**TEMA 3: GESTIÓN DE MEMORIA**

MMU (memory management unit): componente hardware construido dentro de la CPU que facilita la gestión de memoria. Todos los sistemas actuales utilizan memoria virtual y hacemos eficiente la gestión de memoria a través de mmu.

1. **MEMORIA DEL SO**

Hay dos niveles de gestión:

* Gestor memoria del SO: Asigna porciones de memoria a los procesos y a los subsistemas del SO. Se prima al SO frente a los procesos, pues la gestión de los subsistemas es crítica, mientras que asignar memoria a los procesos no es algo crítico.
* Gestor memoria de los procesos: gestión dinámica de procesos (malloc, free…).

La gestión de memoria de los procesos y del SO es transparente a los procesos. La pila queda fuera de la gestión del proceso. Es tan transparente que la conexión se realiza mediante una llamada al sistema pidiendo memoria.

**ASIGNACIÓN DE MEMORIA**

La implementación Linux de gestión de memoria cubre:

* Asignación de memoria kernel:
* Distribuidor sistema amigo (buddy) para crear grandes bloques de memoria continua.
* Distribuidores tableta (slab,slob,slub): asignan memoria inferior a una página.
* Mecanismo vmalloc(): asigna grandes cantidades de memoria no contiguas.
* Asignación de memoria de usuario: construcción y gestión de los espacios de direcciones de los procesos.
* Reclamación de memoria: recuperar la memoria previamente asignada pero que ya no está en uso. Se puede asignar mucha memoria, pero si nunca se libera, el SO detecta cuando un proceso con mucha memoria asignada no la está utilizando y se la quita. Si no puede liberar la memoria, como último recurso se utiliza el swapping, que quita los procesos de memoria directamente. Se hace una partición para swapping del doble de la memoria RAM, se guarda la memoria en disco y se saca de manera rápida, provocando fragmentación.

La asignación de memoria dinámica a los procesos de usuario tiene requisitos diferentes a la propia del kernel:

* Las peticiones de procesos no se consideran urgentes: el kernel intenta diferir su asignación, ya que la solicitud no indica utilización inmediata.
* Los procesos de usuario no son confiables: el kernel debe estar preparado para atrapar errores de direccionamiento.
* Cada proceso tiene su propio espacio de direcciones separado del resto de procesos.
* El kernel puede añadir/suprimir rangos de direcciones lineales.

**APIs DE GESTIÓN EN LINUX**

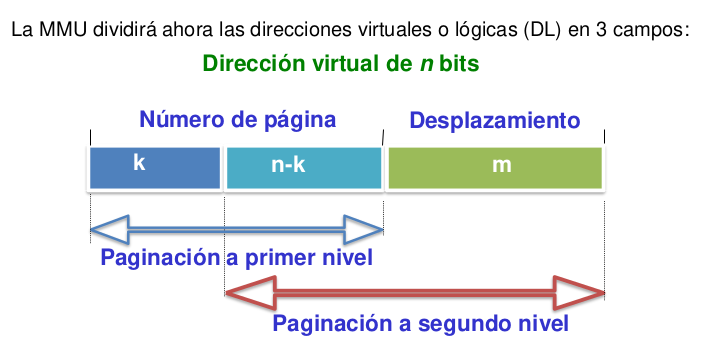
Tienen dos interfaces:

* Interfaces de llamadas al sistema: interfaz de usuario (Ejemplos: malloc/free/... mmap/...mlock/munlock/... wapon/swappoff).
* Interfaz intra-kernel: interfaz de mm al resto de memoria  (Ejemplos: kmalloc/kfree, verify\_area, get\_free\_page/ free\_page).

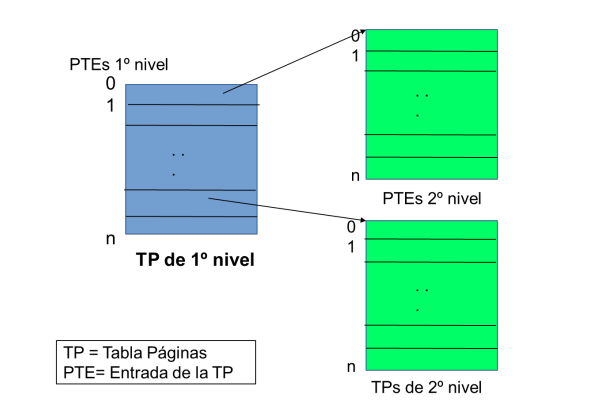
1. **MEMORIA DE PROCESOS**

**Paginación a un nivel**: La memoria física se divide en bloques de tamaño fijo, denominados marcos de página (el tamaño es potencia de dos) y el proceso, en bloques del mismo tamaño, llamados páginas. Necesito en la RAM tantos huecos como el número de páginas en que hayamos dividido el proceso. En los marcos de páginas se situarán las páginas de los procesos cuando estos se carguen en memoria. En una tabla de páginas (con un bit de validez) cada entrada tiene la localización en memoria de cada página. Saber dónde está cargado un proceso viene dado por la tabla de páginas.

**Paginación multinivel**: Se basa en la idea de paginar las tablas de páginas. Con esto se consigue que se cargue sólo la porción de la tabla de páginas que está en uso y se deje sin cargar aquellos espacios de direcciones que no se usan.

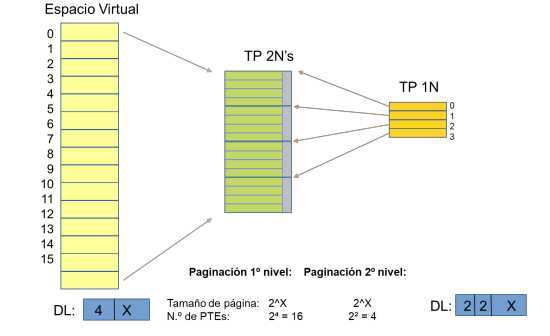
**La paginación a dos niveles**:consiste en aplicar paginación a la tabla de páginas.Dado n-k

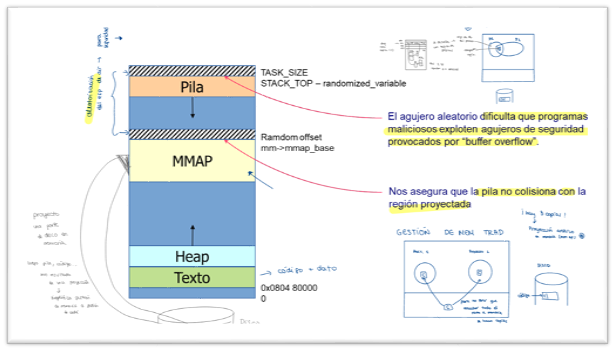
**La paginación multinivel**:  tiene la ventaja de que:

* Tablas de 2º nivel (o mayores) no son necesarias porque mapean regiones no válidas del ED del proceso.
* Si bien la tabla de paǵinas de primer nivel debe estar en memoria principal, no ocurre así con las de segundo nivel (o superiores).

La tabla de páginas a primer nivel siempre tiene que estar cargada en memoria.

Acceder a memoria me cuesta el acceso a la caché. Entre el software y el hardware se ha aumentado la probabilidad de acceso a caché, de casi de un 99%.

ESPACIO VIRTUAL EN x86 



**Aleatorización del espacio de direcciones para que la pila no colisione con la región proyectada.**

Tradicionalmente la memoria se ha gestionado de la siguiente forma. Supongamos que tenemos dos procesos P1 Y P2 que ejecutan el mismo ejecutable. En un esquema tradicional se copia el código en caché y se lleva una copia del código a cada proceso, pero hay un problema: hay tres copias del mismo ejecutable, 1 en el SO + en cada proceso.

Los SO utilizan un mecanismo llamado proyección de archivos en memoria (mmap). La idea es que cuando lea una página de disco se proyecte en la memoria del proceso, evitando la copia en el SO. Se lleva al espacio de memoria del proceso. Cuando otro proceso solicita la copia, ambos procesos compartirán la proyección, desde el punto de vista de la implementación, en el espacio de direcciones del proceso (descrito con la tabla de páginas).

El proceso apunta a la tabla de páginas que apunta a los marcos de página. Se ajusta la tabla de páginas de un proceso para que apunte al marco correspondiente.

De esta forma, se reduce la demanda de memoria. No hay dos páginas con el mismo contenido de memoria, todos los procesos que tengan esa página la van a compartir.

Cuando se quiere leer el contenido de un archivo, se proyecta en memoria y se reajusta la tabla de páginas.

El responsable de hacer las proyecciones es el gestor de memoria, no el sistema de archivos.

Código, datos, pila, …, etc, son resultados de una proyección. Así, se simplifica la gestión de memoria porque toda la memoria asociada a procesos se ve como una caché. Cuando se quiere leer un ejecutable y cargarlo en memoria, se proyecta en ella tanto la biblioteca como los datos.

**Cat maps dentro del pseudosistema de archivos** **/proc:** para cada región de memoria nos da información de la dirección virtual en la que empieza y acaba. Se separa en código, datos y heap, los cuales tienen niveles de protección distintos. La **p** indica que la proyección es privada; por ejemplo, si la página compartida, en vez de ser códigos son datos y desde un proceso intento escribir, entonces se hace un duplicado de la página y luego se permite la escritura. Si es **shared**, es compartida por los procesos  (indica el dispositivo al que ir y el i-nodo). Por último, muestra de donde viene el archivo, las bibliotecas dinámicas.

Si se tiene un disco con dos particiones, cada una va a tener dos números, el número del dispositivo y el de la partición.

El código nunca empieza en la dirección 0, ya que si se referencia un puntero al 0 ocurre un segmentation fault.

En el espacio de direcciones del proceso, encontramos el código, la pila, una zona de datos privada, el vdso (proyección y gestión de bibliotecas dinámicas) y vsyscall (los sistemas actuales ya no utilizan int para hacer las llamadas al sistema, conmutar de modo usuario a modo kernel es muy costoso). Se introdujeron dos funciones de entrada y salida al sistema, proyectando una página en el espacio de un proceso que facilita entrar y salir de un proceso. La dirección virtual en la que se sitúan es en las direcciones más altas que puedo direccionar, proyectando una página que pertenece al kernel dentro del espacio de usuario.

**LINUX: REGIONES DE MEMORIA**

El kernel representa un intervalo de direcciones lineales contiguas con el mismo tipo de protección mediante un recurso denominado región de memoria. Estas se caracterizan por su dirección de inicio, su longitud y los derechos de acceso. Por eficiencia, estas tienen un tamaño múltiplo al del tamaño de página (4KiB en I32). Por ejemplo:

* Región de código: permisos lectura-ejecución.
* Región de datos: permisos de lectura-escritura.
* Región de pila: lectura-escritura, crecimiento.

**ÁREAS DE MEMORIA VIRTUAL (MV)**

Las tablas de páginas no son adecuadas para representar espacios de direcciones grandes, especialmente si son dispersos, por lo que vamos a superponer otra gestión de memoria sobre la paginación. Linux representa cada región de memoria con una estructura denominada vm-area (virtual memory area).

Las vm-areas contienen la información necesaria para poder establecer la traducción de una dirección que la TP no pueda realizar. El espacio de direcciones de un proceso se representa como:

* Una lista de vm-areas, si el número de vm-areas es pequeño
* Si su número es elevado se organizan en un árbol rojo-negro.

Las vm-areas no tienen un contador de referencias, por lo que solo pueden pertenecer a un proceso.

**Problemas**: un proceso puede ser un espacio disperso, tener muchas entradas vacías, ocupar poco espacio, luego tendría una tabla de páginas que ocupa mucho con respecto al proceso. Otro problema es la paginación por demanda, que puede conllevar a ejecutar un proceso parcialmente cargado.

**Solución**: sobre la descripción de páginas creamos otra dirección basada en vm-areas, la cual va a contener información para construir las tablas de páginas y poder proyectar. Crear una lista de vm-areas para describir el espacio de direcciones del proceso. Si el vm-area es pequeño, es decir,  si tengo menos de 32 vms, utilizo una lista. Si no, un árbol.

**ELEMENTOS DE UN VM-AREA**

Las principales componentes son:

* Rango de direcciones – direcciones de inicio/fin de la región.
* Indicadores VM – Palabra que, entre otras cosas, es uno de los indicadores VM\_READ, VM\_WRITE, VM\_EXEC, que controlan si el vm-área se puede leer, escribir, ejecutar.
* Información de enlace – Punteros a la lista de vm-áreas, subárboles derecho e izquierdo.
* Operaciones VM y datos privados – contiene el puntero de operaciones VM que apunta a funciones, y sus datos privados, a invocar cuando se producen ciertos eventos relacionados por memoria virtual, (Ejemplo: una falta de página).
* Información de archivo proyectado – Si el vm-área es la proyección de un archivo, contiene el puntero a archivo y el desplazamiento, necesarios para localizar los datos en el archivo.

**OPERACIONES DE VM**

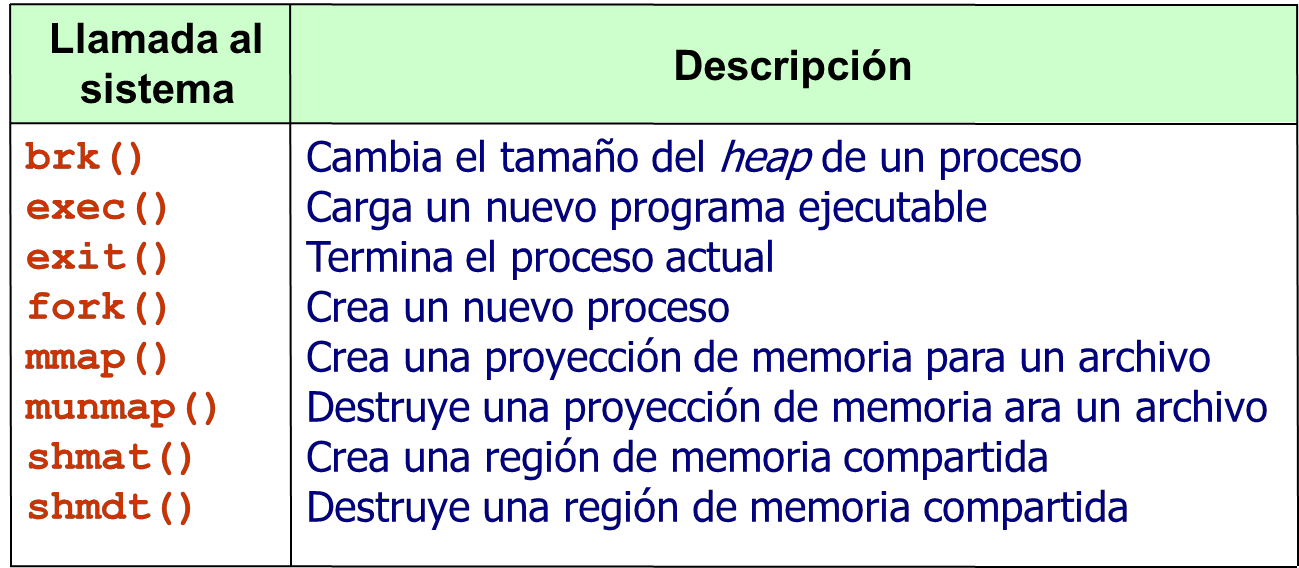
El puntero de operaciones VM proporciona a las vm-áreas características de orientación objetos: cada tipo de vm-área puede tener diferentes manejadores. Cualquier objeto (sistema de archivos, dispositivo de caracteres, ..) proyectado en memoria de usuario (mmap) puede suministrar sus propias operaciones. Algunas de ellas son:

* open() – crear un vm-área.
* close() – destruir un vm-área.
* fault() – manejador de la falta de página que no está en TP.

**Nota:** Si hay una falta de página en el código, hay que ir al disco. Si es en la pila, fault indica donde tengo que ir a hacer la proyección.

**API DE MEMORIA**

Las llamadas al sistema relacionadas con la creación, destrucción o modificación de regiones de memoria son:



Hay un pseudodispositivo que es 0: leo tantos ceros como pida /dev/zero, una fuente infinita de información.

fd= open(“/dev/zero”, ...)

mmap (fd, …) // Hago la proyección

fork()

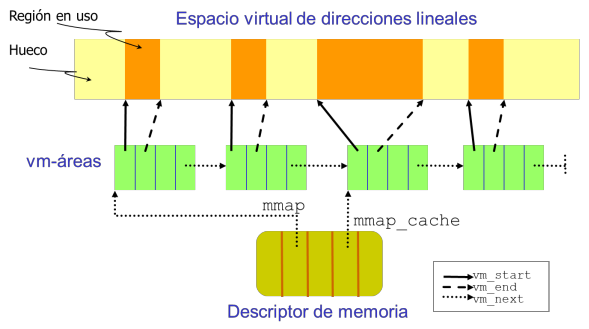
// Creo un hijo y ambos comparten esa región

/dev/null es un sumidero de información, todo lo que mando ahí se pierde

**DESCRIPCIÓN DEL ED  (Espacio de Direcciones)**

Viene descrito por la mm\_struct:

* pgd = tabla de páginas de primer nivel
* Vm\_area\_struct \*mmap = lista de vm-areas.
* start\_code, end\_code: donde empieza el código de un proceso y donde acaba.



Por cada región en uso del ED, se construye un vm-area que la describe. La ordenación de las vm-areas es importante, ya que las direcciones de las vm-areas se crean de manera que sean posteriores a las anteriores. Se van explorando las regiones para ver dónde se ha producido la falta de página. El mm\_struct tiene un puntero a la primera región hábil del proceso, al descriptor de página y a la mm\_cache. En el mm\_struct se guarda cuál fue la última región en la que se produjeron faltas de páginas.

La vm-area da la dirección de inicio, de final y el programa que proyecta el archivo.

**¿Cómo utilizo las vm-areas?** Una de las ventajas de las vm-areas es que permiten crear las tablas de páginas poco a poco, según las necesito, de manera que me ahorro memoria y transferencia de disco. No necesito todos los datos de un programa por algún condicional.

Cuando hay una falta de página, se coge la dirección faltante y se comprueba si está en alguna de las vm-area. Si está en alguno de los huecos, se está haciendo una referencia a una dirección inválida. Si no, ya se determina cuál es la vm-area a utilizar. La vm-area indica dónde está el programa que ha generado la excepción, y una vez localizada, se construye la tabla de páginas correspondiente, se va a memoria y se restaura la memoria física.

**Nota**: La única gestión de memoria que se deja en mano del usuario es la del heap.

1. **MEMORIA VIRTUAL**

Se trata de una gestión de memoria donde se produce una separación del espacio de direcciones de un proceso, que denominamos espacio virtual, del espacio físico, es decir, la memoria física asignada al objeto de poder ejecutar un proceso cuyo espacio virtual sea mayor que la memoria RAM. Considera el disco como una extensión de la RAM, donde se almacenan las páginas que no estén en uso en un momento dado, de manera que cuando sean necesarias instrucciones/datos que no están en memoria se realicen los ajustes necesarios para cargarlas en RAM.

Al conjunto de páginas cargadas en memoria se lo denomina **conjunto residente** (RSS).

Para ello, son necesarios cambios en la gestión de memoria tanto a nivel hardware como software:

**i) PAGINACIÓN POR DEMANDA**

Consiste en esperar a traer una página de disco a RAM solo cuando es referenciada, es decir, demandada. Se gestionan aquellas páginas que contienen datos que no se necesitan. Con cargar las primeras páginas de un proceso es suficiente para que empiece a ejecutarse.

Tiene como ventajas:

* Reducir las E/S
* Menos memoria en uso (tamaño del conjunto residente es menor que el de la TP de un proceso)
* Respuesta más rápida (hay que cargar menos páginas para poder empezar a ejecutar un proceso)
* Más procesos en ejecución

Se necesita una página cuando el proceso referencia una instrucción o dato de los que contiene:

* Si la referencia es inválida, se aborta la ejecución
* Si es válida, se trae a memoria desde disco

**ii) FALTA DE PÁGINA**

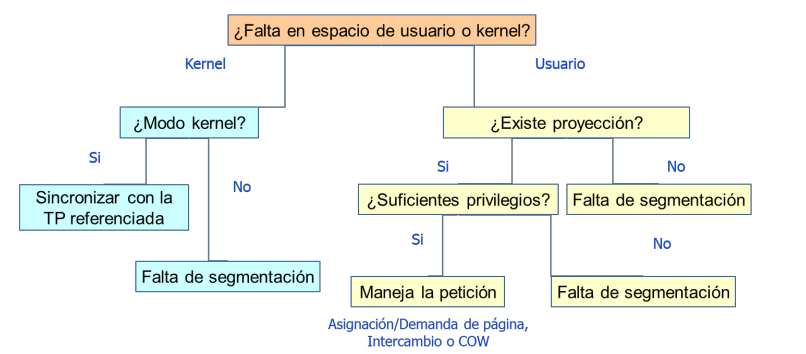
Para implementar paginación por demanda es necesario modificar:

* Hardware: aparece la excepción falta de página para tratar errores de direccionamiento inválido.
* Software: Introducir la rutina de excepción de falta de página.

En cuanto al bit de validez presente en cada PTE:

* Bit a 1: la página está en memoria.
* Bit a 0: la página “no está en memoria”. La página está en RAM o no está cargada en memoria, porque se la ha quitado un proceso.

Al iniciar un proceso, todos los bits de validez de las PTE están a cero. Durante la traducción de direcciones, cuando se referencia una página, la MMU (Unidad de gestión de Memoria) comprueba el bit de validez de la PTE correspondiente, y, si está a cero, va a producir una excepción denominada falta de página. Si no, es decir, está a 1, se realiza la traducción.



Cuando se produce falta de página, en el caso de Linux, ocurre lo siguiente:

Hardware: la mmu suministra al SO la dirección lógica que ha producido la falta de página. Hay dos casos:

- La dirección pertenece al kernel: si estamos en modo kernel hay que sincronizar las tablas de páginas del kernel. Si no estoy en modo kernel, falta de segmentación.

- A nivel de usuario, hay que determinar si la dirección es válida o no consultando si pertenece a una proyección. Si no pertenece, falta de segmentación. Si pertenece, hay que ver si la operación que quiero hacer está permitida o no. Si no está permitida, falta de segmentación. Si hay privilegios, manejo la petición.

TRATAMIENTO DE LA EXCEPCIÓN DE FALTA DE PÁGINA

Etapas de la rutina de tratamiento de la excepción falta de página:

1. Mira si la referencia es válida, si no lo es aborta al proceso. Consulta las vm-areas y a partir de ahí determina si es válida o no.

2. Si es válida pero no está en memoria:

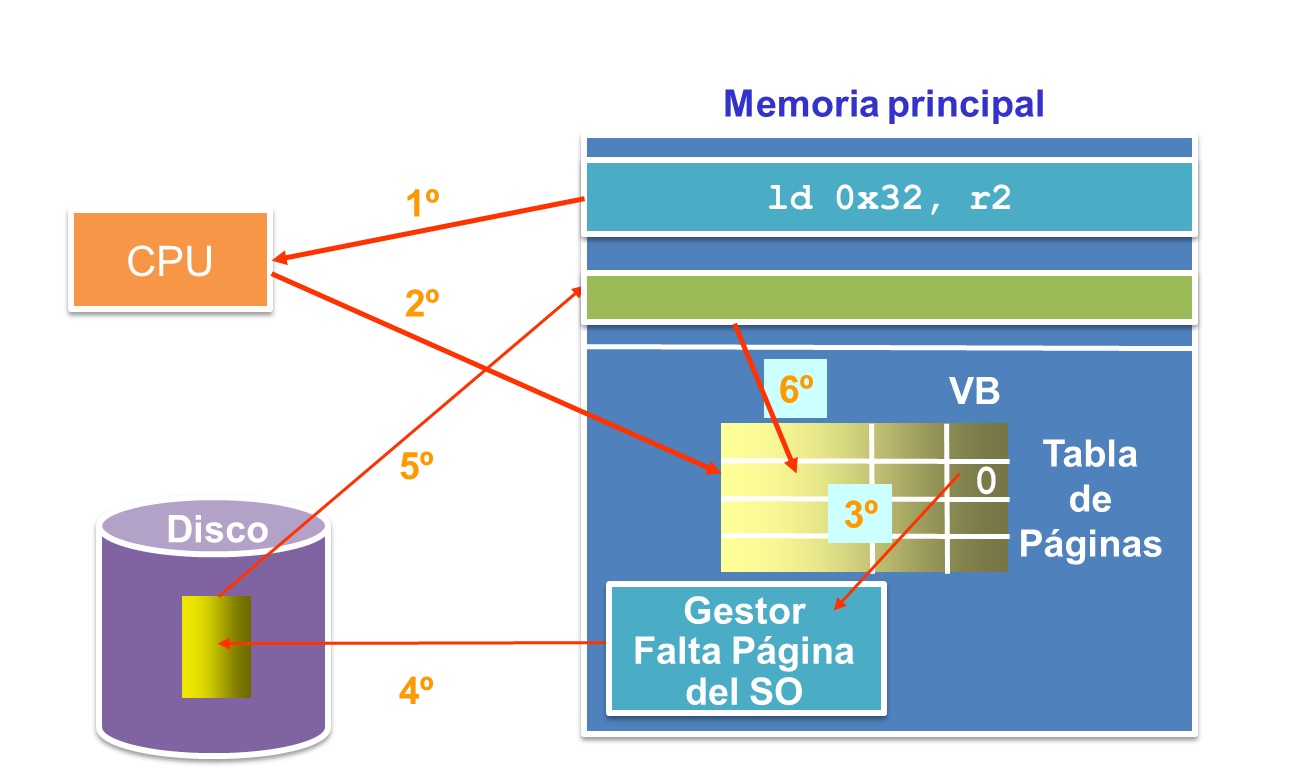
3. Obtiene un marco vacío.

4. Carga la página faltante en el marco libre.

5. Activa el bit de validez.

6. Rearranca la instrucción que produjo la falta.

Si la página está en memoria, pero la traducción no es válida, entonces se reasigna la página. Un proceso puede generar una falta de página sobre un proceso a la que se le acaba de quitar, pero que aún está en memoria → la reajusta y la puede volver a utilizar.



GESTIÓN DE LA PILA

La rutina de gestión de la falta de página permite que el sistema operativo gestione la asignación de memoria para la pila de usuario.

Supongamos que el usuario intenta empujar en la pila un dato que requiere nueva asignación de memoria. La rutina de servicio de falta de página detecta que se está intentando escribir en una dirección inicialmente no válida pero muy próxima a una región que tiene permiso de escritura y es de tipo VM\_GROWSDOWN, entonces asigna una página adyacente (virtualmente) a las páginas de la pila.

En el ejecutable no está la pila, luego la responsabilidad de la gestión de la pila es del SO, no del programador. Cuando empujo algo en la pila ¿esas direcciones son válidas? La última dirección válida es el tope de la pila, entonces a priori no es válida, por lo que en la rutina de excepción tengo que detectar cuando estoy empujando en la pila, para pedir una región de memoria. Si hago referencia a una dirección de memoria cerca del tope, emitiendo que el programa está intentando escribir en la pila,  tengo que validar esa dirección de memoria. Validamos las regiones ajustando la dirección de fin.

El almacén de respaldo, la información del almacenamiento permanente de respaldo de esa región. El almacén de respaldo de la pila es una proyección de la región de pila sobre la página anónima, que es una página del kernel que está llena de ceros (las páginas anónimas están en RAM en la página del kernel).

**iii) BIT DE VALIDEZ REVISITADO**

En ocasiones puede que una página esté en memoria pero el SO haya desactivado el bit de validez. Esto le permite al SO:

* Simular en soft el bit de referencia.
* Implementar la copia-sobre-escritura (copy-on-write), que se trata de un mecanismo que para ahorrar memoria permite compartir páginas cuando son de lectura y duplicarlas cuando se intenta escribir en ellas.

Si hay dos páginas en memoria iguales, el SO quita una y la otra la comparte entre los procesos que la están utilizando. Cuando se intenta escribir en la página, se duplica la página y se da una copia a cada proceso, dándole permiso de escritura a aquella cuyo proceso lo ha pedido.

Si una página no está referenciada, es candidata a sacarla de memoria si necesito memoria.

**iv) SUSTITUCIONES DE PÁGINAS**

Podemos reducir la sobre-asignación de memoria si la RSE (Rutina de Servicio de Excepción) de falta de página incluye la sustitución de páginas, es decir, encontrar páginas asignadas pero que no están actualmente en uso para sacarlas de memoria y poder usar el marco.

La sustitución de páginas completa la separación entre memoria lógica y física.

**Rendimiento de la sustitución**: el algoritmo de sustitución que se utilice debería producir el mínimo número de faltas de páginas, es decir, debemos retirar páginas que tengan la menor probabilidad de ser usadas.

Si la memoria está llena, si me traigo una página a RAM pero está llena, coge una página que no esté en uso,  la lleva a disco y en ese espacio meto la página que quería insertar inicialmente.

ALGORITMOS DE SUSTITUCIÓN

Los sistemas operativos en producción utilizan:

* Linux: LRU (Least Recently Used)
* Windows: monoprocesadores, Reloj (LRU); multiprocesadores, aleatorio.
* OS X: Segunda oportunidad.

1. **ALGORITMO LRU (Least Recently Used)**

No quiero quitarle a un proceso una página que vaya a necesitar de inmediato porque entonces va a inducir en otra falta de página. LRU utiliza el comportamiento pasado de un proceso como estimación de futuro. Mira qué páginas se han usado en el pasado para ver cual referenciar. La que lleva más tiempo sin ser referenciada es la candidata a salir.

Denominamos cadena de referencia ( w = r1,r2,r3,...,ri, ..., ) a la secuencia de números de las páginas referenciadas por un proceso durante su ejecución. Sea BKWDt(r), distancia atrás de la página r en t, la distancia de r a la última ocurrencia de la página en la parte precedente de w. LRU (Least Recently Used) selecciona la página con máxima distancia atrás:  https://lh3.googleusercontent.com/1Y5g1O49esdMbgS0tUh-IiDdI7R-nuClgzSIZdP4XBnjidDt_I-HkAv8jBJQCAU_JM9P99Ww3O4N4bs-eU6G0LthlcJeF9UenIUf0GAbt0eLtwziT_F_QQ3H7msiAXCzTAEo-MDemEee1jLcvS1P3TeVM_beJOMadQp6CVYfc1a0rI57G7C025hnWB8Brw

Si hay más de una, se elige una arbitraria. St(m) es el estado de la memoria definido por el conjunto de páginas cargadas en m marcos en tiempo t, que viene dado por:https://lh4.googleusercontent.com/LWNxiovJ5boxglFBtQf21ADOkwyWPQDJ8UyPAeozmWLjR4Ka_BGMD0J5M8u3N-XHRgAcKzv5H1Qger5HVWB-9UQA1JVocT2GvBqUpY3-nICf76R49bQ0fpF4PVFcDcBvQo2iMm9gBLSbGVa9O3DSv9qT-UEbKVxD9yPWjC1gpda97hTxenQFD9VANX8zWgdonde Xt es la página traída en t, e Yt es la página sustituida en t.

Debido al principio de localidad, la distancia atrás es un buen estimador de la distancia adelante.

Utiliza las referencias pasadas como estimación de lo que va a ocurrir, si no se ha utilizado en mucho tiempo es previsible que en el futuro tampoco.

Implementar LRU requiere contabilizar el uso de cada página. Una forma de implementación es con un contador de la siguiente forma:

* Añadimos a cada PTE un contador.
* Al referenciar una página, se copia el tiempo del reloj en el contador.
* Si necesitamos cambiar una página, se elige aquella cuyo contador sea menor.

Las implementaciones son costosas, por lo que se utilizan aproximaciones como el algoritmo del reloj.

**V) BIT SUCIO**

El uso de un bit adicional en las PTE, el bit sucio (o de modificación), reduce la sobrecarga de transferencia de algunas páginas:

* Aquellas que no se han modificado no hay que salvarlas ya que tenemos una copia en el archivo ejecutable.
* Solo escribimos en disco las páginas que han sido modificadas.

Hay que minimizar el trasiego de página. Cuando se va a sacar una página, los sistemas distinguen entre sí la página que ha sido modificada o no. Si no ha sido modificada, tengo una copia de la página en el ejecutable, luego puede descartarla ya que tengo de donde recuperarlas cuando vuelva a hacer falta. Si se ha modificado con respecto a lo que hay en el ejecutable, no puedo descartarla, tengo que hacer una copia en el área de intercambio, donde se almacenan.

Si se tiene una página en el espacio de intercambio no se puede cambiar. El usuario solo puede activar o desactivar un área.

¿Por qué hay que crear una partición de disco especialmente para intercambios? Los sistemas de archivos se gestionan con unos algoritmos y swapping con otros. Linux crea un partición de disco para el área de swapping, por lo que influye en la ejecución. Esto se debe a que los algoritmos de gestión de disco van a ser diferentes. Por ejemplo, una continuación de disco es un algoritmo más simple.

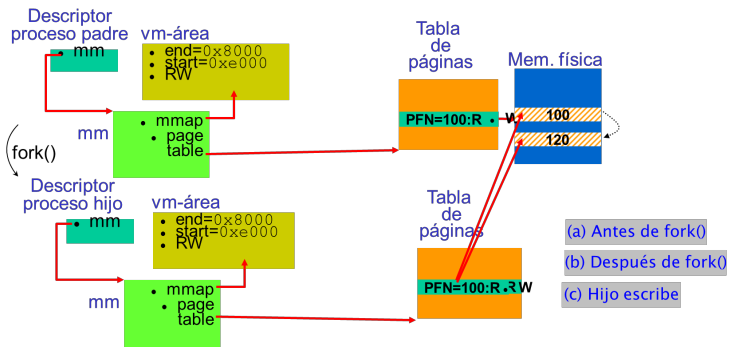
**vi) MECANISMO COW**

El mecanismo COW (Copy-On-Write) permite reducir las operaciones de copia en memoria y el número de copias de marcos. El mecanismo permite que varios procesos compartan un marco de memoria. Si alguno de los procesos intenta escribir en la página compartida COW, el kernel asigna una nueva copia de la página al proceso escritor y realiza la escritura.

Implementación del mecanismo: la página compartida COW se marca como de solo lectura, pero la vm-área correspondiente permite la escritura.

Cuando tengo dos procesos compartiendo una página y quiero escribir en ella, duplico la página, una para cada proceso, y le permito al proceso que escriba en la suya.

Si un proceso intenta escribir en una página compartida, salta una excepción, no se cambian los permisos, es decir, la página sigue sin permisos de escritura, aunque la región de memoria lo permita, entonces se hace una duplicación de la página. Esta funcionalidad permite retrasar la operación de copia hasta que sea completamente necesario, evitando los casos en los que no llega a serlo.



**vii) DEMANDA DE PÁGINA**

La demanda de página es la técnica mediante la cual la asignación dinámica de memoria se pospone hasta que el proceso direcciona una página y genera la excepción “falta de página”. Las razones por la que una página no está presente en memoria principal:

* La página nunca ha sido accedida por el proceso: su PTE está rellena de ceros.
* La página ha sido accedida pero su contenido se ha sacado a disco: su PTE no está rellena con ceros y el indicador Present está limpio (a 0).

Una página puede no estar en memoria, porque no la he cargado, o porque la he cargado previamente, pero he quitado el proceso. Una página que nunca ha sido cargada tiene su PTE a 0. Una página que ha sido cargada pero el SO la ha quitado tiene el bit de presente a 0, pero el resto de bit tiene algo distinto de cero.

En el caso 1, hay dos formas de cargar la página dependiendo de si la página está proyectada en un archivo de disco. Esto se determina consultando vma->vm\_ops->nopage:

* Si es no nulo: apunta a la función que carga la página.
* Si es nulo: se trata de una proyección anónima (no proyectada a ningún archivo).

Para la proyección anónima y de cara a alcanzar el nivel C2 de seguridad, se utiliza la página cero (página asignada estáticamente en la inicialización y almacenada en el sexto marco de página). Esta página no se puede escribir, de cara a disparar el mecanismo COW cuando se intenta.

La función nopage de la vma nos da información de cómo localizar la página en función de la PTE.

**viii) RECLAMACIÓN DE PÁGINAS**

La demanda de páginas no establece un medio para liberar los marcos no necesitados. ¿Qué ocurre si el SO se queda sin memoria libre?  La reclamación de páginas mantiene un depósito mínimo de marcos que permite al kernel recuperarse de condiciones en las que exista poca memoria libre. La reclamación se activa de forma:

* Síncrona: Si falla una asignación de memoria
* Asíncrona: La hebra kswapd descubre que el número de marcos libres cae por debajo de cierto nivel (marca baja de agua – page\_low).

La reclamación se detiene cuando se alcanza cierto nivel de páginas libres (marca alta de agua (page\_high)).

Se usa el mecanismo de reclamación de páginas, cuando hay poca memoria libre el SO tiene que recoger páginas de algún sitio.

Cuando el SO ve que queda poca memoria libre, tiene que recoger memoria de algún sitio. La idea es que el mecanismo de gestión de memoria defina dos umbrales indicando que cuando la cantidad de memoria libre cae por debajo de cierto umbral se dispara el algoritmo de reclamación de página. Hay una hebra kernel que busca las páginas que no están siendo referenciadas por los procesos y se las quita (las marca como libres y las deja en RAM). Hace esto hasta que la memoria libre esté por encima de cierto umbral, entonces la hebra kernel se detiene.

Si el proceso al que le he quitado una página vuelve a referenciarla y está en RAM, simplemente rehago la traducción.

La selección de un buen algoritmo de reclamación de páginas es un trabajo empírico. El principal objetivo es ajustar los parámetros para obtener un buen rendimiento. El algoritmo es una mezcla de heurísticas:

* Selección cuidadosa del orden en el que examinan las cachés.
* Ordenar las páginas por edad (LRU)
* Determinar el estado de las páginas (no sucias frente a sucias).

Con la asignación y reclamación de páginas el SO tiene que ser capaz de tener memoria para los procesos. Si se salta el intercambio de manera periódica no hay suficiente memoria RAM.

**RECLAMACIÓN: FUNCIÓN**

La función try\_to\_free\_pages() se encarga de la reclamación de páginas. El sistema divide todas las páginas de memoria dinámica en dos listas LRU:

* Lista activa – incluye a las páginas que han sido recientemente accedidas.
* Lista inactiva – contiene páginas no accedidas desde hace cierto tiempo.

La idea es “robar” las páginas de la lista inactiva, aunque periódicamente se realiza un filtrado entre las dos listas. Se ejecuta con una hebra, intentando dividir las páginas en varias listas, una lista de páginas activas e inactivas, una implementación variante del LRU.

**CACHÉS**

Existen diferentes cachés desde el punto de vista de memoria. La ventaja del mecanismo de proyección es que cuando implementa el mecanismo de proyección de archivos para todos los archivos, los algoritmos de gestión de memoria se simplifican y utiliza algoritmos de gestión de caché.

* Caché de páginas – Todos los accesos a archivos a través de read(), write()  y mmap() se realizan a través de esta caché.
* Caché de búferes – Almacena los metadatos de los archivos para hacer más rápida su recuperación. Información que describe el archivo proyectado. Contiene metadatos de sistemas de archivos.
* Caché de intercambio – Evita condiciones de carrera entre procesos que intentan acceder a páginas que han sido sacadas de la RAM (una página compartida puede estar presente para un proceso y sacada de memoria para otro).

