Procedure Calls, Interrupts, and Exceptions

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Procedure Call Types. El procesador soporta dos formas de llamados a procedimientos: (1) instrucciones `CALL` y `RET`. (2) instrucciones `ENTER` y `LEAVE` en conjunto con instrucciones `CALL` y `RET`. Ambos mecanismos de llamado a procedimientos usan la pila (*procedure stack/stack*) para guardar el el estado del procedimiento que invoca el llamado, pasa parámetros al procedimiento que se invoca y almacena variables locales para el procedimiento que está en ejecutándose actualmente.

Stack - Pila. Arreglo continuo de ubicaciones de memoria. Está contenido en un segmento y se identifica por medio del selector de segmento en el registro `SS`. Una pila puede ser hasta 4 Gb de largo. La siguiente ubicación disponible de memoria en la pila se llama tope de la pila. En cualquier momento dado el puntero de la pila brinda la dirección del tope de la pila. Los elementos se colocan en la pila por medio de la instrucción `PUSH` y se sacan por medio de la instrucción `POP`. `PUSH` disminuye el registro `ESP` hacia la siguiente palabra almacenada de la pila y coloca (*push*) un valor ahí. `POP` regresa el valor de la pila e incrementa el registro `ESP`, hacia la siguiente palabra almacenada. Un sistema operativo puede tener tantas pilas como el máximo número de segmentos disponibles en la memoria física. Solamente hay una pila (la actual) disponible a la vez.

Setting Up a Stack. Para establecer una pila el programa o sistema operativo debe: (1) establecer un segmento de pila, (2) cargar el selector de segmento para el segmento de pila dentro del registro `SS` y (3) cargar el puntero de pila dentro del registro `ESP`.

Stack Alignment. El puntero de pila para un segmento de pila debería tener 16 o 32 bits dependiendo del ancho del segmento de pila. El ancho se establece con la bandera `D` en el descriptor del segmento. Cuando el ancho de la pila es de 16 bits, el puntero de la pila tiene incrementos o decrementos en 16 bit. Lo mismo sucede si el ancho de la pila es de 32 bits.

Address-Size Attributes for Stack Adresses. Las instrucciones que utilizan la pila de forma implícita tienen dos atributos de tamaño de dirección cada uno de 16 o 32 bits. Esto porque siempre tiene la dirección implícita del tope de la pila y podrían tener también una dirección explícita de memoria. Puntero `SP` para operaciones de 16 bit. Puntero `ESP` para operaciones de 32 bit.

Procedure Linking Information. El procesador provee dos punteros para el enlazamiendo de procedimientos: (1) apuntador base (*stack-frame base pointer*): contenido en el registro `EBP`. Indentifica un punto de referencia fijo dentro de la pila para el procedimiento invocado. `EBP` apunta automáticamente a una dirección en el segmento de pila actual. (2) Puntero de retorno de instrucción (*return instruction pointer*): antes de la primer instrucción del procedimiento invocado, `CALL` pone la dirección del registro *EIP* dentro de la pila actual. A esta dirección se le llama el puntero de retorno de instrucción y apunta a la instrucción donde la ejecución del procedimiento que

invoca debe retornar luego de que el procedimiento invocado retorna. Dado un retorno, la instrucción RET saca el puntero de retorno de instrucción de la pila de vuelta al registro EIP. Se retoma la ejecución del procedimiento que hizo la invocación. El procesador no requiere que el puntero de retorno de instrucción apunte necesariamente al procedimiento que hizo la invocación de vuelta.

Calling Procedures Using CALL and RET. La instrucción CALL permite transferir el control a procedimientos dentro del segmento de código actual (*near call*) y en segmentos de código diferentes (*far call*). *Near calls* usualmente proveen acceso a procedimeintos locales dentro del programa o tarea que se está ejecutando actualmente. *Far calls* se usan usualmente para tener acceso a procedimientos del sistema operativo o procedimientos un diferentes tareas. RET permite que los retornos de tipo *near* y *far* coincidan con las versiones *near* y *far* de la instrucción CALL. Además RET le permite a un programa incrementar el puntero de la pila en un retorno para liberar parámetros de la pila.

Near CALL *and* RET *Operation.* *Near call*: (1) El procesador poner el valor actual del registro EIP en la pila. (2) Carga el desplazamiento(*offset*) del procedimiento a invocar en el registro EIP. (3) Se inicia la ejecución del procedimiento invocado. *Near return*: (1) El procesador saca el valor que está en el tope de la pila y lo pone en el registro EIP. (2 – Si RET *n*) Incrementa el puntero de la pila con el número de byes especificador con el operando *n* para liberar los parámetros de la pila.

Far CALL *and* RET *Operation.* *Far call*: (1) el procesador pone el valor actual del registro CS en la pila. (2) Pone el valor actual del registro EIP en la pila. (3) Carga el selector de segmento del segmento que contiene el procedimiento invocado en el registro CS. (4) Carga el desplazamiento del procedimiento invocado en el registro EIP. (5) Inicia la ejecución del procedimiento invocado. *Far return*: (1) el procesador saca el valor del tope de la pila y lo pone en el registro EIP. (2) Saca el valor del tope de la pila (el selector de segmento) y lo pone en el registro CS. (3 – Si RET *n*) Incrementa el puntero de la pila con el número de bytes especificado en el operando *n* para liberar los parámetros de la pila. (4) Retoma la ejecución en el procedimiento que hizo la invocación.

Parameter Passing. (1) A través de registros de propósito general: se pueden pasar hasta 6 parámetros al procedimiento copiando los parámetros dentro de estos registros (menos ESP y EBP) antes de la ejecución de CALL. (2) Se pueden poner parámetros en el marco de la pila para el procedimiento que se va a invocar. (3) Se pueden colocar parámetros en una lista de argumentos en uno de los segmentos de memoria. Un puntero a la lista de argumentos se puede pasar al procedimiento que se va a invocar.

Saving Procedure State Information. El procedimiento que invoca debería de almacenar explícitamente los valores in alguno de los registros de propósito general que va a necesitar cuando retome la ejecución luego de una instrucción de retorno. PUSHA almacena los contenidos de los registros de propósito general en la pila. POPA saca todo los valores de los registros almacenados por PUSHA de la pila y los pasa a sus respectivos registros. Si el procedimiento invocado cambia el estado de alguno de los registros explícitamente, debería de restaurarlos a sus valores anteriores antes de ejecutar una instrucción de retorno al procedimiento que lo invocó. Si el procedimiento que invoca quiere mantener el estado del registro EFLAGS se puede utilizar las instrucciones PUSHF/PUSHD y POPF/POPFD.

Calls to Other Privilege Levels. Hay 4 niveles de privilegios numerados de 0 a 3 en donde números más grandes significan menos privilegios. El nivel más alto de privilegios es 0 que contiene los módulos de código más crítico, usualmente el *kernel* del sistema operativo. Módulos de código en segmentos con menores privilegios puede tener acceso a módulos en segmentos con privilegios más elevados por medio de una interfase llamada *gate*. Una llamada a un procedimiento que se encuentra en nivel de protección más elevado: (1) Similar a un *far call*. (2) El selector de segmento provisto en la instrucción CALL referencia a una estructura de datos especial llamada *call gate descriptor* la cual provee acceso a la información de los derechos, al selector de segmento del procedimiento

invocado y un desplazamiento dentro del segmento de código. (3) El procesador cambia a una nueva pila para ejecutar el procedimiento invocado. Cada nivel de privilegio tiene su propia pila.

Cuando se hace una llamada a un nivel de protección de mayor privilegio, el procesador: (1) Realiza una revisión de los derechos de acceso (verificación de privilegios) y (2) almacena temporalmente los contenidos actuales de los registros `SS, ESP, CS` y `EIP`.

Interrupts and Exceptions. Interrupciones: evento asíncrono que típicamente lo genera un dispositivo de entrada/salida. Excepción: evento síncrono que se genera cuando el procesador detecta una o más condiciones predefinidas mientras ejecuta una instrucción. Cuando se detecta alguna de las dos se detiene la ejecución del programa actual y se cambia al procedimiento o tarea que maneja la interrupción o excepción, el cual está disponible a través de una tabla de descripción de interrupciones(IDT). La arquitectura Intel proporciona 16 interrupciones predefinidas y 224 definidas por el usuario las cuales están asociadas a entradas en el IDT con un número llamado vector. Los vectores del 32 al 255 son enmascarables, el usuario los puede modificar. Una llamada a un manejador de interrupciones o excepciones es similar a un llamado a un procedimiento en otro nivel de protección, aquí el vector referencia uno de los dos tipos de *gates*: *interrupt gate*(limpia la bandera `IF` del registro de `EFLAGS`) o *trap gate*. `IRET` retorna el control desde una interrupción o excepción y restaura el contenido del registor `EFLAGS` para el procedimiento interrumpido. Las rutinas de manejo de interrupciones y excepciones se pueden realizar también en una tarea por separado, esto cuando causan que una tarea cambie a un manejador de tareas. Al manejador de tareas se le de su propio espacio de direcciones y puede ejecutar a un nivel más elevado de protección. Cuando se opera en modo *real-address* el procesador responde a una interrupción o excepción con un *far call* implícito hacia un manejador. `INT n`: usa un vector de interrupciones como argumento lo que le permite a un programa invocar cualquier manejador de interrupciones. `INTO`: llamado al manejador de excepción por desbordamiento. `INT 3`: llamado al manejador de excepción por punto de interrupción (*breakpoint*). `BOUND`: llamada al manejador de excepciones de rangos rebasados.

Procedure Calls for Block-Structured Languages. Métodos alternativos para llamado a procedimientos: `ENTER` y `LEAVE`. Estas instrucciones automáticamente crean y liberan marcos de pila de procedimientos invocados. Los marcos de pila tienen espacios predefinidos para variables locales y los punteros necesarios para permitir retornos coherentes desde los procedimientos invocados. También permiten la implementación de reglas en el ámbito del alcance(*scope rules*) que tienen los procedimientos para tener acceso a variables locales. Beneficios: (1) Proveen soporte de lenguaje máquina para la implementación de lenguajes con estructuras de bloque como C y Pascal. (2) Simplifican la entrada y salida de procedimientos en código generado por compilador.

`ENTER`. Crea un marco de pila compatile con las reglas en el ámbito del alcance típicamente utilizadas en lenguajes con estructuras de bloque. En estos lenguajes el ámbito de un procedimiento es un conjunto de variables los cuales puede accesar. Tiene dos operandos: el primero especifíca el número de bytes a reservar en la pila para almacenamiento dinámico del procedimiento que se va a llamar. El segundo parámetro es el nivel de anidamiento léxico ($0 - 31$) del procedimiento el cual es el nivel de profundidad de un procedimiento en una jerarquía de llamadas de procedimientos.

LEAVE. Revierte la acción de la instrucción `ENTER` previa. Copia los contenidos del registro `EBP` dentro de `ESP` y libera todo el espacio de pila ocupado por el procedimiento. Luego restaura el viejo valor del registro `EBP` de la pila. Esto simultáneamente restaura el registro `ESP` a to valor original. Una subsecuente instrucción `RET` puede remover cualquier argumento y las direcciones de retorno ingresadas en la pila por el programa invocador para ser usado por el procedimiento.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

El cómo invocar procedimientos o subrutinas en una arquitectura Intel.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

Porque da una descripción detallada (tipo "detrás de cámaras") de la interacción que se da entre componentes tales como el CPU, memoria y dispositivos de entrada y salida a la hora de invocar estos procedimientos. La CPU solamente puede hacer una cosa a la vez y son por medio las interrupciones que se puede "orquestar" y atender los diferentes dispositivos y eventos que pasan en la computadora.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

La primera técnica que se empleó para esto fue el *polling*, que consistía en que el propio procesador se encargara de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tenía pendiente alguna comunicación para él. Este método presentaba el inconveniente de ser muy ineficiente, ya que el procesador consumía constantemente tiempo y recursos en realizar estas instrucciones de sondeo [Wikipedia].

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

Los principales mecanismos para llamar un procedimiento es por medio de las instrucciones `CALL` y `RET`, y `ENTER` y `LEAVE`. Cuando estas instrucciones son invocadas se crea en memoria un estructura de tipo pila por medio de la cual se podrá llevar registro de la preparación y ejecución de un procedimiento. Esta pila permite saber en dónde empieza un procedimiento, cuáles son sus variables locales, sus parámetros, su instrucción actual y hacia adónde retornar el control luego de haber finalizado.

A pesar de contar con dos mecanismos para realizar llamados a procedimientos y de introducir otros conceptos tales como interrupciones, excepciones y llamadas a procedimientos en niveles con privilegios elevados, sigue siendo en el fondo esta estructura de pila la encargada de soportar esta ejecución. Cada concepto nuevo que se presenta modifica esta pila agregando más instrucciones y datos dependiendo de la tarea que esten llevando a cabo.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Ha sido exitosa porque los sistemas orientados a interrupciones son más eficientes (a pesar de que pueden ser mucho más complejos). Aún estan vigentes.

REFERENCIAS

[Wikipedia] Wikipedia. *Interrupción*. Disponible en https://es.wikipedia.org/wiki/Interrupci%C3%B3n