SOA2 (31/03/2025)

Nombre:
DNI:

Primer control de teoría

Responde brevemente a las siguientes preguntas justificando tus respuestas. Una respuesta sin justificar no será dada como válida. Todas las preguntas tienen la misma puntuación.

El ataque Meltdown surgió hace unos 7 años y se basaba en abusar de la latencia de las excepciones en un procesador fuera de orden. Con este ataque, se podía leer la memoria de sistema desde modo usuario y así poder robar información comprometida.

La mitigación que se propuso fue que no se pudiesen traducir direcciones lógicas de sistema cuando el procesador estaba en modo usuario. Para ello, se modificaron las tablas de páginas de los procesos para que no incluyesen las traducciones de direcciones lógicas a físicas del kernel y se creó una nueva tabla de páginas, única en el sistema, con las traducciones de direcciones lógicas a físicas del kernel que se utilizaría, únicamente, en modo de máximos privilegios. A esta nueva tabla de páginas se la llamó OSPT.

Pregunta 1
¿Cuándo se tiene que crear la OSPT?

Pregunta 2
Por desgracia, hay una parte del código del kernel que se tiene que mapear siempre en la tabla de páginas de usuario del proceso. ¿Qué parte del código es ésta?

Pregunta 3
Pon un ejemplo de código que mapee una de las funciones de kernel de la pregunta anterior en la tabla de páginas del proceso Init (InitTP).

Aunque la implementación, a priori, parece sencilla, obliga a hacer cambios más profundos tanto en la parte de usuario como en la parte de kernel. Como veréis a continuación, se tendrán que modificar los wrappers, los handlers y las rutinas de servicio.

Hablando de wrappers…

Pregunta 4
¿Cuál es la finalidad de los wrappers?

Pregunta 5
¿De qué dos formas se pueden devolver resultados desde el kernel a los wrappers?

Nuestro sistema operativo implementa un único punto de entrada al sistema. Todos los wrappers utilizan la instrucción INT.

Pregunta 6
¿Qué ventajas ofrece el hecho de tener un único punto de entrada al sistema frente a tener varios?

Pregunta 7
¿Es compatible la instrucción sysenter con tener más de un único punto de entrada?

Es momento de empezar a modificar código. Empezaremos con los handlers.

Pregunta 8
¿Cuántos handlers tendremos que modificar en un sistema operativo con un único punto de entrada y que se ejecuta en una máquina con un procesador compatible con X86?

Empezaremos con los handlers de las interrupciones hardware. Un típico handler, como el de la interrupción de reloj, podría ser:

```
SAVE_ALL
call clock_routine
EOI
RESTORE_ALL
IRET
```

Pregunta 9
¿Por qué el handler tiene que escribirse en ensamblador?

Tal como está, este handler genera un page fault al intentar ejecutar la primera instrucción de SAVE_ALL. Curiosamente, el procesador al ejecutar la instrucción INT está accediendo a estructuras en memoria del sistema, pero estos accesos no generan page fault incluso si la tabla de páginas actual es la de usuario (que no tiene mapeado el kernel). Pero vamos por pasos:

Nombre:
DNI:

Pregunta 10
Al producirse una interrupción, ¿qué estructuras hardware son accedidas antes de ejecutar la primera instrucción del handler?

Cuando el sistema operativo prepara los registros que apuntan a estas estructuras, siempre tiene que poner la dirección física de estas estructuras ya que el procesador utiliza la dirección física para poder acceder directamente a donde están situadas estas estructuras sin tener que traducir direcciones lógicas. Por eso, el sistema operativo tiene que saber calcular la dirección física de estas estructuras a partir de su dirección lógica.

Pregunta 11
Escribe el código necesario para calcular la dirección física de una de las estructuras hardware anteriores. Puedes suponer que siempre existe una variable cuyo nombre coincide con el nombre de la estructura y que contiene la dirección lógica del inicio de la estructura.

Aunque haciendo un poco de memoria, en el código de ZeOS, en ningún momento se han tenido que calcular las direcciones físicas de ninguna estructura hardware ni de sistema. Esto es raro ya que el procesador sí que las necesita para poder acceder a las estructuras.

Pregunta 12
¿Cómo es posible que en el código de ZeOS no se tengan que calcular las direcciones físicas asociadas a estructuras hardware si el procesador las necesita?

Ahora si, vayamos al page fault causado por la primera instrucción de SAVE_ALL.

Pregunta 13
¿Qué es lo que causa este page fault?

Para solucionar el problema anterior, es necesario cambiar el espacio de direcciones al entrar al handler (para poner la tabla de páginas de sistema) y restaurar la tabla de páginas del proceso antes de salir del handler. Para, al principio del handler, cambiar a la tabla de páginas del sistema, implementamos, en ensamblador, el siguiente código:

```
movl OSTPaddress, %eax
movl %eax, %cr3
```

En la que OSTPaddress es una constante, calculada en tiempo de compilación, que contiene la dirección de la OSTP. Esta función no se ejecutará con call, sino que irá inlined en el código (es decir, haremos copy & paste de su código al sitio donde la necesitamos)

Pregunta 14
¿Se puede poner el código de la función set_cr3 antes del SAVE_ALL?

Pregunta 15
¿Se puede poner el código de la función set_cr3 después del SAVE_ALL?

Pregunta 16
¿Cambian las respuestas de las 2 preguntas anteriores si se aplica set_cr3 en el handler de las llamadas al sistema en vez de al handler de la interrupción de reloj?

Ahora si que tenemos un problema y necesitamos solucionarlo. Se tiene que modificar la función set_cr3 para que consiga, antes del SAVE_ALL, cambiar la tabla de páginas. Toca pensar.

Pregunta 17
Dibuja la pila de sistema antes de la primera instrucción del handler.

Con la respuesta a la pregunta anterior, tu cerebro y un poco de perspicacia, nos damos cuenta de que se puede utilizar un registro en concreto como registro temporal para cambiar cr3 sin perder ningún dato importante.

Pregunta 18
¿Qué registro es y por qué se puede utilizar de forma temporal con toda seguridad?

Pregunta 19
Reescribe la función set_cr3 para que se pueda ejecutar antes de SAVE_ALL y haga TODAS las modificaciones necesarias para que SAVE_ALL funcione perfectamente.

Nombre:
DNI:

Por cierto…

Pregunta 20
¿OSTPaddress es una dirección lógica o física? Es decir, ¿en cr3 se guarda la dirección lógica o física de la tabla de páginas?

Es momento de saltar a alto nivel planteando que parte del sistema tenemos que modificar al haber implementado este cambio. Iremos función a función.

Pregunta 21
Escribe el código de task_switch antes de implementar este cambio en la tabla de páginas.

Pregunta 22
¿Cómo se tiene que modificar el código de task_switch con la nueva implementación?

Pregunta 23
A parte de mejorar la seguridad del sistema, ¿este cambio supone alguna otra mejora en el task_switch?

Sys_fork es otra función que se tiene que modificar. En concreto, se tiene que modificar como se crea la imagen en memoria del proceso hijo.

Pregunta 24
¿Qué parte del espacio de direcciones del proceso no se hereda de padre a hijo con esta nueva implementación?

Pregunta 25
Escribe el bucle de copia de la zona de data+stack del proceso padre al proceso hijo. Esta zona empieza en la página lógica PAG_LOG_INIT_DATA y ocupa NUM_PAG_DATA páginas. Los identificadores de las páginas físicas que se utilizarán en el proceso hijo para esta zona están guardados en la variable local int pf[NUM_PAG_DATA] y ninguna de estas páginas físicas están mapeadas en el espacio de direcciones

lógico de ningún proceso. La tabla de páginas del padre se llamará TPP y la del hijo TPC y que todas las tablas de páginas son de un único nivel.

Pregunta 26
¿Se tiene que hacer, después del bucle anterior, un flush del TLB para eliminar los mapeos temporales que se han hecho para acceder a las páginas físicas del hijo?

Recordemos que sys_fork también tiene que modificar el contexto del proceso hijo para que pueda pasar a RUN a través de task_switch. Para ello, hacemos que ejecute ret_from_fork la primera vez que pasa a RUN.

Pregunta 27
¿Por qué interesa que se ejecute ret_from_fork?

Pregunta 28
Con la nueva implementación de las tablas de páginas, ¿se sigue necesitando ret_from_fork?

Pregunta 29
Dibuja el contenido de la pila de sistema del proceso hijo en sys_fork indicando que modificaciones se tienen que hacer para que sys_fork siga manteniendo su comportamiento, pero sin utilizar ret_from_fork. Fijate como hemos implementado el handler de la interrupción de reloj (Página 2).

El Señor Baka Baka, después de pasar los últimos 10 años de su vida hacinado en una prisión de Alemania por sus reiterados crímenes de lesa humanidad contra los estudiantes de SOA2, ha sido liberado. Como experto mundial en sistemas operativos y en armas de destrucción de materia cerebral, hemos decidido enseñarle esta implementación, la de tener una tabla de páginas para el

kernel para conocer su opinión. Después de pensarlo mucho y de darnos un tierno abrazo de alegría nos ha dicho: "Esta implementación, querido amigo, no solamente soluciona el problema, sino que, además, se puede ver que tiene una segunda ventaja: ahorra memoria guardando tablas de páginas si la tabla de páginas es de dos niveles.". Estupefactos, nos ponemos a hacer cálculos…

Teniendo en cuenta que la memoria usada del kernel del sistema operativo ocupa 10 MBs, que la parte de usuario de cada uno de los procesos en el sistema ocupan 1 MB, que un directorio tiene 1024 entradas, que cada una de las entradas del directorio y de la tabla de páginas ocupa 32 bits y que las páginas físicas son de 4 KBs…

Pregunta 30
¿Cuánta memoria nos ahorramos ocupada por tablas de páginas si se están ejecutando 10 procesos comparado con utilizar tablas de páginas de un solo nivel?

Después de realizar este cálculo te sientes muy satisfecho por el gran esfuerzo y mejora que has realizado en el sistema. Pero cuando menos te lo esperas, el Sr Baka Baka te pega una colleja y te grita: "No estés satisfecho porque ahora el sistema operativo va mucho más lento". Con ganas de llorar, te preguntas:

Pregunta 31
¿Por qué va más lento el sistema?

Información del Sistema Operativo
El sistema dispone de las siguientes rutinas:

- ***struct task_struct *current():*** Retorna la task_struct del proceso actual.
- ***int alloc_frame()*:** Reserva un frame de memoria física.
- ***void free_frame (int frame)***: Libera un frame de memoria.
- ***page_table_entry * get_PT(struct task_struct *t):*** Retorna la tabla de páginas del proceso *t*.
- ***page_table_entry * get_DIR(struct task_struct *t):*** Retorna el directorio de páginas del proceso *t*.
- ***set_CR3 (page_table_entry * dir):*** Sobreescribe el registro CR3 con el nuevo directorio de páginas *dir*, provoca una invalidación de la TLB.
- ***int get_frame (page_table_entry *pt, int logical_page)***: Retorna el frame asignado a una página lógica en la tabla de páginas de un proceso determinado.
- ***int set_ss_page (page_table_entry *pt, int logical_page, int frame):*** Asigna un *frame* a una página lógica de la tabla de páginas *pt*.
- ***int del_ss_page (page_table_entry *pt, int logical_page)*:** Borra una entrada de la tabla de páginas.