Tema 2 Syscalls

##COMO ACTUA LA CPU PARA ESCALAR DEL MODO USER AL MODO PRIVILEGIO##

El SO implementa el acceso a los recursos de la máquina. Por seguridad, aísla al usuario del código de bajo nivel de la máquina. Para eso, asigna determinado espacio seguro del disco al usuario, y implementa políticas de alocatamiento.

De esta manera crea los niveles de privilegio: que determinan quién puede ejecutar determinadas instrucciones, solo el SO o los dos, si el usuario trata de ejecutar algo que no puede, salta una excepción del hardware. Estos niveles de privilegio son el modo sistema privilegiado, y el modo usuario, aunque Intel tiene 4.

Las formas de escalar al modo sistema (privilegio) (de 0 a 3):

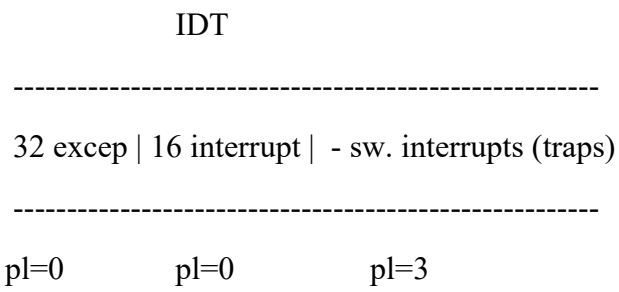
- **Excepción (interrupción síncrona)**: error en el código de usuario, generada por la CPU al final de la ejecución de una instrucción.

- **Interrupción Hardware (asíncrona)**: aviso de otro hw device al cpu sobre error o sobre finalización de tarea. Es asíncrona porque se produce en tiempos arbitrarios.

- **Llamadas al sistema (syscall):** son síncronas, porque las causa el usuario con las instrucciones. Son el mecanismo para hacer request a servicios del SO.

Todas estas formas son gestionadas a través de un vector:

IDT (Interrupt Description Table) es una estructura hardware, un vector en memoria del SO que solo puede tocar este. Es un vector con 256B (o entradas) que contiene el código de la función a lanzar para cada interrupción.



Cuando se produce una de las 3 opciones, esta llega a la CPU, que recibe una petición de cambio de privilegios. Para esto accede a la IDT. Cuando llega una interrupción, es el SO cuando gestiona esta, pero la CPU invoca ese código de gestión. Según el tipo de forma de escalar accede a una posición diferente del IDT.
Cada entrada de la IDT es una interrupción. Cada posición es una @memoria donde está el código o rutina de servicio (handler), que procesará el evento, y su nivel de privilegio, que indica desde que nivel de privilegio se puede invocar el código de esta entrada. Para interrupciones y excepciones el nivel de privilegios será 0, pero el de syscalls será 3 porque provienen del usuario. Así, el usuario no puede lanzar una interrupción o excepción haciéndose pasar por el hardware.

Para validar si la @memoria de la rutina, que contiene la IDT, es correcta, necesitamos la GDT.

La GDT (Global Description Table) es una tabla muy compleja que contiene una descripción de la memoria de la máquina. Rangos del SO, @vacias...

Compara la @ contenida en la IDT con la GPT para saber si existe y es del SO. Si esto es correcto prepara todo para ejecutar el código con máximo nivel de privilegios. Salva el contexto hardware en la pila: SS,ESP, PSW,CS y EIP, y el handler guarda los registros generales y ejecuta la rutina.

CS y SS indican el bloque de memoria del Codigo y de la Stack, y ESP guarda la @ de la pila del usuario, y EIP la @ de la siguiente instrucción del usuario al volver a modo usuario. Por último, PSW es el estado del procesador. El contexto hardware es la información necesaria para volver a modo usuario, y se debe guardar en la pila de sistema, para poder volver atrás desde el modo sistema.

Para hacer esto, la siguiente estructura a la que accederá será TSS (Task State Segment), una estructura en memoria que contiene información sobre el proceso actual en la CPU.

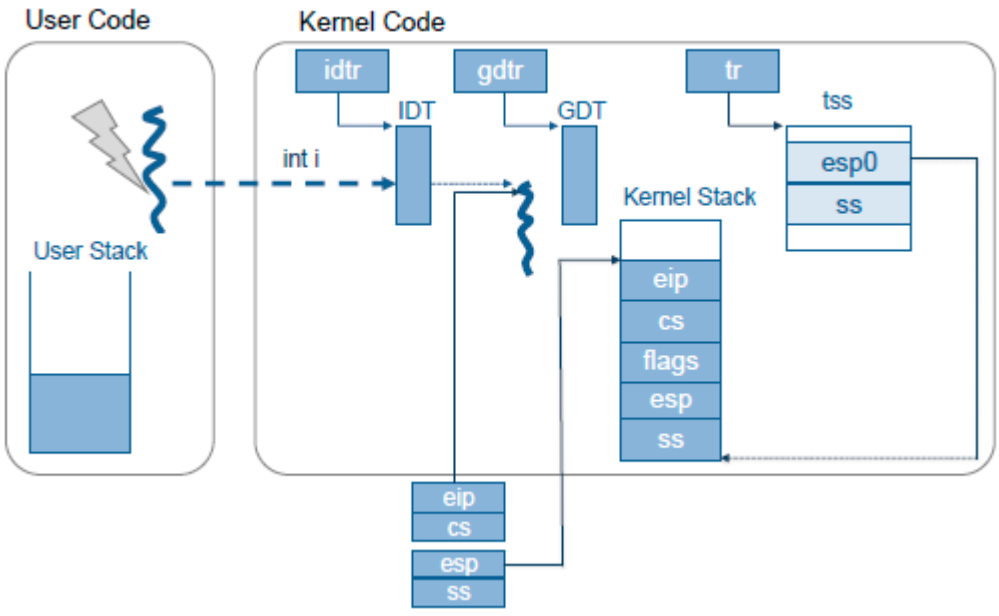
| ESP 0|

Lo importante es el campo ESP 0, que contiene la @ de memoria de lo alto de la pila de sistema para el proceso actual.

Cada proceso del SO tiene dos pilas, una que se usa en modo usuario, y otra que se utiliza en modo sistema, la cual solo puede tocarla el SO. Si tengo 2 procesos, tengo 4 pilas.

Ahora sí, podemos salvar el contexto, y una vez hecho esto, CS+EIP guardará la @ de la primera instrucción de la rutina de servicio, porque es la siguiente instrucción a ejecutar, cambia en PSW el nivel de privilegios al máximo (pl=0), y SS:ESP apuntan a la cima de la pila de sistema.

Ahora ejecuta en máximo nivel la rutina.



BOOT

El SO intenta poner cada estructura en distintas partes de memoria por temas de seguridad. Es el único que puede hacerlo porque requiere PL=0.

Construye IDT en memoria, luego accede a vIDT, inicializa GDT y TSS.

(para acceder a estas tablas usa los registros vIDT, vGST, vTSS, estas @ se guardan una vez inicializadas)

ESCALA DE PRIVILEGIOS. PUNTO DE VISTA DEL SOFTWARE

Handlers:
- **Excepción**: salva el contexto, llama la rutina, restaura contexto, quita el código de error que estaba empilado para pasar el código de error de la excepción, y vuelve a modo user con iret. Las excepciones empilan un parámetro de código de error.

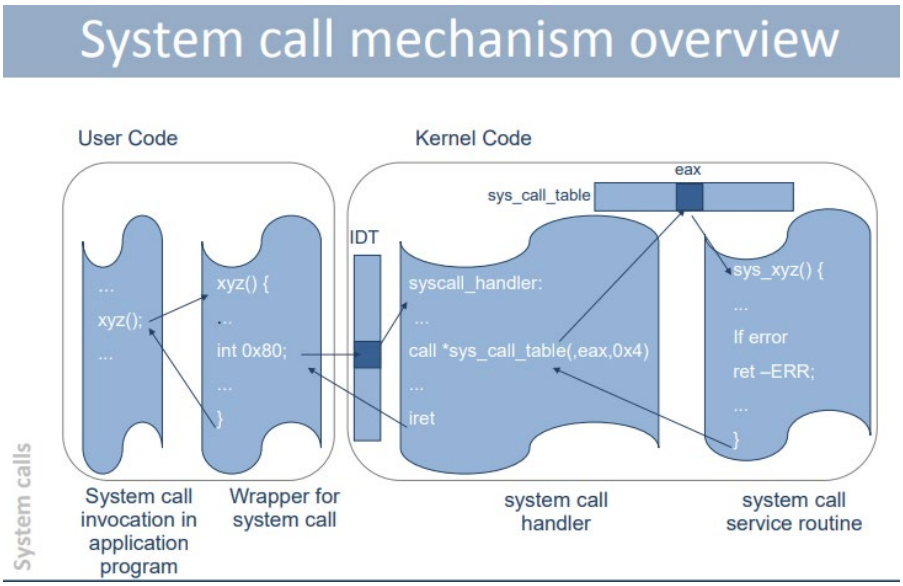
SAVE ALL
CALL routine
ESP <- ESP+4 (elimina el flag de error)
RESTORE ALL
IRET

- **Interrupción**: salva el contexto, llama a la rutina, restaura contexto, envia EOI (End of Interruption) para que el sistema sepa que puede volver a tratar interrupciones, y vuelve a modo usuario con iret.

SAVE ALL
CALL routine
EOI (algunas veces va antes del call)
RESTORE ALL
IRET

- **Sycalls**: salva el contexto hardware y prepara los parámetros, guardándolos en el system stack. Ejecuta la llamada (hace checkeo del # y lo pasa a la syscall_table), actualiza el contexto con el resultado obtenido, restaura el contexto y vuelve con iret al modo user.

Wrappers y Syscall handlers:



Cuando estamos en modo usuario, la función que utilizamos es un **wrapper**, un código ensamblador que "envuelve" todo el procedimiento, ya que se encarga de invocar al handler de la syscall (salvado y paso de parámetros, identificar la syscall que se desea, y generar la llamada con INT o SYSENTER) y devolver el resultado al código del usuario (para los errores usa la variable errno en -1). Estos wrappers se escriben en asembler, y se programan conscientes de la arquitectura y del compilador, y dependen de cómo se haya hecho el handler de syscalls.

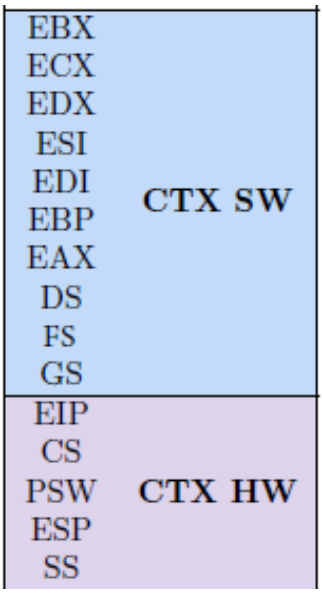
Las llamadas a sistema son instrucciones en assembler que generan una interrupción, ya que tienen un servicio del SO que demandar.
Para todas las llamadas, tenemos un mismo punto de entrada hacia el modo sistema, que puede ser de dos tipos:
- int 0x80 : para llamar al sistema y acceder al syscall_handler.
- sysenter: para llamadas rápidas, que tiene el handler en SYSENTER_EIP_MSR.

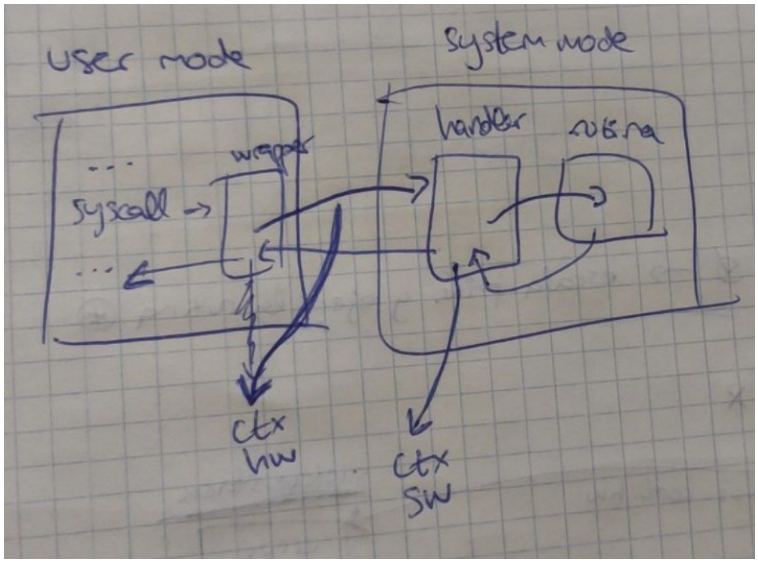
Las syscalls solo tienen un handler, al que se accede con la entrada 0x80 de la IDT.
Antes de entrar al handler, la CPU ha guardado en la system stack el contexto hardware, la mínima información que necesita la CPU para volver a modo usuario.

Wrapper (pila de usuario):
- Guarda registros obligatorios (ebx, esi, edi si se cambian)
- Guarda parámetros de la función
- Asigna #servicio a EAX
- Llama al handler (int / sysenter)
- Checkea errores (si eax<0 error -> errno // si eax>0 todo bien)
- Desempila
- Return

Las syscalls no comparten pilas, por lo que deben ir pasándose parámetros, que se guardan en orden: ebx, ecx, edx, esi, edi, ebp. Y devuelven los resultados por eax.

Una vez estamos en el **handler**, usa EAX para asignar el número de servicio deseado, ya que la IDT solamente tiene una entrada (fila), y dentro de esta divide por servicios.
Si se intenta pasar un # de interrupción o excepción no nos dejará, porque requieren pl=0 y estamos en pl=3, por lo que solo podemos acceder a las # de syscalls, que solo requieren pl=3.
(((También es posible (en otros casos), que la IDT tenga una entrada por servicio (sistemas embeded, #muy limitado), o un conjunto de servicios por entrada (#ilimitado) donde INT indica el conjunto y eax la entrada.)))





La pila de sistema está vacía al entrar al modo usuario, y debe quedar vacía al salir. Al saltar al handler, la CPU guarda el contexto hardware, y dentro del handler se guarda el contexto software, y crea el frame de activación empilando @retorno y ebp.

Dentro del handler, guarda el contexto de usuario (SAVE_ALL), que guarda todo el contexto software, la mínima información necesaria que necesita el SO para volver al modo usuario desde el handler (sistema) al wrapper (usuario), y también se crea el frame de activación. Después, comprueba que el eax sea un número correcto, y se llama a la **syscall_table** donde indica qué rutina corresponde al # contenido en eax, y a partir de aquí se puede ir a ella y ejecutarla.

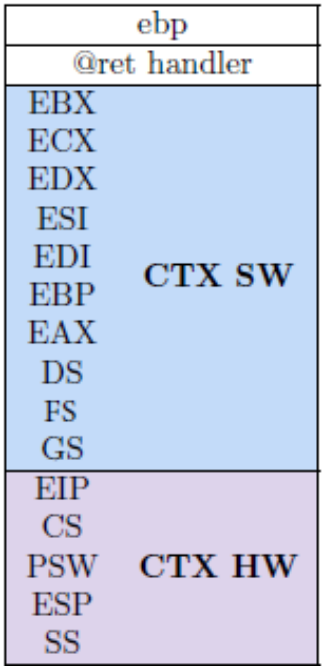
Handler (pila de sistema):

- SAVE ALL
- CHECK EAX (0<eax<max_syscall)
- CALL syscall_table(eax) # (offset, %eax posicion, 4 tamaño entrada)
- 24(ESP) <- EAX
- RESTORE ALL
- IRET

El eax se machaca al restaurar el modo usuario con el valor que tenía antes de saltar al sistema, en este registro teníamos el #servicio cuando hicimos la salvada de contexto de usuario al principio del handler. Al volver de la rutina, eax tiene su valor de retorno. Para preservarlo, debemos acceder al contexto software y cambiar el valor del eax a mano para restaurar el valor de retorno con eax.

Este eax se pisa (y no usa un código diferente) porque en otros casos, como el de interrupciones o excepciones, sí es necesario, y queremos tener la menor cantidad de código posible.

Dentro de la **rutina** se comprueban los parámetros obtenidos por si hay un error, que en dicho caso devolvería un valor de error a través de errno, se devuelve -1 y código de error en errno con valor negativo, si es correcto devolverá un valor positivo. Luego accede al espacio de memoria del proceso si es necesario, y ejecuta el código.



APUNTES DE LAB

En el wrapper: Se pushea %edx y %ecx para guardar los valores originales (parámetros, o lo que haya). También se salva %ebp para acceder con él a la pila de usuario desde modo sistema y obtener a datos de usuario si es necesario. Se llama sysenter, que entrará en modo sistema accediendo a los MSR para obtener las @ que necesita (codigo segmento, system stack y syscall handler). Una vez en el handler: Ejecuta la rutina a la que se llame (lee el #servicio en %eax) y pone en %ecx y %edx los @pila user y @code user, porque sysexit accederá a dichos registros para usar este contenido para volver a modo usuario, encontrándolo más rápidamente que si tuviera que buscar por toda la pila de sistema. Una vez vuelve al modo usuario, en el wrapper, se popea del top de la pila de usuario el %ecx y %edx para restaurar el valor original.