¿Cuánto espacio en memoria ocupan las tablas de paginación inicial del kernel, definidas en entrypgdir.c? Considerar tanto "page directory" como "page tables". *

Nota: tener en cuenta la implementación sin large pages.

- ( ) No ocupa memoria, porque viene con la imagen del kernel
- (•) Una página (4Kib)
- ( ) Dos páginas (8Kib)
- ( ) Tres páginas (12Kib)

¿Cuánto espacio en memoria ocupan las tablas de paginacón inicial del kernel, definidas en entrypgdir.c, luego de ser modificado para usar large pages? *

- ( ) No ocupa memoria, porque viene con la imagen del kernel
- (•) Una página (4Kib)
- ( ) Dos páginas (8Kib)
- ( ) Tres páginas (12Kib)

¿Cuál es la complejidad computacional de page_alloc? *

Nota: P es la cantidad de páginas físicas disponibles

- (•) O(1)
- ( ) O(P)
- ( ) O(P^2)
- ( ) No se puede determinar, porque no sabemos cuánta memoria física hay en primera instancia.

¿Qué combinación de flags logra una "page table entry" que sea de sólo lectura para kernel y usuario? *

Nota: asumir PTE_P en todos los casos

- ( ) Ningún flag es necesario, todas las páginas son accesibles con esos permisos por defecto.

- (●) PTE_U
- ( ) PTE_U | PTE_W
- ( ) PTE_U | PTE_K

¿Qué combinación de flags logra una "page table entry" que sea de sólo lectura para el usuario, pero de lecto-escritura para el kernel? *

Nota: asumir PTE_P en todos los casos

- ( ) PTE_U | PTE_W
- ( ) PTE_W
- ( ) (PTE_U & !PTE_W) | (PTE_K & PTE_W)
- (●) No es posible

¿Cuánto tamaño ocupa una "page table entry" en la arquitectura x86 que emplea JOS? *

- ( ) 16 bits
- (●) 32 bits
- ( ) 40 bits
- ( ) 44 bits

¿Cuántas entradas tiene una "page table" en la arquitectura x86 que emplea JOS? *

- ( ) 512
- (●) 1024 (1Ki)
- ( ) 2048 (2Ki)
- ( ) 4096 (4Ki)

¿Qué otro nombre le darían a la siguiente macro? #define UNA_MACRO(x) (x >> 12) *

- ○ PGSIZE
- ◉ PGNUM
- ○ PTX
- ○ page2pa

¿Cómo se reserva memoria para el arreglo pages? *

- ○ La memoria viene pre-alocada en la imagen del kernel
- ○ Se utiliza page_alloc
- ◉ Se utiliza boot_alloc
- ○ Se utiliza boot_map_region

¿Qué flag hay que marcar y en qué parte cuando se quiere crear una large page? *

- ○ Se marca la flag PTE_P en el PDE
- ○ Se marca la flag PTE_P en el PTE
- ◉ Se marca la flag PTE_PS en el PDE
- ○ Se marca la flag PTE_PS en el PTE

La función page_init debe aumentar el campo pp_ref de cada página que agrega al arreglo pages: *

- ◉ Verdadero
- ○ Falso

¿Por qué las direcciones de cada página (i.e. la primera dirección) tienen siempre los últimos 12 bits en cero? *

- ○ Es una convención
- ○ Es una decisión de JOS
- ○ Es una consecuencia de toda arquitectura de 32 bits
- ◉ Es una consecuencia del tamaño de las página

¿En qué ring se ejecuta el bootloader de JOS? *

Nota: recordar que ring 3 es el menos privilegiado, mientras que ring 0 es el más privilegiado

- ○ Ring 3
- ○ Ring 0
- ◉ No se ejecuta en ningún ring, todavía no corrió el kernel y no los configuró

La función boot_map_region no necesita incrementar el pp_ref de cada página que mapea. Esto es porque: *

- ○ Las regiones para las cuales se utiliza, nunca fueron pedidas con page_alloc
- ◉ Se realiza después, por fuera de la función
- ○ boot_map_region no opera con páginas

La MMU verifica los flags de ambas entradas (page directory y page tables) *

- ○ Verdadero
- ◉ Falso

La cantidad de memoria física RAM en una arquitectura x86 de 32 bits son siempre 4 Gb (como máximo), aunque JOS solamente mapee 256 Mb. *

Nota: tener en cuenta el mapa de memoria física

- (●) Verdadero
- ( ) Falso

Los números de página de un “page directory” y de las “page tables” son siempre de páginas virtuales *

- ( ) Verdadero
- (●) Falso

¿Cuál es el propósito de la TLB? *

- ( ) Permite hacer que las tablas de paginación sean más pequeñas.
- ( ) Reduce los tiempos de acceso a memoria de las instrucciones (i.e. aumenta la velocidad de todo acceso a memoria).
- ( ) Contiene los registros utilizados para segmentación (e.g. cs en arquitectura x86).
- (●) Reduce los tiempos promedio de traducción de direcciones.

¿Qué ocurre con las entradas en la TLB al llamar a page_remove? *

- ( ) Nada, porque JOS no configura la TLB
- (●) La entrada en la TLB correspondiente a la página removida debe ser invalidada manualmente
- ( ) Nada, la entrada en la TLB correspondiente se invalidará sola cuando la MMU intente acceder a esa misma direción

¿Cómo sabe la MMU donde encontrar el "page directory" para iniciar una traducción de direcciones por paginación? *

- ( ) Siempre va a buscar a un dirección física fija, definida por la arquitectura
- ( ) Se obtiene la página física a partir del registro eip al momento de traducir
- (●) Se obtiene la página física a partir del registro cr3

- Depende del tipo de acceso (e.g. datos, instrucciones, etc), se utilizará el registro de segmento correspondiente

Crea tu propio formulario de Google
Notificar uso inadecuado