¿Cómo queda configurado el handler de la interrupción 75? *

SETGATE(idt[75], 1, GD_KT, &handler_75, 3);

- (●) Permite interrupciones anidadas. Permite interrupciones por software en modo usuario.
- ( ) No permite interrupciones anidadas. Permite interrupciones por software en modo usuario
- ( ) Permite interrupciones anidadas. No permite interrupciones por software en modo usuario
- ( ) No permite interrupciones anidadas. No permite interrupciones por software en modo usuario

La instrucción "iret"... *

- ( ) Puede ser ejecutada en ring 3 para volver al kernel
- ( ) Cambia atómicamente el eip, el cs y todos los registros de propósito general
- (●) Funciona igual que la instrucción "ret"
- ( ) Permite cambiar el stack que se está usando

El formato y orden del "struct Trapframe" se define por: *

- ( ) Convención de JOS
- ( ) La arquitectura cuando se realiza un cambio de contexto
- (●) La arquitectura cuando se realiza un cambio de contexto y JOS
- ( ) Ninguna de las anteriores

El anidamiento de interrupciones no se implementa en JOS porque no está el soporte en la arquitectura x86 para hacerlo *

- ( ) Verdadero
- (●) Falso

El copiado del binario al espacio del proceso hijo se hace teniendo en cr3 el "page directory" del mismo porque es la única forma de hacerlo *

- ○ Verdadero
- ◉ Falso

El formato del binario ELF tiene una parte fija de metadatos y una parte variable donde se definen cada uno de los "program headers" *

- ◉ Verdadero
- ○ Falso

Sobre entrypoint donde arranca cada uno de los programas de usuario: *

- ◉ Es una constante definida por el kernel (UENTRY)
- ○ Cada procceso de usuario la define al momento de compilarse
- ○ El kernel calcula dinámicamente un nuevo entrypoint al momento de crear un proceso
- ○ Está dado por la arquitectura

JOS decide colocar los binarios de los programas de usuario concatenados al binario del kernel porque: *

- ○ Es una convención del kernel
- ◉ Es la única opción viable ya que todavía no existe un sistema de archivos
- ○ Lo define la arquitectura x86
- ○ Es la forma más fácil de hacerlo

La dirección del stack en donde se tiene que ejecutar un "handler" de interrupción: *

- ◉ Es una constante definida por la arquitectura

- ○ Es una constante que configura el kernel
- ○ Es una variable que calcula el kernel al crear el proceso
- ○ Es una variable que se calcula cuando se recibe la interrupción

¿Cuál es el mecanismo por el que se implementan las syscalls en JOS? *

- ○ Se utiliza instrucción privilegiada de x86 llamada call_syscall
- ○ Como el kernel está en el pgdir del usuario, alcanza con una llamada a la función syscall del kernel desde userspace
- ○ Se utiliza segmentación, el usuario cambia el valor del registro "cs" para "entrar" en el kernel (i.e. mov $GD_KT %cs)
- ◉ Se utiliza una interrupción generada por software

En la arquitectura x86, clasifique los registros según cómo se modifican en el cambio de contexto. *

	Lo modifica la arquitectura	Lo modifica JOS
eip	☐	☑
cr3	☐	☑
ebx	☑	☐
ds	☑	☐
ss	☑	☐

Juan configuró el handler de las syscalls como se indica. ¿Qué es lo que va a ocurrir al correr las pruebas? Nota: suponer el resto de la implementación no tiene errores. *

SETGATE(idt[T_SYSCALL], 0, GD_KT, KERNBASE, 3);

- ○ La función env_pop_tf va a fallar, previniendo que se ejecute un proceso de usuario

- (●) El handler de las syscalls está bien configurado, va a pasar las pruebas!
- ( ) El programa de usuario divzero va a fallar
- ( ) El programa de usuario buggyhello va a fallar

Para un proceso nuevo ¿dónde se configura cada uno de estos registros? *

	En env_alloc	En env_create	En load_icode
eip	☐	☐	☑
esp	☑	☐	☐
eax	☐	☑	☐
cs	☑	☐	☐
eflags	☐	☐	☑

¿Cuál/les de las siguientes opciones es/son verdad respecto a la implementación final del TP2 de JOS? *

- ☐ Un proceso de usuario puede leer los contenidos de KERNBASE abusando de la syscall sys_cputs
- ☐ Un proceso de usuario puede monopolizar todo el tiempo de CPU
- ☑ Un proceso de usuario puede usar la macro ENV_CREATE para crear un nuevo proceso
- ☑ Podemos correr más de un proceso de usuario a la vez

Todo proceso de usuario tiene al kernel mappeado en su propio espacio de direcciones *

- (●) Verdadero
- ( ) Falso

¿Qué representan los valores GD_KT, GD_KD, GD_UT y GD_UD? *

- ○ Son constantes que resuelven al número de ring (i.e. "3" o "0" donde corresponda)
- ○ Son constantes definidas por la arquitectura, JOS sólo las usa
- ◉ Son índices en la tabla de segmentos global (gdt), configurada por JOS
- ○ Son índices en la tabla de segmentos de interrupciones (idt), configurada por JOS

¿Por qué, conceptualmente, es necesario tener dos macros: TRAPHANDLER_NOEC y TRAPHANDLER? *

- ○ Para distinguir entre interrupciones por hardware e interrupciones por software
- ○ Para distinguir entre excepciones e interrupciones
- ◉ Porque algunas interrupciones tienen un código de error asociado
- ○ Porque algunas interrupciones no son recuperables (e.g. división por zero)

¿Dónde almacena la arquitectura x86 el nivel de priviegio actual? *

- ○ En el registro eflags
- ○ En la idt
- ○ En el registro cpl
- ◉ En el registro cs

Para pasar del kernel al proceso de usuario se realiza un move del "entry point" del binario asociado, al registro "eip" (e.j: mov entry_poiny, %eip) *

- ○ Verdadero
- ◉ Falso

Si en un proceso de usuario, se realiza una desrrefencia de un puntero a una dirección que no está mapeada, el mecanismo que genera la excepción es: *

- [ ] El kernel por medio de "user_mem_check"
- [x] La MMU
- [ ] La librería de usuario lo chequea y genera una interrupción por software
- [ ] El compilador, al generar el programa de usuario