Interrupts

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

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

Las interrupciones causan al microprocesador de un sistema embebido a suspender lo que sea que este haciendo y a ejecutar un código diferente en su lugar, código que va a responder al evento que causó la interrupción.

Microprocessor Architecture. Lenguaje ensamblador: un lenguable llamado ensamblador traduce el lenguaje ensamblador en números binarios antes de que el microprocesador pueda ejecutarlos. Cada instrucción de lenguaje ensamblador se convierte en una sola instrucción para el microprocesador. Cuando el compilador traduce C, la mayoría de los comandos se convierten en múltiples instrucciones para el que el microprocesador las ejecute. Cada familia de de microprocesadores tiene diferente lenguaje ensamblador porque cada familia entiene un conjunto de instrucciones diferente.

Un microprocesador típico tiene:

- un conjunto de registros que pueden tener un valor con el que el procesador está trabajando. Antes de hacer operaciones en datos, el procesador tiene que mover datos hacia los registros.
- un *program counter* que lleva registro de la dirección de la próxima instrucción que va a ser ejecutada.
- un *stack pointer* que almacena la dirección en memoria del tope de la pila de propósito general.

Interrupt Basics. Las interrupciones se inicia con una señal del *hardware*, por ejemplo cuando el chip del puerto serial recibe un caracter, necesita el microprocado para leer ese caracter y almacenarlo en memoria. El microprocesador tiene un pin de entrada llamado *interrupt request (IRQ)* el cual le deja saber al microprocesador que algún otro chip necesita ayuda. Cuando se detecta una señal de uno de los IRQ se detiene la ejecución de las instrucciones que se estaban ejecutando, se guarda en la pila la direcciòn de la instrucción que sigue y se salta a una rutina de interrupción que algunas veces se le llama *interrupt handler* o *interrupt service routine (ISR)*. La última instrucción que ejecuta una rutina de interrupción es un `RETURN`, la cual hace que el procesador retome el trabajo previo. **Guardar el contexto**: se ingresan todos los registros en la pila al inicio de la rutina de interrupción. **Restaurar el contexto**: sacar todos los registros al final. Los chips de entrada y salida y microprocesador pueden ignorar señales de interrupción. Interrupción no enmascarable: una interrupción que no puede ser deshabilitada. Algunos microprocesadores asignan una prioridad a cada señal de petición de interrupción para especificar la prioridad de la interrupción con más baja prioridad que esta dispuesta a manejar en un momento dado. Se pueden deshabilitar todas las interrupciones seleccionando la prioridad más alta como la aceptable o habilitar todas seleccionando la prioridad más baja como la aceptable.

The Shared-Data Problem. Este problema se da cuando hay datos que se comparten entre la rutina de la interrupción con otro código que lleva a cabo una tarea en ese momento. Para resolver el problema se puede: deshabilitar las interrupciones cuando el código que está realizando una tarea use datos compartidos. Una parte **atómica** de un programa que no se puede interrumpir, para no crear conflictos con datos compartidos. A un

conjunto de instrucciones que deben de ser atómicas para que un sistema trabaje apropiadamente se le llama **sección crítica**. Dependiendo del microprocesador, si los registros son pequeños, la cantidad de instrucciones `MOVE` que se ejecutan podrían hacer que un código no sea atómico (por ejemplo si tiene 2 o más). Es por esto que para deshacerse de este comportamiento siempre se recomienda deshabilitar las interrupciones cuando se lee alguna variable compartida. En C, con la declaración de la palabra reservada `volatile` el compilador le hace saber al microprocesador que tiene que leer el valor de una variable desde la memoria cada vez que sea referenciada. La palabra reservada `volatile` permite advertir al compilar que ciertos variables pueden cambiar debido a una rutina de interrupción u otras cosas que el compilador no sabe.

Interrupt Latency. ¿Qué tan rápido un sistema puede responder a cada interrupción? Factores:

1. El periódo más largo durante el cual esa interrupción está deshabilitada.
2. El periódo que toma ejecutar cualquier rutina de interrupción para interrupciones que tienen una prioridad más alta que la que está en cuestión.
3. Cuánto le toma al microprocesador detener lo que está haciendo, registrar lo pendiente e iniciar la ejecución de instrucciones dentro de una rutina de interrupción.
4. Cuánto le toma a la rutina de interrupción guardar el contexto y luego hacer suficiente trabajo que sea catalogado como una respuesta.

El término **interrupt latency** se refiere a la cantidad de tiempo que le toma a un sitam responder a una interrupción. Es buena idea escribir rutinas de interrupción que sean cortas. El tiempo de procesamiento usado por una rutina de interrupción retarda la respuesta en todas las otras interrupciones con el mismo o menos nivel de prioridad. El otro factor que contribuye a la latencia es la práctica de deshabilitar interrupciones. Esto en ocasiones es necesario con el fin de resolver problemas de datos compartidos pero entre más corto sea el periódo durante el cual las interrupciones están deshabilitadas, mejor será la respuesta.

Aunque existen técnicas para evitar tener que deshabilitar interrupciones, están se consideran frágiles y deberían de ser usedas si son absolutamente necesarias.

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

Cómo lidiar con interrupciones dentro del contexto del problema de la respuesta. El problema de la respuesta (*response problem*) es el de asegurarse que un sistema embebido reaccione rápidamente a eventos externos, incluso si está en medio de alguna otra cosa. Por ejemplo, incluso el sistema de monitoreo de un tanque de gasolina está ocupado calculando cuánta gasolina hay en el tanque número 6, el sistema debe aún responder prontamente si un usuario presiona un botón para saber cuánta gasolina hay en el tanque número 2.

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

Porque refleja lo intrincado y complicado de diseñar un sistema que tome en cuenta tantas cosas que pueden pasar en un solo momento y aunque en la práctica mucho de esto se vuelve en algo imperceptible para nosotros como usuarios, la verdad es que en el interior la computadora están pasando muchas cosas casi al mismo momento y se necesita tener mecanismos para controlar toda esta interacción y proveer servicio.

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

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

El enfoque de los autores para atacar el problema de la respuesta es por medio del uso de interrupciones. Por medio de su uso se puede resolver este problema pero no sin antes pagar el precio de una programación dificultosa o el de añadir aún nuevos problemas durante la implementación de la solución.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?