ChatGPT

Memoria Virtual – del Concepto a la Simulación

Memoria Virtual vs Memoria Física

La **memoria virtual** es una abstracción que da a cada proceso su propio espacio de direcciones independiente de la memoria física real. En este modelo, el procesador genera direcciones lógicas (virtuales) que son traducidas por el hardware/sistema operativo a direcciones físicas en RAM. Por tanto, cada programa «ve» una memoria contigua e ilimitada, aunque sus páginas puedan residir en RAM o en disco [1] [2]. En cambio, la **memoria física** es el hardware real (RAM) disponible en el sistema. El enlace entre ambas es que el OS mantiene tablas que mapean páginas virtuales a marcos físicos, dando la ilusión de mayor memoria y aislamiento de procesos [2] [1].

Paginación y Tablas de Páginas

En la paginación, el espacio de direcciones virtuales se divide en **páginas** de tamaño fijo (por ejemplo 4KB) y la memoria física en **marcos** (frames) del mismo tamaño. Cada dirección virtual se compone de un número de página (bits altos) y un desplazamiento dentro de la página (bits bajos). El número de página indexa la **tabla de páginas** del proceso: ésta contiene por cada página virtual el número de marco físico asignado (o un indicador de ausencia) [3] [4]. El OS usa esta tabla para calcular la dirección física: sustituye el número de página por el número de marco obtenido y conserva el desplazamiento. Así, la tabla de páginas implementa una función que "traduce" páginas virtuales a marcos físicos [3] [4]. En máquinas modernas se usan tablas multinivel o invertidas para ahorrar memoria, pero el principio básico es siempre el mismo: cada proceso mantiene su propio mapeo virtual→físico. Además, el acceso a estas tablas suele acelerarse con una **TLB** (Translation Lookaside Buffer): un pequeño caché asociativo donde el MMU busca primero la entrada correspondiente. Si la dirección está en la TLB (hit), se obtiene el marco inmediatamente; en caso contrario (miss) se consulta la tabla de páginas en memoria [5].

Manejo de Fallos de Página

Cuando un proceso accede a una página virtual que no está en RAM, se produce un **fallo de página** (page fault) y el CPU "trampa" al SO. El sistema operativo debe entonces traer la página faltante desde el respaldo en disco. En términos generales, el manejo de un fallo de página incluye:
1. El procesador detecta el fallo de página y genera una excepción (trap) al kernel.
2. El SO verifica que la dirección sea válida en el espacio de direcciones del proceso.
3. El SO elige un marco físico libre (o selecciona una página víctima para expulsar). Si el marco víctima contiene datos modificados (bit sucio), primero escribe esos datos al respaldo en disco.
4. Luego lee la página requerida desde el respaldo (swap) al marco seleccionado.
5. Actualiza la tabla de páginas del proceso (marca la página como presente, limpia el bit sucio, ajusta permisos, etc.) y **reasigna** la instrucción fallida para reintentarse [6].

En palabras de Tanenbaum: «Esta trampa se llama fallo de página. El sistema operativo selecciona un marco poco usado y escribe su contenido a disco (si no está ya allí). Luego carga la página referenciada desde el disco al marco, actualiza el mapa y reanuda la instrucción capturada» [6].

Almacenamiento de Respaldo y Bit Modificado

El **almacenamiento de respaldo** (backing store) es el espacio en disco donde se guardan las páginas que no caben en RAM. Lo común es usar una **partición de intercambio (swap)** dedicada, separada del sistema de archivos usual, de modo que las páginas expulsadas se escriben en bloques referenciados directamente al inicio de dicha partición [7]. Al iniciar, el kernel reserva áreas en swap iguales al tamaño de cada proceso; a medida que un proceso ejecuta, sus páginas se llevan a swap conforme es necesario [8]. Esto reduce la sobrecarga de gestión: no es necesario un sistema de archivos, solo se usan desplazamientos a la partición de swap [9].

Cada entrada de tabla de páginas incluye un bit **modificado** (dirty) que indica si la página fue escrita (alterada) desde que fue cargada. Si se expulsa una página y este bit está activado, el sistema la escribe a disco; si no (página «limpia»), simplemente se descarta, pues en disco ya existe una copia válida [10]. En palabras de Tanenbaum, el bit modificado «se pone automáticamente cuando se escribe a la página; si está marcada (dirty), debe escribirse a disco, en caso contrario puede abandonarse, ya que la copia en disco sigue siendo válida» [10]. Este bit, a menudo llamado **bit sucio**, evita escrituras innecesarias y mejora el rendimiento del swapping.

Reemplazo de Página FIFO

El algoritmo de reemplazo **FIFO** (First-In, First-Out) es uno de los más sencillos. El SO mantiene una cola con las páginas actualmente en memoria, ordenadas por llegada; la página más antigua (la que llegó primero) está al frente. Ante un fallo de página, se elimina la página del frente y se inserta la nueva página al final de la cola [11]. Es decir, siempre se expulsa la página que lleva más tiempo en memoria. Si bien este método es fácil de implementar, tiene desventajas: la página más vieja puede seguir siendo útil, por lo que a menudo expulsa páginas activas. Como concluye Tanenbaum, «cuando se aplica a los computadores ocurre el mismo problema: la página más antigua aún puede ser útil. Por ello, el FIFO en su forma pura rara vez se usa» [12].

Traducción de Direcciones

Cada acceso a memoria (lectura o escritura) implica traducir la dirección virtual a una física. Este proceso lo realiza la **Unidad de Manejo de Memoria (MMU)**, usando la tabla de páginas del proceso y la TLB. En resumen: la dirección virtual se divide en número de página y desplazamiento, el MMU busca el número de página en la TLB, y si lo encuentra obtiene el número de marco físico al instante [5]. Si no hay entrada en la TLB, el hardware consulta la tabla de páginas en memoria (posiblemente ocasionando varios accesos a RAM), copia la entrada al TLB y luego continúa. De este modo se logra que la *traducción de direcciones* sea eficiente: la TLB actúa como cache de traducciones recientes, evitando costosas consultas a tabla en cada acceso [5]. En última instancia, el marco obtenido se concatena con el desplazamiento para formar la dirección física final, la cual es enviada al bus de memoria para completar el acceso.

Aislamiento de Procesos

La memoria virtual también **aísla** los procesos entre sí. Cada proceso tiene su propia tabla de páginas y, por tanto, su propio espacio de direcciones virtual. Como señala un recurso educativo, «cada proceso tiene su propio espacio de direcciones… (y) un proceso no puede acceder al espacio de direcciones de otro» [13]. Esto significa que un proceso no puede leer ni escribir en la memoria de otro (salvo que el sistema operativo explícitamente lo permita mediante mecanismos de compartición). De esta forma el

hardware y el SO garantizan protección: si un proceso intenta acceder fuera de sus páginas mapeadas o con permisos inválidos, se genera una excepción de protección o segmentación, impidiendo que un proceso corrompa la memoria ajena [13]. La compartición (p.ej. mapeo de una misma página en dos procesos) solo ocurre cuando el SO lo programa, como en bibliotecas compartidas o canales IPC.

Conexión con Simulación de Software

Los conceptos de memoria virtual se pueden ilustrar mediante **simuladores de sistemas operativos**. Por ejemplo, en un laboratorio de la Universidad de Cornell se provee un simulador MIPS donde la parte «hardware» ya implementa la memoria física y la traducción de direcciones: para cada carga/almacenamiento el simulador recorre la tabla de páginas ubicada en el registro c0r4 y traduce la dirección virtual a física [14]. El trabajo restante recae en el estudiante, que debe simular el sistema operativo: implementar funciones como `create_address_space()`, `map_page()`, y `pfault()` para crear espacios de direcciones, mapear páginas y manejar fallos de página [15]. En otras palabras, el simulador ya sabe cómo traducir direcciones, pero el estudiante debe escribir el código que el sistema operativo real ejecutaría (asignar marcos, seleccionar páginas a expulsar, etc.) [15]. Plataformas educativas como **Pintos** (Stanford) o **Nachos** (UC Berkeley) siguen una idea similar: se pide al alumno programar algoritmos de reemplazo de páginas, manejo de errores y estructuras de memoria para observar el comportamiento de la VM en la práctica.

Referencias: Autores reconocidos definen y describen detalladamente estos conceptos (p.ej. Tanenbaum [2] [6], Silberschatz [1]). Los ejemplos de simuladores estudiantiles (Cornell, Stanford) demuestran cómo la teoría se implementa por software [14] [15]. Los extractos citados ilustran la implementación de tablas de páginas, detección de fallos, bits de control y polticas de reemplazo en sistemas reales.

---

[1] Figure 9.01
https://www.os-book.com/OS8/os8j/slide-dir/PDF-dir/ch8.pdf

[2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] 0133592480.pdf
https://csc-knu.github.io/sys-prog/books/Andrew%20S.%20Tanenbaum%20-%20Modern%20Operating%20Systems.pdf

[13] Microsoft PowerPoint - lect.14.OS2.ppt
https://web.stanford.edu/class/archive/ee/ee108b/ee108b.1082/handouts/lect.14.OS2.pdf

[14] [15] CS3410 Spring 2014 Lab 4
https://www.cs.cornell.edu/courses/cs3410/2014sp/lab/lab4/lab4.html