Ejercicio 3. *Hardware y software de memoria virtual en un sistema moderno.*

a. **Indique cómo son los campos de una dirección virtual en un procesador Intel/AMD de arquitectura AMD64. Considere un sistema Linux actual en ese sistema que usa tabla de páginas de 4KB.**

En una arquitectura AMD64 (x86-64), bajo Linux actual y con un tamaño de página de 4 KB (4096 bytes), una dirección virtual de 64 bits está dividida en varios campos jerárquicos, que se usan para traducir la dirección virtual a una dirección física mediante el uso de tablas de páginas multinivel.

Responder:
¿Cuál es el tamaño de cada tabla de páginas o directorio? Estas son llamadas (AMD programmer manual): Page Map Level 4 Table, Page Directory Pointer Table, Page Directory Table, Page Table.

La Unidad de Gestión de Memoria (MMU) divide la dirección virtual en campos que se usan para navegar por cuatro niveles de tablas de páginas:

- Nivel 4: PML4 – Page Map Level 4
  - Tiene 512 entrada de 8 bytes, siendo su tamaño total 4096 bytes (4 KB)
- Nivel 3: PDPT – Page Directory Pointer Table
  - Tiene 512 entrada de 8 bytes, siendo su tamaño total 4096 bytes (4 KB)
- Nivel 2: PD – Page Directory Table
  - Tiene 512 entrada de 8 bytes, siendo su tamaño total 4096 bytes (4 KB)
- Nivel 1: PT – Page Table
  - Tiene 512 entrada de 8 bytes, siendo su tamaño total 4096 bytes (4 KB)

Cada nivel de tabla de páginas (PML4, PDPT, PD, PT) mide 4 KB y contiene 512 entradas de 8 bytes.
Esto hace que el diseño sea eficiente y escalable: cada tabla cabe exactamente en una página de memoria

¿Cuántos bits realmente se utilizan de la dirección virtual?
En la arquitectura AMD64 (x86-64) con Linux actual y páginas de 4 KB, se utilizan actualmente 48 bits de los 64 bits disponibles en la dirección virtual.

Aunque las direcciones virtuales son de 64 bits, el hardware (procesador y sistema operativo) solo usa 48 bits significativos para direccionar memoria. Esto se debe a:

- Limitaciones de diseño para mantener compatibilidad y eficiencia.
- Las direcciones virtuales deben estar "canónicamente sign-extended", lo que significa que los bits 63–48 deben ser iguales al bit 47 (signo extendido).

¿Cuál es el tamaño máximo de memoria virtual que un proceso puede utilizar?

Como ya vimos:

- Se utilizan 48 bits de los 64 disponibles en una dirección virtual.
- Eso permite un espacio de direcciones virtuales de:

  2^48 bytes=256 TB (terabytes)

Entonces un proceso puede usar como máximo 256 TB de memoria virtual en un sistema AMD64 estándar con soporte de 48 bits de direcciones virtuales. Pero esto no significa que un proceso tenga 256 TB de RAM disponible, sino que puede mapear hasta 256 TB de espacio virtual, que incluye RAM, memoria compartida, archivos mapeados, bibliotecas dinámicas, stacks, heaps, etc.

En la práctica, Linux impone límites más bajos mediante configuraciones del kernel (ulimit, vm.max_map_count, etc.) y la cantidad de memoria física disponible.

¿Cómo es posible que un proceso funcione si el tamaño de memoria virtual que utiliza es mayor que la memoria física?

La clave para entender esto es el funcionamiento de la memoria virtual y el concepto de paginación que usa el sistema operativo moderno.

La memoria virtual es un espacio de direcciones abstracto que el sistema operativo le da a cada proceso. A cada proceso le parece que tiene un espacio de memoria continuo y privado, aunque en realidad:

- No todo está en RAM al mismo tiempo
- Puede exceder el tamaño de la memoria física

La memoria virtual se divide en paginas (normalmente de 4KB) y se mapea a marcos de página (frames) en la memoria física

- Solo las páginas que realmente se usan se cargan en RAM
- Las demás pueden estar en disco (en el archivo de swap) o ni siquiera haberse reservado fisicamente todavia

Cuando la RAM se llena, el sistema operativo puede mover páginas inactivas a un área del disco llamada swap

- Esto permite liberar RAM para otras páginas más activas
- Cuando se necesite de nuevo una página que está en el swap, se trae de vuelta a RAM (con cierto costo en tiempo)

Sea un proceso que utiliza un total de 128MB de memoria en este sistema. ¿Cuándo el proceso comienza a ejecutarse, cuántas páginas necesita cargar el SO para que el proceso comience a ejecutarse?

Cuando un proceso comienza a ejecutarse en un sistema moderno (como Linux sobre arquitectura AMD64 con paginación de 4 KB), no necesita cargar todas sus páginas inmediatamente. El sistema operativo usa una técnica llamada "demanda de paginación"

(*demand paging*), que permite cargar sólo las páginas necesarias para comenzar la ejecución.

Las páginas que se cargan al inicio serian las que contiene el código que se ejecuta primero, las necesarias para el stack y heap inicial o las que contienen las tablas de símbolos y estructuras del programa. Esto puede ser del orden de unas decenas a cientos de páginas, dependiendo del binario y las bibliotecas dinámicas.

En conclusión, aunque el proceso use 128 MB de memoria virtual, el sistema operativo solo carga unas pocas páginas (las necesarias para comenzar la ejecución, típicamente cientos de KB) al principio. El resto se carga bajo demanda, conforme el proceso accede a ellas

¿Qué significa fragmentación interna?

La fragmentación interna es un tipo de desperdicio de memoria que ocurre cuando el sistema asigna más memoria de la necesaria a un proceso o estructura, y el espacio sobrante dentro de esa asignación no se puede usar para otra cosa.
Sucede principalmente por:

- Tamaño fijo de paginas o bloques: si un proceso necesita 6 KB y el sistema asigna bloques de 4 KB, necesita dos paginas (8 KB en total), desperdiciando 2 KB
- Asignación alineadas o redondeadas: cuando un proceso pide por ejemplo 33 bytes y el sistema asigna 64 bytes

Existe la fragmentación interna y externa:

- Interna: espacio dentro de bloques asignados que no se usa
- Externa: espacio libre entre bloques que no se puede usar por estar fragmentado

b. Sea un proceso que referencia una dirección de memoria virtual. Describa un posible escenario para cada una de las siguientes situaciones (si no es posible tal escenario, explique por qué):

i. fallo del TLB sin fallo de página;

El proceso accede a una dirección virtual cuya traducción *no* está en el TLB (es decir, *fallo de TLB*). La página está cargada en memoria física, por lo que el sistema consulta en la tabla de páginas, encuentra la traducción, y la carga en el TLB.

ii. fallo del TLB con fallo de página;

El proceso accede a una dirección virtual, la cual no está en la TLB (fallo de TLB) y tampoco está mapeada en memoria física (fallo de pagina). El sistema operativo entonces debe interrumpir el proceso, traer la página desde el disco (swap o archivo ejecutable) y actualizar la tabla de páginas y luego la TLB

iii. acierto del TLB sin fallo de página;

El proceso accede a una dirección virtual que ya tiene su traducción cargada en el TLB (acierto de TLB) y la página está en memoria (sin fallo de página)

iv. acierto del TLB con fallo de página.

No es posible. Si el TLB tiene una traducción válida, significa que la página está mapeada en memoria física. Si no está en memoria, la entrada no puede estar presente en la TLB, porque se elimina cuando la página se intercambia al disco.

v. ¿Puede suceder también un fallo de caché?

Si. La TLB y la caché son mecanismos distintos, en donde la tLB traduce las direcciones virtuales a físicas y la caché almacena datos de memoria recientes. Puede haber un acierto en el TLB pero un fallo en la caché si los datos no estaban en la caché L1/L2 y se deben buscar en la RAM