Interrupts, Traps, 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

El concepto de interrupción se ha expandido a través de los años. Diferentes fabricantes usan términos como *exceptions, faults, aborts* y *interrupts* para describir este comportamiento. No hay un claro concenso ni un significado exacto. Una trampa (*trap*) denota una transferencia de control iniciada y esperada de un programador hacia una rutina de manejo (*handler routine*) en particular. Algunos textos se refieren a ellos como interrupciones de software. Las trampas son incondicionales, esto es, que cuando se ejecuta una instrucción `int`, el control siempre se transfiere al procedimiento asociado con la trampa. Dado que las trampas se ejecutan de modo explicíto, es fácil determinar exactamente cuáles instrucciones van a invocar a una trampa en un programa. Una excepción es una trampa que se genera automáticamente(no se solicita). No hay una instrucción específica asociada para las excepciones. Una excepción ocurre en respuesta a algún comportamiento degenerativo en la ejecución normal de un programa. Cuando se presenta una excepción la CPU suspende la ejecución de la instrucción actual y transfiere el control a una rutina de manejo de excepción, la cuál decide qué hacer. Interrupciones de *hardware (hardware interrups)*, son interrupciones del control del programa a partir de eventos de *hardware* externo que le informa al CPU que necesita atención. El CPU interrumpe la ejecución del programa actual, atiende el dispositivo y luego retorna el control de vuelta al programa. Una rutina de servicio de interrupción (*interrupt service routine – ISR*) es un procedimiento escrito específicamente para manejar la trampa, excepción o interrupción.

80x86 Interrupt Structure and Interrupt Services Routines(ISRs). El chip de la 80x86 permite hasta 256 interrupciones vectorizadas: 256 fuentes diferentes para una interrupción en donde la 80x86 llamará directamente a la rutina sin procesamientos intermedios. Las interrupciones no-vectorizadas transfieren el control a un único ISR independientemente de la fuente de la interrupción. La 80x86 provee una tabla de vectores de interrupción de 256 entradas iniciando en la dirección 0:0 en memoria. Esta es una tabla de 1K que contiene 256 entradas de 4 bytes. Cada entrada en esta tabla contiene una dirección que apunta al ISR en memoria (0:0, 0:4, 0:8, etc). Las excepciones y las interrupciones de *hardware* tiene una reestricción: tienen que preservar el estado del CPU, en particular tienen que preservar todos los registros que modifica.

Traps. Una trampa es una interrupción invocada por *software* (por medio de `int` *n*). Su propósito principal es el de proveer una subrutina fija la cual varios programas pueden llamar sin tener la necesidad de saber la dirección real en donde se ejecuta (MS-DOS se puede ejecutar con `int 21h`). Dicha instrucción transfiere el control al ISR cuya entrada aparezca en la posición enésima en la tabla de interrupciones.

Exceptions. Las excepciones se "lanzan" cuando una condición anormal ocurre durante la ejecución. Aunque los manejadores de excepciones son definidos por el usuario, la 80x86 define las excepciones que pueden ocurrir, también asigna un número fijo de interrupción a cada excepción.

1. *Divide Error Exception* (`INT 0`): cuando se intenta dividir un valor por cero o cuando el cociente no cabe en el registro destino cuando se usan las instrucciones `div o idiv`.
2. *Single Step (Trace) Exception* (`INT 1`): ocurre luego de cada instrucción si el *bit* de *trace* en el registro de `FLAGS` es igual a uno. Varios programas lo utilizan esta bandera para poder seguir la ejecución de un programa.
3. *Breakpoint Exception* (`INT 3`): es una trampa no una excepción. Ocurre cuando el CPU ejecuta la instrucción `int 3`.
4. *Overflow Exception* (`INT 4/INTO`): técnicamente una trampa. Solamente se lanza esta excepción cuando se ejecuta la instrucción `into` y la bandera de *overflow* está habilitada.
5. *Bounds Exception* (`INT 5/BOUND`): si un registro específico está fuera de sus límites especificados, la instrucción bound actúa como una instrucción `int 5`.
6. *Invalid Opcode Exception* (`INT 6`): se lanza si se intenta ejecutar un código de operación (*opcode*) el cual no corresponde a una instrucción 80x86 válida.
7. *Coprocessor Not Available* (`INT 7`): se lanza si se intenta ejecutar una instrucción de un coprocesador no instalado (por ejemplo ejecutar una instrucción de punto flotante sin tener el coprocesador instalado).

Hardware Interrupts. Las interrupciones vienen de varias fuentes: teclado, puertos seriales y paralelos, rejoles, unidades de disco y otros dispositivos periféricos. Estos dispositivos se conectan a un controlador de interrupciones programable (PIC), el Intel 8259A que prioriza las interrupciones y las interfaces con el CPU. La 80x86 provee hasta 15 interrupciones vectorizadas usando un par de PICs (maestro-esclavo). El PIC también permite enmascarar dispositivos que pueden interrumpir el sistema.

1. *The Timer Interrupt* (`INT 8`): La tarjeta madre contiene un chip temporizador con tres canales, uno de los cuales genera interrupciones cada 55 msec.
2. *The Keyboard Interrupt* (`INT 9`): genera dos interrupciones en cada pulsación de tecla: una cuando se presiona y otra cuando se libera.
3. *The Serial Port Interrupts* (`INT 0Bh` *and* `INT 0Ch`): el chip de comunicaciones seriales genera una interrupción si: un caracter arriba a la línea serial, si finaliza la comunicación, si ocurre un error o si hay un cambio en el estado.
4. *The Parallel Port Interrupts* (`INT 0Dh` *and* `INT 0Fh`): un enigma.
5. *The Diskette and Hard Drive Interrupts* (`INT 0Eh` *and* `INT 76h`: se genera cuando una unidad de *floppy* o de disco duro completa una tarea.
6. *The Real-Time Clock Interrupt* (`INT 70h`): temporizador capaz de generar una interrupción cada 1 msec.
7. *The FPU Interrupt* (*INT 75h)*: cuando ocurre una excepción de punto flotante.
8. *Nonmaskable Interrupts* (`INT 2`)
9. *Other Interrupts*

Chaining Interrupt Service Routines. Existen dos variedades de ISRs:

1. Los que necesitan acceso exclusivo a un vector de interrupciones (ISRs de manejo de errores: división por cero, *overflow*)
2. Los que tienen que compartir un vector de interrupciones con varios otros ISRs (Ej: ISRs para relojes en tiempo real, teclado. Se pueden encontrar varios de estos ISRs en memoria compartiendo cada una de estas interrupciones)

Los ISRs escritos por los programadores pueden necesitar coexistir con otros ISRs en memoria. No se puede simplemente reemplazar un vector de interrupciones con la dirección de un ISR y hacer que este ISR tome el control a partir de ese momento. Con frecuencia, se necesitará crear una cadena de interrupciones (*interrupt chain*) e invocar ISRs previos en la cadena una vez que se haya terminado con la programación del ISR.

Reentrancy Problems. ¿Qué pasaría si se habilitan interrupciones mientras se está en un ISR y una segunda interrupción llega también desde del mismo dispositivo? Esto podría interrumpir al ISR y entonces re-entrar (*reenter*) en el ISR desde el principio. Una aplicación que puede manejar apropiadamente esta situación se dice que es "re-entrante" (*reentrant*). Los segmentos de código que no operan apropiadamente cuando re-entran son "no re-entrantes". Un programa no debería re-entrar mientras se ejecutan porciones de código que se consideren críticas, esto es, regiones en donde se están modificando registros que podrían ser afectados si se interrumpiera el ISR y se fuera al inicio otra vez.

The Efficiency of an Interrupt Driven System.

Interrupt Driven I/O vs. Polling. El propósito de un sistema orientado a interrupciones es el de permitir al CPU a continuar procesando instrucciones mientras alguna actividad de entrada/salida ocurre. Esto es lo opuesto a un sistema de consulta constante (*polling*) en donde en CPU está constantemente probando un dispositivo de entrada/salida para ver si una operación de entrada/salida está completa. En un sistema orientado a interrupciones, el CPU hace su trabajo y el dispositivo de entrada/salida interrumpe cuando necesita ser atentido. Esto es generalmente más eficiente.

Interrupt Service Time. Dos factores controlan el impacto de un ISR en una computadora:

1. Frecuencia de interrupciones: cuántas veces ocurre una interrupción por segundo. Varia dependiendo de la fuente de la interrupción: chip temporizador 18 veces por segundo, puerto serial 100 veces por segundo, teclado 20 veces por segundo.
2. Tiempo de servicio de una interrupción: cuánto tiempo le toma a un ISR atender una interrupción. Depende del número de instrucciones que un ISR debe ejecutar. También depende del CPU y de la frecuencia del reloj. La cantidad de tiempo que le toma a una rutina de ISR manejar una interrupción multiplicado por la frecuencia de la interrupción, determina el impacto de la interrupción en el sistema.

Interrupt Latency. Es el tiempo que hay entre el punto en donde el dispotivo envía un mensaje de que necesita ser atendido y el punto donde el ISR provee el servicio necesario. El PIC 8259 necesita enviar una señal al CPU, el CPU necesita interrumpir el programa actual, poner los *flags* y direcciones de retorno en la pila, obtener la dirección del ISR y transferir el control al ISR. El ISR podría necesitar poner varios registros, configurar variables, revisar el estado del dispositivo para determinar la fuente de la interrupción, entre otras cosas. Además, pueden haber otros ISRs encadenados dentro del vector de interrupciones que se van a ejecutar antes de transferir el control a un ISR en particular. En teoría un ISR que tome menos de 10 segundos se considera bueno.

Prioritized Interrupts. ¿Cuál interrupción debe de ser atendida primero por el CPU? ¿Cuál es más importante? El PIC 8259 no lleva un registro de cuál interrupción sucedió primero. El PIC 8259 provee varios esquemas de priorización pero es el BIOS el que se encarga de inicializar el chip con prioridades fijas: el dispositivo en `IRQ 0` (el temporizador) tiene la más alta prioridad y el dispositivo en `IRQ 7` la más baja. La interrupción de no-enmascaramiento tiene prioridad más alta de todas en el PIC 8259.

Debugging ISRs. Los ISRs son dificiles de *debuggear*. ISRs errantes pueden modificar valores que el programa principal u otro programa usa. También, la mayoría de los *debuggers* tienen problemas a la hora de intentar fijar un *breakpoint* dentro de un ISR porque ellos no son re-entrantes. De tal forma que cuando se fija un *breakpoint* dentro de un ISR que tiene interrupciones de BIOS o DOS y el *debugger* llama al BIOS o al DOS, el sistema podrían caerse por problema de re-entrada.

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

El cómo manejar escenarios "excepcionales" que se dan durante la ejecución de un programa en la 80x86. Estos escenarios se pueden dar a raíz de errores de la programación, pueden ser inducidos por el programador o bien se

dan cuando un dispositivo de entrada/salida necesita ser atendido y el CPU necesita interrumpir su labor actual con el fin de brindarle servicio.

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

Porque mientras una computadora se encuentra en funcionamiento, estos escenarios excepcionales se dan más bien con muchísima frecuencia, los programas fallan y se necesita contar con mecanismos para manejar estos fallos y no causar efectos secundarios en la actividad del CPU, los registros, en la memoria o en otros programas. Al mismo tiempo, el computador necesita interactuar con dispositivos de entrada/salida y cuando esto se hace, se necesita proveer atención a los mismo porque de lo contrario toda la actividad que lleven a cabo sería ignorada.

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

Diferentes fabricantes utilizan diferentes instrucciones e implementaciones para el manejo de interrupciones. Se menciona que en la 80x86 existen tres tipos pero por ejemplo en la serie Motorola 68000, la competidora de la época de la 80x86, existen otros tipos y otras implementaciones. En [ANTIC] se puede encontrar una reseña sobre excepciones e interrupciones en la 68000.

Un enfoque alternativo al uso de interrupciones es por medio *polling* en donde CPU esta constantemente consultando si una tarea se ha completado en un dispositivo de entrada/salida. Si los escenarios excepcionales fueran implementados por medio de *polling* se corre con el riesgo de que el CPU no tenga ciclos para realizar otro tipo de procesamiento. Se da un ejemplo en donde si se llegara a implementar la comunicación serial por medio de *polling*, entonces el CPU deberá de pasar todo su tiempo verificando si un caracter ha arribado.

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

El principal mecanismo para el manejo de excepciones trampas e interrupciones en la 80x86 es por medio de:

- 256 interrupciones vectorizadas, las cuales llaman directamente a una rutina de interrupción.
- Interrupciones no vectorizadas, entregan el control a un servicio de rutinas para interrupciones (ISR). La 80x86 provee una tabla de vectores de interrupción de 256 entradas iniciando en la dirección 0:0 en memoria. Esta es una tabla de 1K que contiene 256 entradas de 4 bytes. Cada entrada en esta tabla contiene una dirección que apunta al ISR en memoria. Cuando ocurre una interrupción:
  1. El CPU pone el registro de `FLAGS` en la pila
  2. El CPU pone un *far return address (segment/offset)* en la pila [1]
  3. El CPU determina la causa de la interrupción
  4. El CPU transfiere el control a la rutina especificada por la tabla de vectores de interrupción

Luego de los pasos anteriores el ISR toma el control y luego lo retorna por medio la instrucción `iret`.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

A pesar de la complejidad que las interrupciones introducen en un sistema (por ejemplo en *debugging*), los sistemas orientados a interrupciones son usualmente superiores en eficiencia aunque esto no siempre es el caso. Para muchos otros sistemas, métodos alternativos proveen mejor rendimiento.

REFERENCIAS

[ANTIC] Actic Magazine, Classic Computer Magazine Archive. *68000 EXCEPTIONS & INTERRUPTS: Part II: Into the ST*. Vol 5, No 2. June 1986. Disponible en http://www.atarimagazines.com/v5n2/ExceptionsInterrupts.html

[1] *Far return: un retorno a una llamada de un procedimiento localizado en un segmento diferente que el segmento del código actual.*