Interrups, 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 *interrups* 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 cuales 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 algun comportamiente 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 (*interrup service routine – ISR*) es un procedimiento escrito específicamente para manejar la trampa, excepción o interrupción.

80x86 Interrup Structure and Interrup 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 interrupciones 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 "son lanzadas" 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 seleccionada.
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 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 Interrup* (`INT 8`): La tarjeta madre contiene un chip temporizador con tres canales, uno de los caules genera interrupciones cada 55 msec.
2. *The Keyboard Interrup* (`INT 9`): genera dos interrupciones en cada pulsación de tecla: una cuando se presiona y otra cuando se libera.
3. *The Serial Port Interrups* (`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 Interrups* (`INT 0Eh` *and* `INT 76h`: se genera cuando una unidad de *floppy* o de disco duro completa una tarea.
6. *The Real-Time Clock Interrup* (`INT 70h`): temporizador capaz de generar una interrupción cada 1 msec.
7. *The FPU Interrup* (*INT 75h)*: cuando ocurre una excepción de punto flotante.
8. *Nonmaskable Interrups* (`INT 2`)
9. *Other Interrups*

Chaining Interrup Service Routines.

Reentrancy Problems.

The Efficiency of an Interrup Driven System.

Interrup Driven I/O vs. Polling.

Interrup Service Time.

Interrup Latency.

Prioritized Interrups.

Debugging ISRs.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?
2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?
3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?
4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?
5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?