Taller de Sistemas Operativos

Interrupciones y Excepciones

Agenda

- Conceptos generales
- Interrupciones
- Excepciones
- Líneas IRQs y PIC
- Tabla de descriptores de interrupciones
- Manejadores de interrupciones
- Manejador de excepciones
- Bottom Halves
  - Softirqs
  - Tasklets
  - Work Queues

Conceptos generales

- Las interrupciones pueden ser:
  - Sincrónicas: son las generadas por la CPU al ejecutar instrucciones
  - Asincrónicas: son las generadas por otros dispositivos y no están alineadas al clock del sistema
- Intel designa como excepciones e interrupciones a las sincrónicas o asincrónicas respectivamente

Interrupciones

- Cuando una interrupción es detectada por el procesador, se debe parar la ejecución de lo que se estaba haciendo y ejecutar el código correspondiente de atención de interrupciones
- Esto genera un kernel control path, que ejecuta en la estructura del proceso que estaba ejecutando
- Los registros de ejecución son salvados en el Kernel Mode Stack del proceso interrumpido
- El manejador de la interrupción tiene su propio stack de una página (4 KB)

Interrupciones

- La atención de interrupciones debe satisfacer:
  - Las interrupciones deben ser atendidas de forma rápida
    - Toda tarea que no sea necesaria se debe diferir para otro momento
  - Debe ser posible ejecutarlas en forma anidada
  - Deben ejecutar, todo lo que sea posible, con las interrupciones habilitadas

Interrupciones

- Intel proporciona una instrucción que deshabilita las interrupciones “enmascarables”
- Hay algunas que no puede deshabilitar: No “enmascarables”
  - Por lo general son errores de hardware
- La flag IF del registro EFLAGS determina el estado de las interrupciones
- Se garantiza que no se pierden interrupciones
- Estas son generadas luego que se habiliten nuevamente

Excepciones

- Las excepciones son generadas por:
  - Errores en la programación (p.ej.: división por cero)
  - Condiciones anómalas (p.ej.: fallo de página)
- Si son de errores de programación se le envía una señal al proceso que las generó
- El proceso debe capturar estas señales, sino se ejecuta el handler por defecto que finalizará la ejecución
- Si se da una condición anómala, el núcleo deberá reestablecer la condición para continuar la ejecución

Excepciones

- Las excepciones clasifican en:
  - Detectadas por el procesador
    - Fallos
      - Pueden ser corregidos y retoman la ejecución. Se retoma la instrucción que generó el fallo
    - Traps
      - Es utilizada, en mayor media, para debugging
    - Aborts
      - Un error grave ocurrió como un fallo del hardware
  - Programadas
    - Son generadas por los programas al ejecutar la instrucción int o int3

Excepciones

#	Exception	Exception handler	Signal
0	Divide error	divide_error( )	SIGFPE
1	Debug	debug( )	SIGTRAP
2	NMI	nmi( )	None
3	Breakpoint	int3( )	SIGTRAP
4	Overflow	overflow( )	SIGSEGV
5	Bounds check	bounds( )	SIGSEGV
6	Invalid opcode	invalid_op( )	SIGILL
7	Device not available	device_not_available( )	None
8	Double fault	doublefault_fn( )	None
9	Coprocessor segment overrun	coprocessor_segment_overrun( )	SIGFPE
10	Invalid TSS	invalid_TSS( )	SIGSEGV
11	Segment not present	segment_not_present( )	SIGBUS
12	Stack segment fault	stack_segment( )	SIGBUS
13	General protection	general_protection( )	SIGSEGV
14	Page Fault	page_fault( )	SIGSEGV
15	Intel-reserved	None	None
16	Floating-point error	coprocessor_error( )	SIGFPE
17	Alignment check	alignment_check( )	SIGBUS
18	Machine check	machine_check( )	None
19	SIMD floating point	simd_coprocessor_error( )	SIGFPE

Líneas IRQs y PIC

- El sistema cuenta con una unidad de hardware denominada PIC (Programmable Interrupt Controller)
- Los controladores de dispositivos tienen una línea (IRQ) de conexión contra la controladora de interrupciones
- La tarea de la controladora PIC se resume en:
  - Monitorear las líneas de las controladoras
  - Si una está encendida:
    - Poner disponible en los puertos (I/O ports) de la controladora datos para el procesador.
    - Enviar una señal al procesador.
    - Esperar a que el procesador acuse la señal generada a través de la escritura en un puerto.
    - Volver al paso de monitoreo

Líneas IRQs y PIC

- El PIC es un chip programable
  - Se permiten deshabilitar/habilitar las líneas IRQ
  - Si están deshabilitadas, las interrupciones no se pierden
  - Posteriormente, cuando el procesador habilita la línea, se le reportan las interrupciones
  - El chip soporta 8 líneas y se conectan en cascada (15 líneas)

Tabla de descriptores de Interrupciones

- Se define una tabla global que contiene la asociación de las interrupciones o excepciones con el handler correspondiente (IDT – Interrupt Descriptor Table)
- Existe un registro (idtr) que tiene la dirección base de la tabla y el largo
- Es posible definir hasta 256 entradas
- Se tiene 3 tipos de descriptores:
  - Task gate
  - Interrupt gate (interrupciones)
  - Trap gate (excepciones)
- La tabla es creada al iniciarse el sistema y luego los device drivers de los dispositivos la actualizan con los handlers correspondientes.

Manejadores de interrupciones

- El device driver de los dispositivos es el encargado de registrar la rutina de interrupción
- Esta tarea involucra invocar la rutina:

```
int request_irq (
   unsigned int irq,
   irq_handler_t handler,
   unsigned long irqflags,
   // IRQF_DISABLED, IRQF_SAMPLE_RANDOM,
   // IRQF_TIMER, IRQF_SHARED
   const char * devname,
   void * dev_id)
```

Manejadores de interrupciones

- El handler tiene el cabezal:

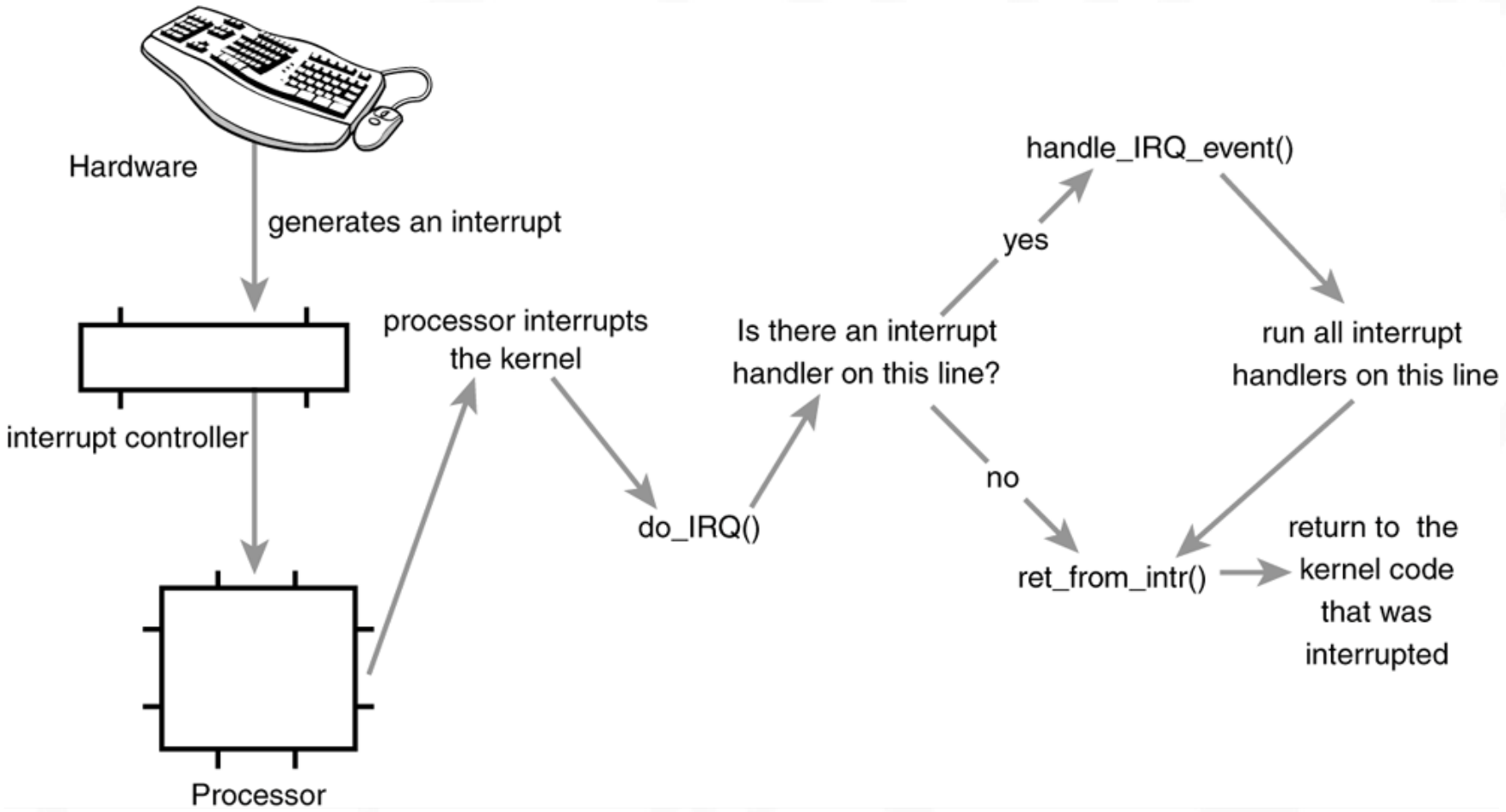
```
irqreturn_t intr_handler(int irq,
                         void * dev_id)
```

  - irq: Número de línea de interrupción
  - dev_id: El que se le paso cuando se declaró el handler

Manejadores de interrupciones

- El código debe ser bastante limitado
- Solo debería tomar en cuenta copiar la información a memoria y liberar a la PIC.
- La rutina no tiene necesidad de ser reentrante ya que la interrupción queda deshabilitada en todos los procesadores
- Es posible que en una línea se comparta por varios dispositivos. En estos casos el sistema invocará todas las rutinas correspondientes y solo deberá ejecutarse la del dispositivo adecuado
- La rutina tiene un stack de una página y, como se pueden anidar, se debe hacer un uso lo más limitado posible

Camino de una interrupción



Ejecución del *handler*

```
irqreturn_t handle_IRQ_event(unsigned int irq, struct
                                              irqaction *action) {
    int status = 1;
    int retval = 0;
    if (!(action->flags & IRQF_DISABLED))
       local_irq_enable(); // deshabilitadas por la CPU
    do {   status |= action->flags;
           retval |= action->handler(irq, action->dev_id);
           action = action->next;
    } while (action);
    if (status & IRQF_SAMPLE_RANDOM)
       add_interrupt_randomness(irq);
    local_irq_disable();
    return retval;
}
```

Habilitación de interrupciones

- En el handler es posible deshabilitar las interrupciones en forma momentánea:

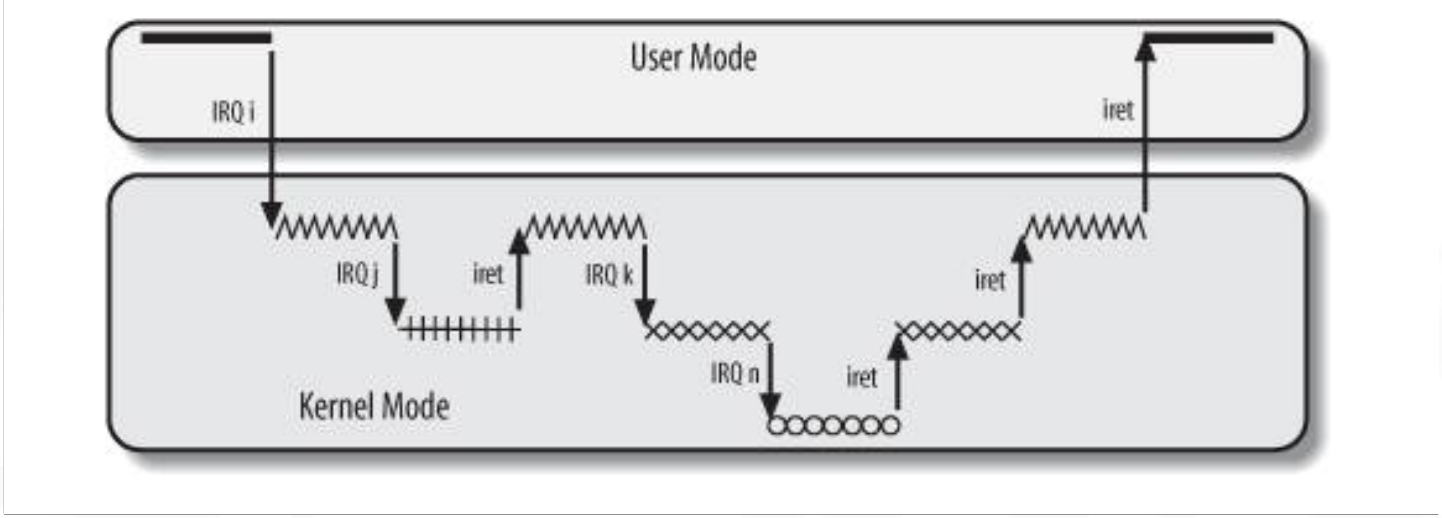
```
local_irq_disable() /* cli */
…
local_irq_enable()  /* sti */
```

- También es posible deshabilitar alguna interrupción específica:

```
disable_irq(unsigned int irq)
enable_irq(unsigned int irq)
```

Interrupciones anidadas

- Las interrupciones no tienen una jerarquía
- Cuando surgen deben ser atendida en forma inmediata
- Es posible que se aniden:



- De esta forma, se gana:
  - Mayor fluidez en el trabajo de los dispositivos
  - Tener un modelo de interrupciones sin privilegios (facilita el código, lo hace portable)

Manejo de excepciones

- La estructura para el manejo de las excepciones tiene 3 pasos:
  - Salvar los registros en el kernel mode stack
  - Invocar el manejador
  - Retornar de la excepción
- Previo a la invocación del manejador se configura el error en la estructura del proceso. Posteriormente, se envía la señal al proceso para ser atendida

```
current->thread.error_code = error_code;
current->thread.trap_no = vector;
force_sig(sig_number, current);
```

- Si el proceso no implementa la rutina, se ejecuta el handler del núcleo del sistema

Bottom Halves

- Las rutinas de atención de interrupciones deben:
  - Ejecutar en forma asincrónica y, por lo tanto, interrumpir otro código (incluso otras rutinas de atención)
  - Ejecutar con su interrupción deshabilitada y, en algunos casos, con todas las interrupciones deshabilitadas. Por lo que deben ejecutar lo más rápido posible
  - Son bastante dependientes del tiempo ya que trabajan con el hardware
- No corren en el contexto de un proceso, por lo que no pueden bloquearse (no se pueden planificar)
- La rutina de atención de una interrupción debe ejecutar y hacer lo mínimo necesario, delegando para otro momento las tareas que no requieran ser realizadas cuando se genera la interrupción
- La segunda parte ejecuta con todas las interrupciones habilitadas

Bottom Halves

- Varios sistemas operativos implementan el concepto de top half y bottom half
- En el bottom half se realizan las tareas que no requieran un tiempo especifico para ejecutar o que generen una degradación del sistema
- En Linux existen tres formas de ejecutar el bottom half de una interrupción:
  - Softirqs
  - Tasklets
  - Work queues

Softirqs

- Los softirqs son compiladas estáticamente en el sistema operativo (kernel/softirq.c)
- Linux soporta hasta 32 softirq
- El núcleo solo implementa 10 (linux/interrupt.h):
  - HI_SOFTIRQ (prioridad 0)
  - TIMER_SOFTIRQ (prioridad 1)
  - NET_TX_SOFTIRQ (prioridad 2)
  - NET_RX_SOFTIRQ (prioridad 3)
  - BLOCK_SOFTIRQ (prioridad 4)
  - BLOCK_IOPOLL_SOFTIRQ (prioridad 5)
  - TASKLET_SOFTIRQ (prioridad 6)
- Son reservadas para tareas que requieran un tiempo crítico (red, discos)

Ejecución de softirqs

- Los softirq ejecutan con las interrupciones habilitadas y no pueden bloquearse
- En un procesador solamente ejecuta un softirq a la vez y solo puede cortarse por una interrupción
- En un sistema multiprocesador se permiten que ejecuten en paralelo, incluso la misma rutina de atención
- Es por eso necesario tener especial cuidado en proteger los datos globales de la rutina
- Los softirq deben ser marcadas antes de su ejecución (raising the softirq)
- El handler antes de finalizar su ejecución marca su correspondiente softirq para que sea ejecutada en el futuro

Ejecución de softirqs

- Las softirq pendientes son controladas y ejecutadas en:
  - El retorno de un handler de una interrupción (do_softirq)
  - El kernel thread ksoftirqd
  - En otras partes del código del núcleo que verifica si existen softirq pendientes (ej. subsistema de red)

Ejecución de softirqs

- La ejecución de los softirq se realiza en la función do_softirq:

```
pending = local_softirq_pending();
if (pending) {
   struct softirq_action *h;
   set_softirq_pending(0);
   h = softirq_vec;
   do {
      if (pending & 1)
         h->action(h);
      h++;
      pending >>= 1;
   } while (pending);
}
```

Kernel thread ksoftirqd

- Linux implementa un kernel thread (ksoftirqd) con nice 0
- Esto permite que, ante un potencial ráfaga de interrupciones, los procesos de usuario puedan ejecutar
- Por cada procesador se dispone de un ksoftirqd (ksoftirqd/n)
- El kernel thread verifica si existen softirq pendientes, en cuyo caso invoca a la rutina do_softirq
- Si no hay pendientes, se bloquea e invoca al planificador para que asigne a otra tarea

Tasklets

- Son construidas sobre los softirq
- Son softirq dinámicas
- Se utilizan para tareas que no requieren una urgencia de tiempo
- Son representadas por HI_SOFTIRQ y TASKLET_SOFTIRQ en los tipos de softirq
- Su estructura se define como:

```
struct tasklet_struct {
        struct tasklet_struct * next; // Próximo de la lista
        unsigned long state;       // Estado (SCHED, RUN)
        atomic_t count; // (<> 0 deshabilitada, 0 para ejecutar)
        void (*func) (unsigned long); // Puntero a función
        unsigned long data;        // argumento
}
```

Planificación de Tasklets

- Las Tasklets planificadas están en dos estructuras por procesador (tasklet_vec y tasklet_hi_vec)
- Deshabilita interrupciones locales para asegurarse que ejecuta sola
- Planifica la tarea (agrega a la estructura del procesador)
- Habilita el TASKLET_SOFTIRQ o HI_SOFTIRQ para ser ejecutado
- Restaura las interrupciones y finaliza

Ejecución de *Tasklets*

- Las Tasklets se ejecutan a través de la rutina do_softirq
- La rutina task_action invoca a todas las rutinas pendientes:
  - Deshabilitar las interrupciones
  - Obtener el número de CPU de ejecución
  - Cargar el vector en una variable local
  - Dejar el vector vacío
  - Habilitar las interrupciones
  - Para cada descriptor de tasklet hacer
    - Verificar si la tasklet está ejecutando en otro procesador
    - Si está, cargarla nuevamente para una futura ejecución
    - Si no está, marcar el estado como ejecutando para que no se pueda ejecutar en otro procesador
  - Verificar si la tasklet está habilitada
  - Ejecutar la tasklet correspondiente

Work Queues

- Las work queues se diferencian con los softirq y tasklets en:
  - Ejecutan en el contexto de un proceso, por lo tanto, pueden bloquearse
  - Son ejecutadas a través de un worker thread
- Linux provee la creación de nuevos worker thread (uno por procesador o uno único)
- El worker thread por defecto es el event, y existe uno por procesador en el sistema: event/0, event/1 …
- Son bastante útiles cuando la atención consume bastante tiempo y el trabajo se puede diferir bastante en el tiempo
- Ejemplos:
  - reiserfs
  - xfslogd, xfsdatad

Work Queues

- La estructura workqueue_struct contiene los descriptores cpu_worqueue_struct para cada procesador
- La razón de implementar un thread por CPU es con fines de eficiencia (sobre todo por la memoria cache)
- En cada descriptor se tiene una lista de las funciones pendientes a realizar por el work_queue (lista de tipo work_struct)
- Además, contiene un campo para que el worker thread espere mientras espera por trabajo
- La estructura work_struct contiene el puntero a función que realiza la tarea

Primitivas para administración

- Para manipular una work queue se tienen disponibles las siguientes primitivas:
- create_workqueue
  - Recibe como parámetro el nombre del thread
- destroy_workqueue
  - Destruye la workqueue
- queue_work
  - Agrega una función en la work queue
  - Verifica si ya está en la work queue. Si es así, retorna
  - Se agrega al final de la lista de pendientes de la work_struct
  - Si el worker thread está “durmiendo” para la CPU local, entonces se lo despierta

Work queues – 2.6.36

- Las work queues fueron modificadas a partir del kernel 2.6.36 para disminuir la cantidad de thread del kernel en ejecución (y memoria utilizada por las estructuras)
- Se sustituyo por un conjunto de kernel threads kworker/0:0
- Las tareas diferidas son asignadas a los thread a través de un algoritmo que intenta mantener un tarea ejecutando por procesador.
- Para crear una workqueue se sustituye la función create_workqueue por la función:
  - alloc_workqueue
- Recibe como parámetros el nombre, tipo (REENTRANT, UNBOUND,etc.) y el límite del número de tareas que pueden ejecutar en paralelo