Tema-3-Memoria-Virtual.pdf

fer__luque

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

Tema 3: Memoria Virtual

1 Generalidades sobre gestión de memoria

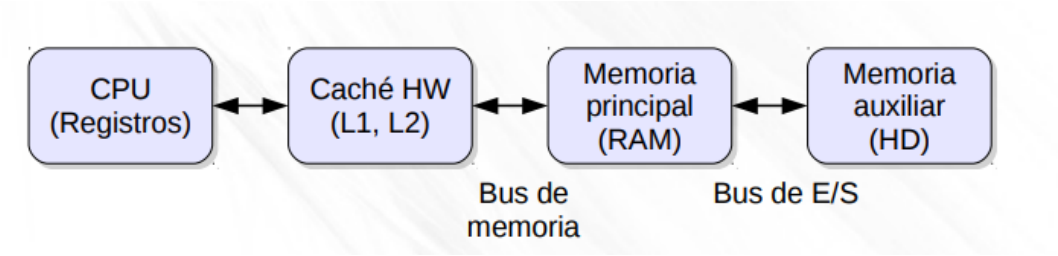
Existen distintas formas de implementar los sistemas de memoria. Una buena idea para ello es utilizar una **jerarquía de memorias**.

Jerarquía de memoria

Principios generales:

- A menor cantidad, menor tiempo de acceso
- A mayor cantidad, menor coste por byte

Por tanto, los elementos que sean frecuentemente accedidos se colocarán en las memorias más rápidas, mientras que el resto, en memorias más lentas:



Conceptos sobre cachés

El uso de cachés es un principio importante en los sistemas de computación y se basa en la idea de tener una **copia original de la memoria principal** que permite ser **más rápidamente accedida** que la original. El objetivo por tanto es hacer eficientes los casos más frecuentes.

En el momento de localizar un determinado dato en caché se pueden dar las siguientes situaciones:

- ***Cache hit***: El dato se encuentra en cache
- ***Cache miss***: El dato no se encuentra en cache. Hay que ir a "buscarlo" a memoria principal.

El **TAE** (Tiempo Acceso Efectivo) es el tiempo medio de acceso a una celda de memoria, y se calcula:

> $$TAE = Probabilidad\_acierto * coste\_acierto + Probabilidad\_fallo * coste\_fallo$$

La cache funciona porque los programas no generan solicitudes de datos aleatorias, sino que siguen el **principio de localidad**, que establece que, durante la ejecución de un programa las referencias a memoria por parte del procesador, tanto para instrucciones como para datos, tienden a estar agrupadas.

Espacios de direcciones lógico y físico

Lógico: Conjunto de direcciones lógicas (o relativas si hay reubicación dinámica) (o virtuales si el sistema soporta memoria virtual) generadas por un programa.

Físico: Conjunto de direcciones físicas correspondientes a las direcciones lógicas en un instante dado de ejecución del programa.

Código absoluto

En el código absoluto las direcciones se generan en tiempo de compilación (y/o enlace), por tanto hay que conocer las direcciones de memoria donde se va a ejecutar el programa en tiempo de compilación/enlace. Por supuesto, no es reubicable.

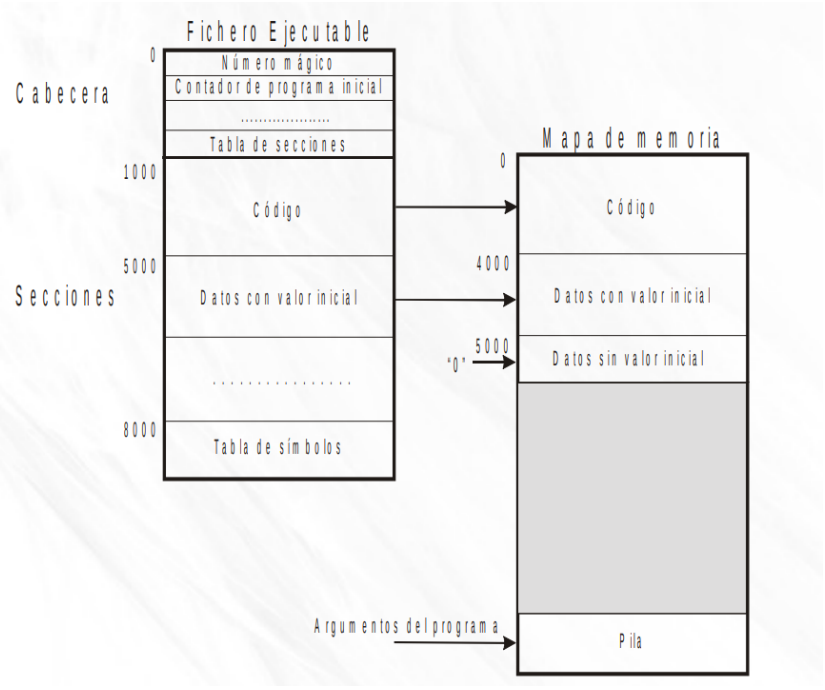
Reubicación estática y dinámica

Con la **reubicación estática** las direcciones se generan en tiempo de carga, lo que hace que el ejecutable tenga referencias relativas. Esto provoca que una vez cargado el programa en memoria no pueda moverse.

Con la **reubicación dinámica**, las direcciones se generan en tiempo de ejecución (el programa maneja referencia que no son las direcciones de memoria reales a las que accede). Este tipo de reubicación permite que los programas puedan salir de memoria y volver a ella en cualquier ubicación. Es con esta reubicación donde aparece la **distinción entre el espacio virtual** (o lógico) **y el espacio físico** de direcciones a las que se accede

Mapa de memoria de un proceso

Se define la **imagen del proceso** al par formado por el mapa de memoria y el PCB.



Formato del ejecutable

Al final del proceso de compilación y montaje, se genera un archivo ejecutable que contiene el código del programa. En la figura podemos ver que este ejecutable está dividido en varias secciones:

- **Cabecera**: Información que permite interpretar el contenido del ejecutable (número mágico, dirección del *entry point* del programa, tabla que describe las secciones del ejecutable...)
- **Secciones**:
    - Código
    - Datos con valor inicial (globales por supuesto)
    - Datos sin valor inicial (globales también)

Las regiones que presenta el **mapa de memoria** inicial de un proceso se corresponden con las secciones del ejecutable más la pila del proceso.

Podemos encontrar además otras regiones en el mapa de memoria tales como el *heap*, archivos proyectados, memoria compartida, pilas de threads...

Objetivos de la gestión de memoria

Organización

Hay dos tipos de organización de la memoria:

- **Organización contigua**: La asignación de memoria para un programa se hace en un único bloque de posiciones contigua en memoria principal. Este método es muy antiguo. Los bloques podían ser de tamaños fijos (particiones fijas) o variables (particiones variables).
- **Organización no contigua**: Permiten dividir el programa en bloques que se pueden colocar en zonas no necesariamente contiguas de memoria principal. Hay tres tipos:
    - **Paginación**
    - **Segmentación**
    - **Segmentación paginada**

¿Qué nos proporciona la paginación/segmentación sin VM?

Las referencias a memoria dentro de un programa en ejecución con paginación/segmentación se realizan sobre el espacio de direcciones lógico. Esto permite que, el programa pueda ser retirado de memoria y volver a ser traído a esta en una nueva área de memoria, y además permite que este esté dividido en trozos (páginas o segmentos) que **NO** tienen que estar ubicados en memoria de forma contigua.

Sin embargo, durante la ejecución de un programa, **todos los trozos** deben residir en memoria principal.

Gestión

Dado ya un esquema de organización, ¿qué estrategias hay que seguir para obtener un rendimiento óptimo?

- De **asignación** de memoria: Contigua, no contigua (diferenciar de la organización)
- De **sustitución** o reemplazo de programas (o partes) en memoria principal
- De **búsqueda** o recuperación de programas (o partes) en memoria auxiliar

De todo este trabajo se encarga el **gestor de memoria** de un sistema operativo, que influye enormemente en el rendimiento del sistema operativo.

Protección/compartición

De memoria entre el SO y los procesos de usuario y entre los propios procesos de usuario.

Intercambio (*Swapping*)

La idea principal del *swapping* es intercambiar procesos (programas) entre memoria principal y memoria auxiliar. El proceso pasará a estado "SUSPENDIDO_BLOQUEADO" y el programa pasará a disco (memoria auxiliar).

Para que sea eficiente, la memoria auxiliar debe ser rápida, pues el principal factor en el tiempo de intercambio es el tiempo de transferencia entre MP y MA.

El ***swapper*** tiene las siguientes responsablidades:

- Seleccionar procesos para retirar
- Seleccionar procesos para incorporar
- Gestionar y asignar el espacio de intercambio.

Cabe destacar que en sistemas de tipo Unix solo está activo si la carga del sistema es muy grande. Además también existen ciertos tipos de procesos que nunca deberían ser intercambiados, por ejemplo, algunos procesos del sistema. Si el SO aborda este problema, permite que se especifique si un programa es o no intercambiable pero es un privilegio restringido a una clase de procesos o de usuarios.

2 Organización de la Memoria Virtual

Concepto de Memoria Virtual

Apuntes previos:

- Necesitamos paginación/segmentación básica
- El tamaño de un programa puede exceder la cantidad de memoria física disponible para él
- El número de procesos ejecutándose en MP (Grado de multiprogramación) aumenta drásticamente
- Resuelve el problema del crecimiento dinámico del mapa de memoria de los procesos

Idea clave: se usan dos niveles de la jerarquía de memoria: **MP**, donde residen las partes necesarias en un momento dado (**conjunto residente**); **MA**, donde reside el espacio de direcciones completo del programa.

Requisitos:

- Disponer de la información relacionada con qué partes del programa se encuentran en MP y qué partes en MA: **Tabla de Ubicación en Disco (TUD)**
- Política para la resolución de un acceso a memoria situado en una parte que en ese momento no reside en MP
- Política de movimiento de partes del espacio de direcciones entre MP y MA

Unidad de Gestión de Memoria

El **MMU** (*Memory Management Unit*) es un dispositivo hardware que **traduce direcciones virtuales a físicas**. Este dispositivo está gestionado por el SO, pues este le suministra la información necesaria que cambia cuando se hace un cambio de contexto.

En su esquema más simple, el valor del registro base se suma a cada dirección generada por la ejecución del programa en CPU y este resultado se utiliza en el bus de memoria para acceder a la dirección física deseada.

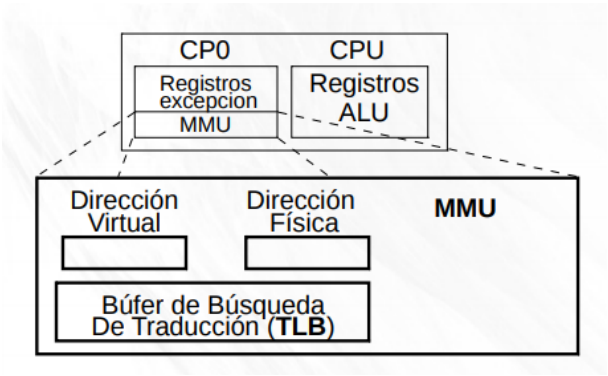
El programa de usuario siempre trata con direcciones virtuales, nunca físicas.

El MMU tiene unos registros **TLB** (*Translation Look-aside Buffer*, búfer de búsqueda de traducción previa), encargados de mantener las correspondencias página virtual-física resueltas con anterioridad.

Entre sus responsabilidades:

- Si ***TLB hit*** (acierto) traduce de dirección virtual a física

- Si ***TLB miss*** (fallo) usa la tabla de páginas del mapa de memoria para realizar la traducción, teniendo en cuenta que: si parte del espacio de direcciones que contiene la dirección resultado de la traducción reside en MP, carga un nuevo registro TLB y realiza la traducción; si no, genera *page fault exception*.



Memoria Virtual Paginada

Con memoria virtual paginada, la organización del espacio de direcciones físicas de un proceso es **NO contigua**.

Esta **memoria física** se asigna mediante bloques de tamaño fijo, denominados **marcos de página** o páginas físicas (*physical frames*), cuyo tamaño es siempre potencia de dos. Este tamaño viene dado por el harware y determina varios aspectos:

- El número de bits menos significativos que se emplearán en las direcciones virtuales para completar direcciones físicas en el proceso de traducción. En otras palabras, el **offset en el marco de página y en la página virtual**
- El número de bits menos significativos de la dirección física alojada en cada entrada de la tabla de páginas que valdrán 0. **Bits menos significativos de la dirección física base del marco**

Las direcciones del **espacio lógico** (virtual) de un proceso, se interpretan a dos niveles:

- Bits **más significativos** determinan la **página virtual** en la que se encuentre la dirección
- Bits **menos significativos** completan la dirección física (*offset* en marco de página)

La correspondencia entre página virtual y marco de página la almacena la entrada de la **tabla de páginas**, donde se almacena la dirección base del marco de página.

Dirección virtual

La genera la CPU y se interpreta como un par:

- **Número de página**: Determina la entrada de la tabla de páginas (TP) y se sitúa en los bits más significativos de la dirección virtual
- ***Offset***: Permite completar la dirección física correspondiente a la dirección virtual y se sitúa en los bits menos significativos de la dirección virtual

Dirección física

Dirección real de memoria principal, que se calcula en base a:

- **Dirección base del marco de página**, que se relaciona con la correspondiente página virtual por medio de la TP

- ***Offset*** de la dirección virtual, que se suma a la dirección base para obtener la dirección física final

Estructuras

Para la traducción de una dirección virtual generada por la CPU a una dirección física que nos permita acceder a la memoria, se emplean las siguientes estructuras:

Tabla de páginas

Mantiene la información necesaria para realizar dicha traducción. Esta representa el mapa de memoria de la MV de un proceso. En un registro de la CPU se guarda la dirección de comienzo de la TP del proceso actual, PBTB. Su valor forma parte del PCB.

Tabla de ubicación en Disco

Mantiene la ubicación de cada página en el almacenamiento auxiliar, para el *swapping*

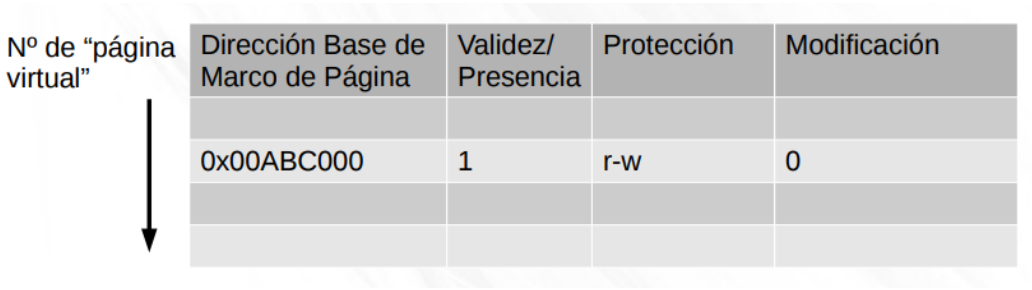
Tabla de Marcos de Página

Mantiene información relativa a cada marco de página en el que se divide la memoria principal

Tabla de páginas

Esta tabla contiene una entrada por cada página virtual del proceso con los siguientes campos:

- Dirección base del marco
- Protección, modo de acceso
- Bit de validez/presencia
- Bit de modificación (*dirty bit*)



Para ejecutar una instrucción o acceder a un dato, la página virtual que la/lo contiene debe estar presente en memoria principal. Es esa la función del **bit de validez/presencia** que tiene un doble significad:

- Vale 1 si la página es válida y está cargada en MP
- Vale 0 si la página no está en MP (no está presente) o es una página que no pertenece a su espacio de direcciones

En el campo de **protección** mantenemos la información sobre si se puede leer, escribir y ejecutar dicha página virtual.

La TP está ordenada o indexada por número de página virtual, obtenido de los bits más significativos de la dirección virtual del espacio de direcciones virtual del proceso.

Implementación

La TP se mantiene en memoria principal (*kernel*). El **registro base de la tabla de páginas** (RBTP) apunta a la TP y forma parte del contexto de registros (PCB).

Siguiendo este esquema inicial, cada acceso a memoria requiere realmente dos accesos:

- Uno a la TP
- Otro a la dirección real

La solución pasa por el MMU y sus registros TLB (*Translation look-aside buffer*). Siempre que se produzca un *TLB hit* solo se realizará un acceso a memoria. Otro problema viene determinado por el tamaño de la tabla de páginas.

Ejemplo

Pongamos que la dirección virtual tiene 32 bits y que el tamaño de página son 3 KB (2^{12} bytes), por lo que el *offset* requerirá de 12 bits y tendremos 20 bits para el tamaño del número de página virtual. Es decir **tendremos 2^{20} páginas virtuales**.

Suponiendo que cada entrada de TP es sólo el campo de dirección base (que no es así), cada entrada son 32 bits = 4 bytes, por lo que:

tamaño TP = 2^{20} entradas $*4$ bytes/entrada = 4194304 bytes = 4096 KB

Como podemos comprobar, la tabla de páginas ocupa muchísima memoria. Par reducir este tamaño, se aplica la **Paginación multinivel**

Paginación multinivel

Idea: Paginar las tablas de páginas, es decir dividir las tablas de páginas en partes que coincidan con el tamaño de una página. Las partes no válidas del espacio de direcciones virtual se dejarán sin paginar a nivel de página, lo que implica disponer de distintas granularidades para paginación.

La idea es que **no es necesario hacer explícita la paginación a nivel de página hasta que se habilite esa parte del espacio de direcciones**. Esas partes del espacio de direcciones virtual no necesitan tener tabla de páginas.

Paginación a dos niveles

Pongamos direcciones virtuales de 32 bits, tamaño de página física de 4096 bytes y tamaño de la entrada de TP ocupa 4 bytes.

A un nivel:

> Indice en TP (PV) | Offset

A dos niveles con solo ocupación de la primer ay última entrada de TP de primer nivel:

$$\frac{4096\ bytes/pagina}{4\ bytes/entrada} = 1024\ entradas$$

Si una página física contiene 1024 entradas de la TP, entonces necesitamos $2^n = 1024;\ n = 10\ bits$ para indexar entradas y la dirección virtual se interpreta:

> Indice en TP 1º nivel | Indice en Tps de 2º nivel | Offset

Compartición de páginas

Una página en memoria que contenga código (texto) puede ser compartida por varios procesos. Básicamente las Tablas de Páginas de los procesos que comparten dicha página reflejan en su entrada la misma dirección base de marco.

Cuando se seleccionen "víctimas" para el *swapping* estas páginas compartidas no serán seleccionadas.

Un ejemplo es el código de la **libc** que reside en páginas compartidas por todos los procesos, así como el del programa cargador/enlazador ld.so.

Tabla de ubicación en Disco

La TUD contiene una entrada por cada página virtual con los siguientes campos:

- Identificación del dispositivo lógico que soporta la memoria auxiliar (*major, minor*)
- Identificación del bloque que contiene el respaldo de la página virtual

También está indexada por el número de página virtual, obtenido de los bits más significativos de la dirección virtual del espacio de direcciones virtual del proceso.

Contenido de la TP y TUD

PV	Marco	V/P	Prot.	Mod.	Dispositivo lógico	Nº bloque
0	0x20	1	r-x	0	(8,3)	1328
1	0x00	0	r-x	0	(8,3)	1329
2	0x50	1	rw-		(8,3)	1330
3	-	0	rw-		(8,3)	1331
4	-	0				
	...	0				
14	0x60	1	rw-	1		
15	0x70	1	rw-	1		

	RAM
0x00	PV1
0x10	
0x20	PV0
0x30	
0x40	
0x50	PV2
0x60	PV14
0x70	PV15
0x80	
...	

Swap device
PV0
PV1
PV2
PV3
PV14
PV15

El espacio virtual para este ejemplo es de 2^4 páginas y 2^4 para desplazamiento, porque:

- 2^4 páginas porque tenemos 16 entradas en la tabla (PV)
- 2^4 para desplazamiento porque los bits menos significativos de las direcciones base de marco que están a 0 son los **4 primeros**, es decir, la primera cifra hexadecimal. Luego el tamaño para una dirección virtual mínimo debe ser igual al *offset* en página física

A pesar de esto: **se podría ampliar el número de bits para identificar páginas virtuales**.

Apuntes interesantes

- La entrada 3 pertenece a una página virtual válida que nunca se cargó en memoria pero que tiene respaldo en disco
- La entrada 4 no es válida porque no tiene respaldo
- La entrada 1 estuvo en memoria en el marco 0x00 pero ahora no está (V/P=0)

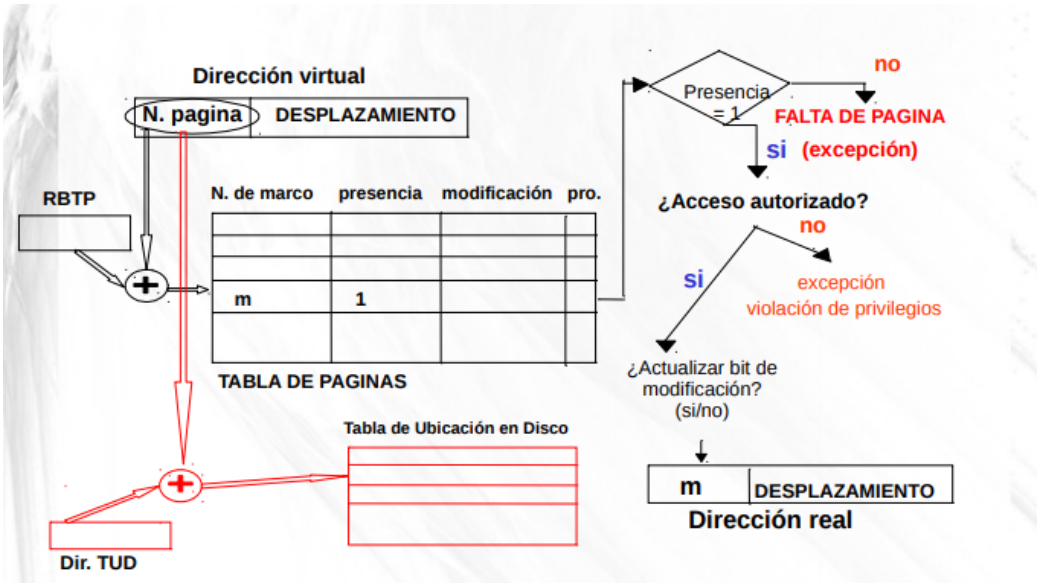
Memoria virtual mediante paginación por demanda (*demand paging*)

El *demand paging* es la forma más común de implementar memoria virtual. Se basa en el **modelo de localidad** para la ejecución de los programas:

- Una **localidad** se define como un conjunto de páginas que se utilizan durante un período de tiempo
- Durante la ejecución de un programa, este utiliza distintas localidades.

Con esta implementación, los programas residen en el dispositivo de intercambio (*backing store*). Cuano el SO crea un proceso, antes de pasarlo a LISTO, solo carga en MP un subconjunto de páginas del programa.

La Tabla de Páginas para el proceso se inicializa con los valores correctos, páginas válidas y cargadas en RAM, páginas válidas pero no cargadas en RAM y páginas inválidas. Tras esto ya se puede cambiar a LISTO para ser elegido por el *scheduler*.



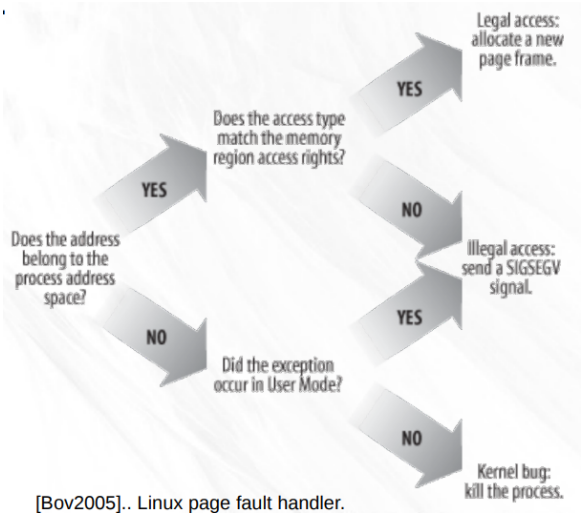
Falta de página (*page fault*)

Cuando se intenta acceder a un marco que no se encuentra en Memoria Principal, se lanza una excepción que, siempre que se trate de una página válida, realiza lo siguiente:

1. Encontrar la ubicación en disco de la página solicitada mirando la entrada de la TUD.
2. Encontrar un marco libre. Si no hubiera, se puede optar por reemplazar una página de memoria RAM.
3. Cargar la página desde disco al marco libre de memoria RAM → proceso "BLOQUEADO".
4. FIN E/S (RSI)
    1. Actualizar TP (bit presencia=1, nº marco,...)
    2. Desbloquear proceso → proceso "LISTOS"
5. Planif_CPU() selecciona proceso → Reiniciar la instrucción que originó la falta de página

Linux

Linux distingue entre errores de programación relativos a acceso a memoria y **errores debidos a falta de página**. ***Page fault exception handler***:



[Bov2005].. Linux page fault handler.

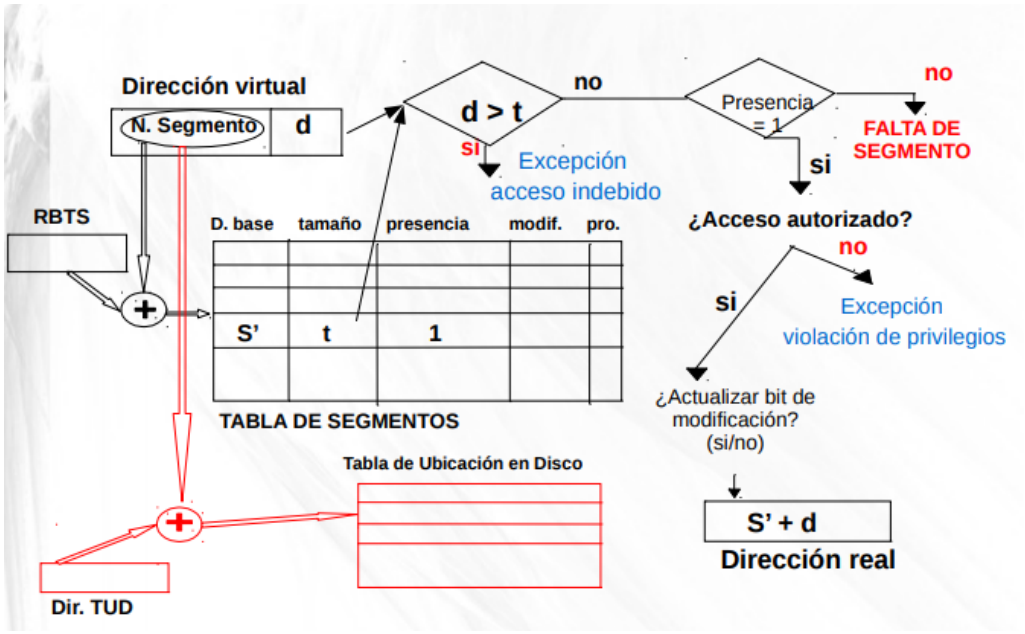
Memoria Virtual Segmentada

En esta ocasión la **dirección virtual** es una tupla formada por dos elementos (**id_segmento**, **offset**), que genera la CPu y se mantiene en dos registros distintos , el registro de **segmento** (segmento de memoria virtual) y el registro de ***offset*** (desplazamiento en el segmento).

La **dirección física** se calcula en base a:

- Dirección de memoria principal donde comienza el segmento virtual, se encuentra en la entrada TS (Tabla de segmentos)
- *Offset*, que se suma a la dirección anterior.

Esquema de traducción



RBTS: Registro base de la Tabla de segmentos. También se almacena en el PCB y se carga en un registro.

A diferencia de la paginación, ahora las longitudes de los segmentos son variables, por eso se comprueba siempre que el valor de *offset* esté dentro del límite de la TS (Registro Límite de la Tabla de Segmentos).

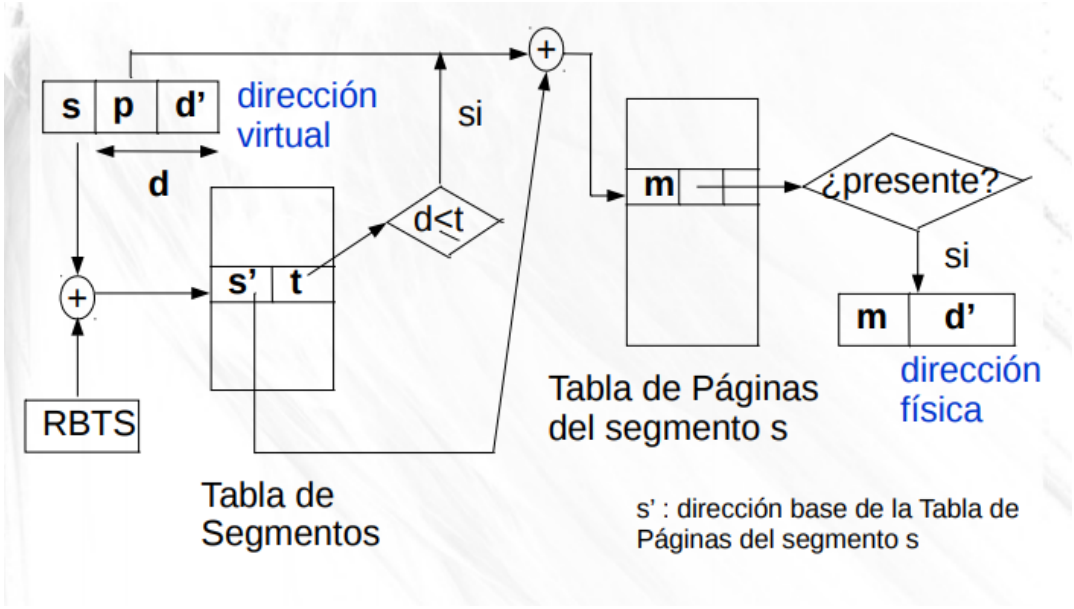
Segmentación paginada

Con segmentación: La variabilidad del tamaño de los segmentos y el requisito de memoria contigua en un segmento complican la gestión de la MP y MA.

Con paginación: Simplifica la gestión pero complica la compartición y protección.

Algunos sistemas combinan ambos enfoques obteniendo las ventajas de la segmentación y eliminando los problemas de una gestión de memoria compleja.

Esquema de traducción



3 Gestión de la Memoria Virtual

Conceptos

Con **Gestión de Memoria Virtual paginada** nos referimos al conjunto de criterios de clasificación respecto a:

- Políticas de **asignación de memoria principal**: que controlan el espacio de MP asignado a un proceso. La asignación de muchos marcos reduce las faltas de página pero también el número de procesos activos y al revés
- Políticas de **búsqueda (recuperación) de páginas alojadas en memoria auxiliar**: controlan cuándo traer a MP una página de MA. Tipos:
    - *Demand paging*: Se trae cuando se produce una falta de página. Esta puede tener un efecto importante sobre el rendimiento del sistema. Garantiza que en MP sólo están las páginas necesarias en cada momento y la sobrecarga de decidir qué páginas llevar a MP es mínima
    - Paginación anticipada: Puede cargar páginas erróneas
- Políticas de **sustitución (reemplazo) de páginas de memoria principal**:
    - Sustitución local
    - Sustitución global: Problema, el conjunto de páginas en MP de un proceso no sólo depende de su comportamiento, sino también del del resto de procesos.

Independientemente de la política de sustitución que usemos, existen ciertos criterios que siempre deben cumplirse:

- Páginas "**limpias**" frente a "**sucias**", para minimizar el número de transferencias MP-swap
- Seleccionar en último lugar **páginas compartidas** entre varios procesos (para reducir el número de faltas de página)
- **Páginas especiales (bloqueadas)** que no pueden ser elegidos para la sustitución. Búferes de E/S, búferes de cauces...

Influencia del tamaño de página

Páginas pequeñas

- TPs mayores
- Mayor nº transferencias MP-swap
- Reducen fragmentación interna

Páginas mayores

- Información que no se está usando (y quizás nunca se use) está ocupando MP
- Aumenta fragmentación interna

Por eso se busca un equilibrio en el tamaño de las páginas. Normalmente, se sitúan entre los 512 bytes y los 16 kB.

Algoritmos de sustitución

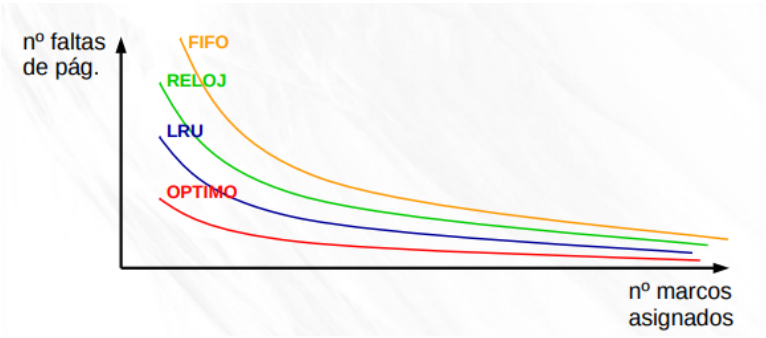
En cuanto al tipo de asignación de memoria principal podemos tener las siguientes combinaciones:

- **Asignación fija** y **sustitución local**
- **Asignación variable** y **sustitución local**
- **Asignación variable** y **sustitución global*

El algoritmo de sustitución más **óptimo** es aquel que sustituye la página que **no se va a referenciar** en un futuro o aquella **que se va a referenciar más tarde**. Como esto no es posible saberlo, los tipos que se implementan son:

- **FIFO**: Se sustituye la página más antigua
- **LRU** (*Last Recently Used*): Sustituye la página que se referenció hace más tiempo
- **Algoritmo de reloj**: Aproximación más eficiente al algoritmo LRU en la que cada página tiene asociado un **bit de referencia**, que activa el hardware cuando se accede a dicha página. Los marcos de página se representan por una lista circular y un puntero a la página visitada hace más tiempo. Para la selección de una página se sigue el siguiente método:
    1. Consulta el marco actual
    2. Es ref=0?
        1. No: ref=0; siguiente marco y paso 1
        2. Sí: se selecciona y se incrementa la posición

Comparativa de algoritmos



La cantidad de marcos asignados influye más en las faltas de página que el algoritmo de sustitución usado.

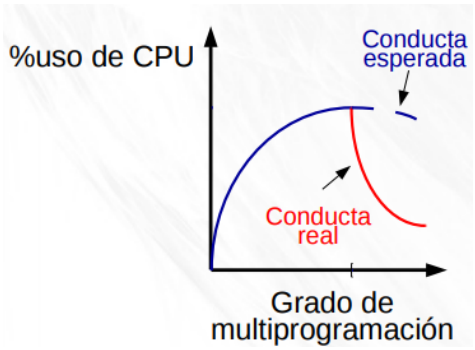
Hiperpaginación (*thrashing*)

Un proceso está *thrashing* (**hiperpaginando**) si está más tiempo haciendo E/S sobre *backing store* que ejecutándose, es decir, si la tasa de faltas de página es muy alta.

Si un sistema presenta suficientes procesos en este estado, puede ocurrir que:

- El uso de CPU sea muy bajo porque la falta de página genera E/S
- El bajo uso de CPU provoca la creación de nuevos procesos para incrementar el uso
- Estos nuevos procesos incrementan de nuevo el promedio de faltas de página y entran en *thrashing*

Típico escenario de hiperpaginación en el que SO está resolviendo faltas de página, es decir, el tiempo en modo *kernel* aumenta, y el tiempo útil de computación cae:



Para **evitar la hiperpaginación**:

- Asegurar que cada proceso existente tenga asignado un espacio en relación a su comportamiento. **Algoritmos de asignación variable de marcos**, capaces de estimar el conjunto residente de páginas óptimo
- Actuar directamente sobre el grado de programación: **Algoritmos de regulación de carga**, encargados de detectar el comienzo de la hiperpaginación y mantener el grado de multiprogramación próximo a un valor óptimo (un ejemplo de criterio sería que si el **tiempo en modo núcleo** supera al 50% del tiempo total de CPU, se está produciendo hiperpaginación)

Principio de localidad. Modelo del conjunto de trabajo

Principio de localidad: Establece que los programas referencian una **pequeña parte del espacio de direcciones** durante un **tiempo determinado**.

Existen **dos tipos** de localidad:

- **Temporal**: Una posición de memoria referenciada recientemente tiene una alta probabilidad de ser referenciada próximamente
- **Espacial**: Si una posición de memoria ha sido referenciada recientemente, existe una alta probabilidad de que las posiciones adyacentes sean referenciadas

Construcciones de programación que provocan ambas localidades:

	Espacial	Temporal
Código	Secuencia (ni bifurcación ni saltos)	ciclos
Datos	arrays	Contadores en ciclos

Modelo del conjunto de trabajo: El conjunto de trabajo de páginas (*Working Set*), $W(t, \tau)$, de un proceso en el instante t es el conjunto de páginas referenciado por el proceso durante el intervalo de tiempo $(t - \tau, t)$.

Mientras este conjunto de trabajo pueda residir en MP, el nº de faltas de página es pequeño. Cuando se eliminan de MP páginas de este conjunto, crece el nº de faltas de página.

Propiedades:

- Son transitorios y difieren en su composición
- No se puede predecir el tamaño ni la composición de un conjunto de trabajo futuro

Requisitos:

- Un proceso solamente puede ejecutarse si su conjunto de trabajo actual está en memoria principal
- Una página no puede retirarse de MP si forma parte del conjunto de trabajo actual

Presenta una estrategia absoluta:

- Si aumenta el número de páginas que referencia y no hay espacio en MP, entonces el proceso se intercambia a disco
- Al sacar de memoria varios procesos, el resto finalizan antes y dan paso a los que se han sacado

Algoritmos de cálculo del conjunto residente de páginas

Algoritmo del conjunto de trabajo

En cada referencia, se determina el conjunto de trabajo y **sólo** esas páginas son mantenidas en MP.

Ejemplo:

Algoritmo de Frecuencia de Falta de Página (FFP)

Para ajustar el conjunto de páginas de un proceso que residen en MP (**conjunto residente**), usa los **intervalos de tiempo entre dos faltas de página consecutivas**:

- Si el intervalo es mayor que un umbral, entonces, todas las páginas no referenciadas en dicho intervalo son retiradas de MP
- En otro caso, la nueva página es simplemente incluida en el conjunto de páginas residentes

Formalización:

- t_c = instante de tiempo de la actual falta de página
- t_{c-1} = anterior falta de página

- Z = conjunto de páginas referenciadas en un intervalo de tiempo
- R = conjunto de páginas residentes en MP

$$R(t_c, Y) = \begin{cases} Z(t_{c-1}, t_c) & si\ t_c - t_{c-1} > Y \\ R(t_{c-1}, Y) + Z(t_c) & en\ otro\ caso \end{cases}$$

Garantiza que el conjunto residente crece cuando las faltas de página son frecuentes y decrece cuando no lo son

Ejemplo: