprogramación

  De esta manera hay más frames libres para que los procesos puedan tener sus páginas en memoria, reduciendo la frecuencia de page faults.

Ninguna de las dos opciones anteriormente mencionadas son siempre efectivas por si solas el 100% de las veces.

CONCLUSIÓN SOBRE THRASHING

Si un proceso cuenta con todos los frames que necesita, no habría thrashing.

- Una manera de abordar el Thrashing es utilizando la estrategia de Working Set, la cual se apoya en el modelo de localidad
- Otra estrategia con el mismo espíritu es la del algoritmo PFF (page fault frequency)

EL PRINCIPIO DE LOCALIDAD

(Modelo de localidad, cercanía de referencia, principio de cercanía, son todos sinónimos)

- **Las referencias a datos y programa dentro de un proceso tienden a agruparse físicamente**
- La localidad de un proceso en un momento dado se da por el conjunto de páginas que tiene en memoria en ese momento.
- En cortos períodos de tiempo, el proceso necesitará pocas “piezas” del proceso (por ejemplo, una página de instrucciones y otra de datos...)

EL MODELO DE WORKING SET

Recordemos que el Working Set es de cada proceso. Todo lo siguiente de este apartado se debe leer considerando que es para cada proceso mirado individualmente.

Se basa en el modelo de localidad.

- **Ventana del working set (Δ):** las referencias de memoria más recientes. (Δ es una cantidad).
- **Working set:** es el conjunto de páginas que tienen las más recientes Δ referencias a páginas.
- Si hay que seleccionar una victima, se selecciona una que no esté en el conjunto de trabajo en ese momento.

PROBLEMAS DEL MODELO DE WORKING SET

¿Que valor se le da al Δ?

Con un Δ chico: no cubrirá la localidad entera.

Con un Δ grande: puede tomar páginas que no se usan hace mucho, es decir de otras localidades.

¿Cómo hago para saber las últimas páginas referenciadas a memoria?

Para llevar el registro se necesita del apoyo del HW.

Usando el Bit de Referencia de las entradas de las tabla de página, el SO podría calcularlo, pero es muy costoso porque usa la CPU constantemente para saber la información, sacandole tiempo de CPU a los procesos.

Teoricamente el modelo WS está bueno, pero es impracticable en la realidad.

PREVENCIÓN DEL THRASHING POR PFF

La técnica PFF (Page Fault Frecuency o Frecuencia de Fallo de Páginas), en lugar de calcular el Working Set de los procesos, utiliza la tasa de fallos de página para estimar si el proceso tiene un conjunto residente que representa adecuadamente al Working Set.

- **PFF alta:** Se necesitan más frames, porque los marcos no le alcanzan para mantener la localidad que necesita
- **PFF baja:** Los procesos tienen frames asignados que le sobran

Se establecen límites superior e inferior de las PFF's deseadas.

- **Si se Excede PFF máx:** Se le asignan frames a mas al proceso, ya que el mismo genera muchos PF y probablemente los esté necesitando.
- **Si la PFF está por debajo del mínimo:** Se le pueden quitar frames al proceso para ser utilizados en los que necesitan mas. (se hace si necesito frames para otro proceso, sino no es necesario hacerlo)

DEMONIO DE PAGINACIÓN

Demonio: Proceso del sistema operativo que se ejecuta por atras

Es un servicio del SO

El demonio de paginación es un proceso creado por el SO durante el arranque que apoya a la administración de la memoria

Se ejecuta cuando el sistema tiene una baja utilización o cuando algún parámetro de la memoria lo indica

Tareas:

- **Limpiar páginas modificadas sincronizándolas con el swap** (Reduciendo el tiempo de swap posterior ya que las páginas están "limpias" y reduciendo el tiempo de transferencia al sincronizar varias páginas contiguas)
- **Mantener un cierto número de marcos libres en el sistema.**
- **Demorar la liberación de una página hasta que haga falta realmente**

MEMORIA COMPARTIDA

Es un servicio del SO

Gracias al uso de la tabla de páginas varios procesos pueden compartir un marco de memoria; para ello ese marco debe estar asociado a una página en la tabla de páginas de cada proceso

El número de página asociado al marco puede ser diferente en cada proceso

Código compartido

- Los procesos comparten una copia de código (sólo lectura) por ej. editores de texto, compiladores, etc
- Los datos son privados a cada proceso y se encuentran en páginas no compartidas

MAPEO DE ARCHIVO EN MEMORIA

Es un servicio del SO

Técnica que permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtuales. Es decir, puedo modificar y trabajar sobre el archivo directamente en memoria principal, el SO se encarga de la correlación del mapeo y de llevarlo luego a la región de memoria secundaria correspondiente.

- El contenido del archivo no se sube a memoria hasta que se generan Page Faults.
- Cuando el proceso termina o el archivo se libera, las páginas modificadas son escritas en el archivo correspondiente
- Permite realizar E/S de una manera alternativa a usar operaciones directamente sobre el Sistema de Archivos

COPIA EN ESCRITURA

La copia en escritura (Copy-on-Write, COW) permite a los procesos compartir inicialmente las mismas páginas de memoria

Si uno de ellos modifica una página compartida la página es copiada

En fork, permite inicialmente que padre e hijo utilicen las mismas páginas sin necesidad de duplicación.

COW permite crear procesos de forma más eficiente debido a que sólo las páginas modificadas son duplicadas

AREA DE INTERCAMBIO

Area utilizada para mantener páginas de un proceso que no estén en el WS

Físicamente puede ser un Área dedicada, separada del Sistema de Archivos (Por ejemplo, en Linux) o un archivo dentro del Sistema de Archivos (Por ejemplo, Windows)

Técnicas para la Administración:

a) Cada vez que se crea un proceso **se reserva una zona del área de intercambio** igual al tamaño de imagen del proceso. A cada proceso se le asigna la dirección en disco de su área de intercambio. La lectura se realiza sumando el número de página virtual a la dirección de comienzo del área asignada al proceso. Esta técnica desperdicia espacio del area de intercambio

b) No se asigna nada inicialmente. A cada página se le asigna su espacio en disco cuando se va a intercambiar, y el espacio se libera cuando la página vuelve a memoria. Problema: se debe llevar contabilidad en memoria (página a página) de la localización de las páginas en disco (una nueva estructura de datos en memoria)