INTERRUPCIONES HARDWARE EN LINUX

S. Candela

© Universidad de Las Palmas de Gran Canaria

Introducción

- Una interrupción se genera cuando se quiere que la CPU deje de ejecutar el proceso en curso y ejecute una función específica de quien produce la interrupción.
- Cuando se ejecuta esta función específica decimos que la CPU está atendiendo la interrupción.

clasificación atendiendo a la fuente

- **Interrupcion software**, se produce cuando un usuario solicita una llamada del sistema.
- **Interrupciones hardware**, son causadas cuando un dispositivo hardware requiere la atención de la CPU para que se ejecute su manejador.
- **Excepciones**, son interrupciones causadas por la propia CPU, cuando ocurre algo no deseado, por ejemplo una división por cero.

Las interrupciones hardware son producidas por varias fuentes:

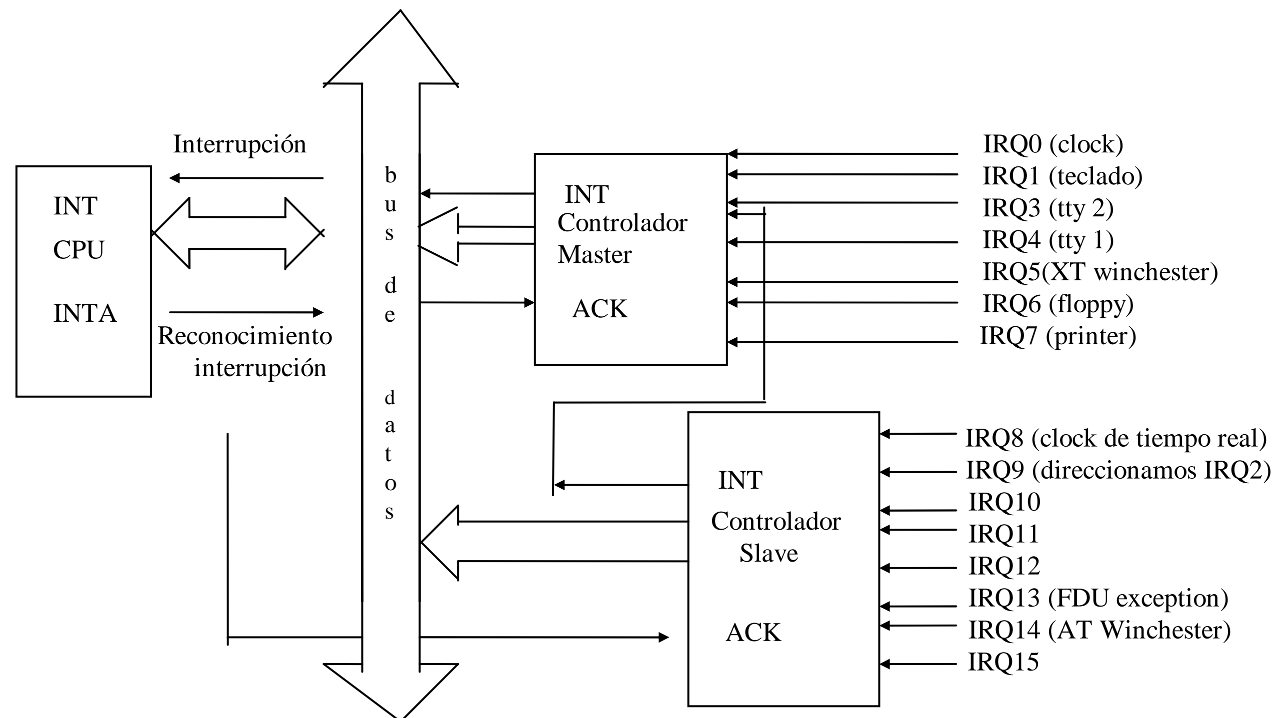
teclado, reloj, impresora, puerto serie, disquete, etc.

- Esta señal informa a la CPU que el dispositivo requiere su atención.
- La CPU parará el proceso que está ejecutando para atender la interrupción.
- Cuando la interrupción termina, la CPU reanuda la ejecución en donde fue interrumpida, pudiendo ejecutar el proceso parado originalmente o bien otro proceso.

hardware especifico para interrumpir

- La CPU tiene la entrada INT y la salida INTA para reconocer la interrupción.
- La placa base del computador utiliza un controlador para:
- Decodificar las interrupciones.
- Colocar en el bus de datos información de que dispositivo interrumpió.
- Activar la entrada INT de interrupción a la CPU.
- Proteger a la CPU y aislarla de los dispositivos que interrumpen.
- Proporcionar flexibilidad al diseño del sistema.
- Mediante un registro de estado permite o inhibe las interrupciones en el sistema.

Controladores de Interrupciones Programables PIC



IRQ interrupt request

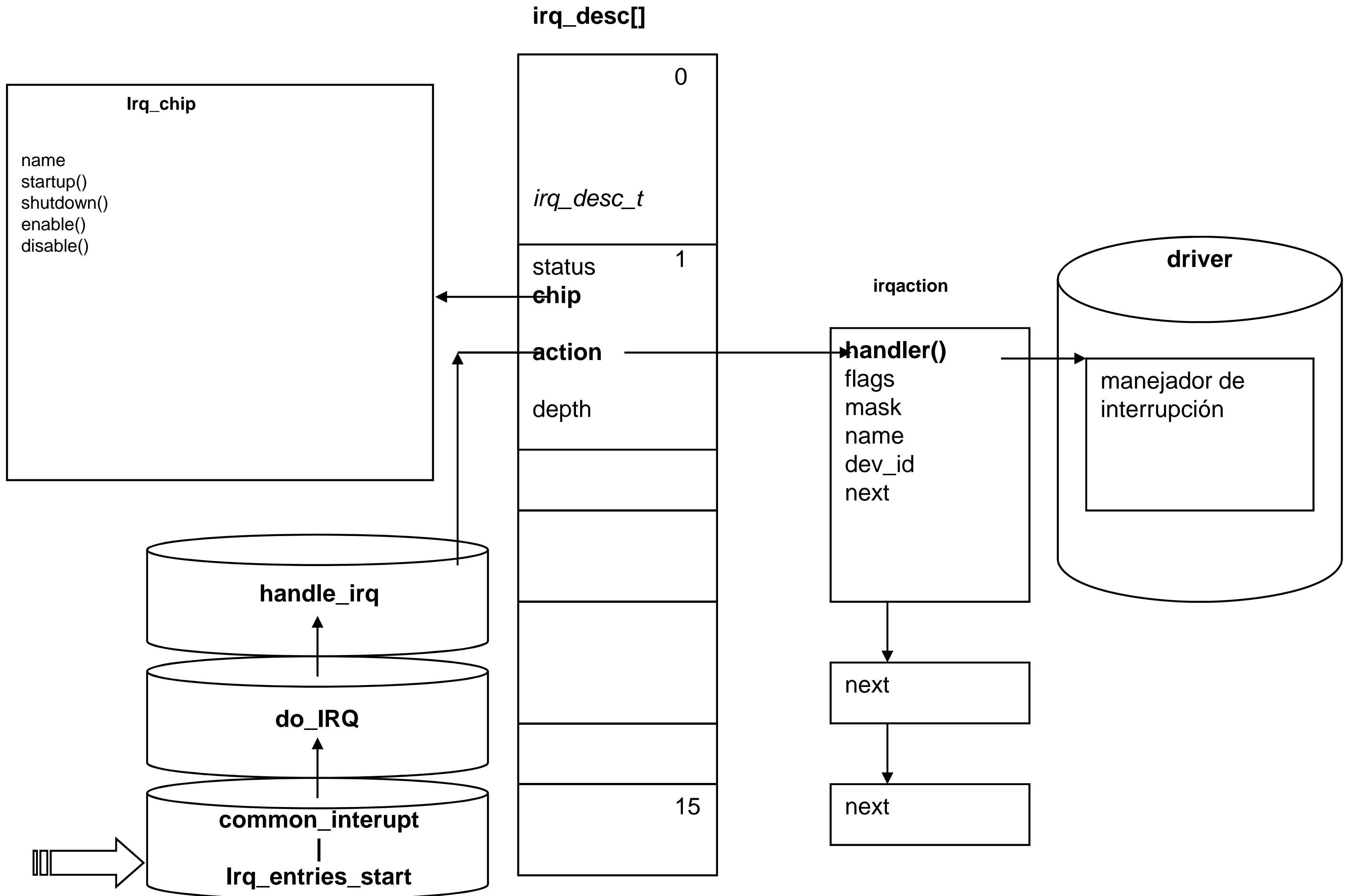
- Los IRQ son las notificaciones de las interrupciones enviadas desde los dispositivos hardware a la CPU.
- Los IRQ se encuentran numerados, y cada dispositivo hardware se encuentra asociado a un número IRQ.
- Los Controladores controlan la prioridad de las interrupciones. El reloj (en IRQ 0) tiene una prioridad más alta que el teclado (IRQ 1).
- el IRQ 2 del primer PIC, valida o invalida las entradas del Segundo PIC (8 a 15).

Interrupciones hardware en Linux.

- Linux proporciona un conjunto de estructuras de datos y de funciones para el sistema de manejo de interrupciones.
- Linux en la medida de lo posible, tratará de que sea independiente de la máquina en la que reside el sistema.
- Veamos
  1. Estructuras de datos
  2. Funciones que las manejan
  3. Flujo de una interrupción

Estructuras de datos

- **irqaction** almacena la dirección de la función de manejo de interrupciones.
- **irq_chip** contiene las funciones que manejan un controlador de interrupciones particular, es dependiente de la arquitectura.
- **irq_desc** vector con una entrada para cada una de las interrupciones que pueden ser atendidas.
- IDT+GDT


irq_desc[]
Irq_chip
name
startup()
shutdown()
enable()
disable()
0
irq_desc_t
status
1
chip
action
depth
15
irqaction
handler()
flags
mask
name
dev_id
next
next
next
driver
manejador de
interrupción
handle_irq
do_IRQ
common_interupt
Irq_entries_start

irqaction include/linux/interrupt.h

- **struct irqaction** almacena un puntero a la dirección de la función que hay que llevar a cabo cada vez que una interrupción se produce.

```
struct irqaction {
void ( * handler ) (int, void *, struct pt _regs *);
unsigned long flags ;
unsigned long mask;
const char *name;
void * dev_id;
struct irqaction *next;
};
```

Campos de irqaction

- **handler** un puntero a la función que atiende a la interrupción, es inicializado por el manejador.
- **flags** información sobre cómo tratar en ciertas situaciones la interrupción. Los valores que puede tomar son los siguientes:
  - **SA_INTERRUPT** Indica que esta interrupción puede ser interrumpida por otra.
  - **SA_SAMPLE_RANDOM** Esta interrupción puede ser considerada de naturaleza aleatoria.
  - **SA_SHIRQ** Esta IRQ puede ser compartida por diferentes **struct irqaction**
- **mask** Este campo no se usa en la arquitectura i386
- **name** Un nombre asociado con el dispositivo.
- **dev_id** número identificador único grabado por el fabricante.

```
#define PCI_DEVICE_ID_S3_868 0x8880
#define PCI_DEVICE_ID_S3_928 0x88b0
```

- **next** puntero que apunta a la próxima **struct irqaction** en la cola, sólo tiene sentido cuando la IRQ se encuentra compartida.
- **Irq** número de la línea IRQ
- **Dir** puntero al descriptor del directorio /proc/irq/n asociado con la irq

irq_chip include/linux/irq.h

Describe el decodificador de interrupciones y las funciones que lo manejan a bajo nivel.

```
struct irq_chip {
const char * typename;
void (*startup) (unsigned int irq);
void (*shutdown) (unsigned int irq);
void (*handle) (unsigned int irq, struct pt_regs * regs);
void (*enable) (unsigned int irq);
void (* disable) (unsigned int irq);
void (*mask)(unsigned int irq);
int (*set_type)(unsigned int irq, unsigned int
  flow_type);
} ;
```

Campos de irq_chip

- Esta estructura contiene todas las operaciones específicas de un determinado controlador de interrupciones
- **typename** Un nombre para el controlador.
- **startup** Permite que una entrada irq pueda interrumpir.
- **shutdown** Deshabilita una entrada irq.
- **handle** Puntero a la función que maneja e inicializa el decodificador.
- **enable y disable** Igual que **startup/shutdown** respectivamente.
- **mask** permite colocar mascaras
- **type** define la forma de interrupir

irq_desc arch/i386/kernel/irq.h

- Es un vector de estructuras de tipo **irq_desc_t**. Por cada entrada del decodificador de interrupciones habrá un elemento en el **array irq_desc[ ]**.

```
  struct irq_desc {
 irq_flow_handler_t     handle_irq;
 struct irq_chip        *chip;
 struct msi_desc        *msi_desc;
 void                   *handler_data;
 void                   *chip_data;
 struct irqaction       *action;      /* IRQ action list */
 unsigned int           status;       /* IRQ status */

 unsigned int           depth;        /* nested irq disables */
 unsigned int           wake_depth;   /* nested wake enables */
 unsigned int           irq_count;    /* For detecting broken IRQs */
 unsigned int           irqs_unhandled;
 unsigned long          last_unhandled; /* Aging timer for unhandled count */
 spinlock_t             lock;
 const char             *name;
}
```

Campos de irq_desc

- **status** Estado en el que se encuentra la línea de interrupción IRQ:

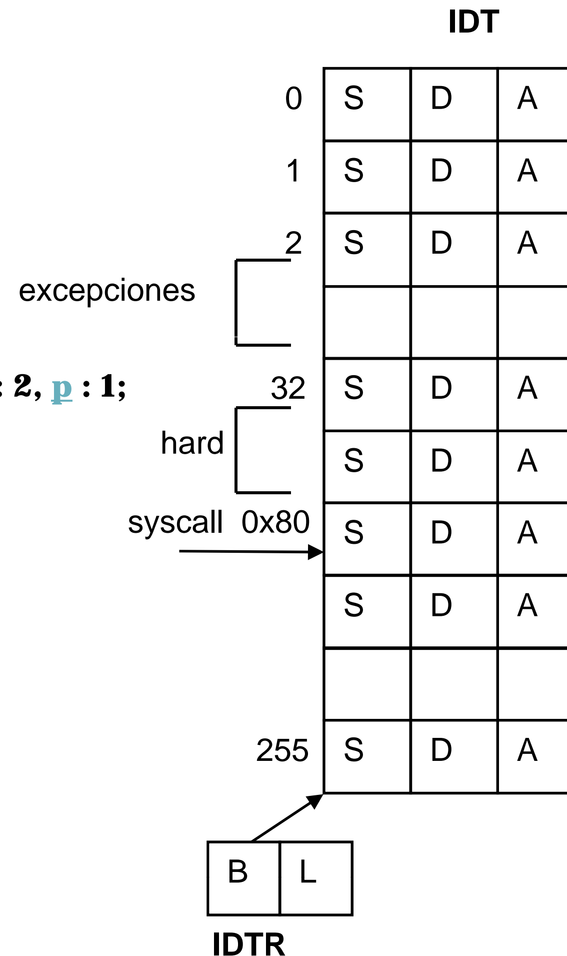
```
#define IRQ_INPROGRESS    1  /* IRQ activa, no entrar! */
#define IRQ_DISABLED      2  /*IRQ no permitida, no entrar! */
#define IRQ_PENDING       4  /* IRQ pendiente, reintentar */
#define IRQ_REPLAY        8  /* IRQ reintenta pero todavía no atendida */
#define IRQ_AUTODETECT   16  /* IRQ esta siendo auto detectada */
#define IRQ_WAITING      32  /* IRQ no lista para auto detección */
```

- **chip** Un puntero a la estructura **chip_irq**, que será el controlador que está asociada la IRQ.
- **action** Un puntero a la cabeza de la lista de **irqactions**. Una irq puede ser compartida por varios equipos.
- **depth** El número de equipos enlazados a esta irq.

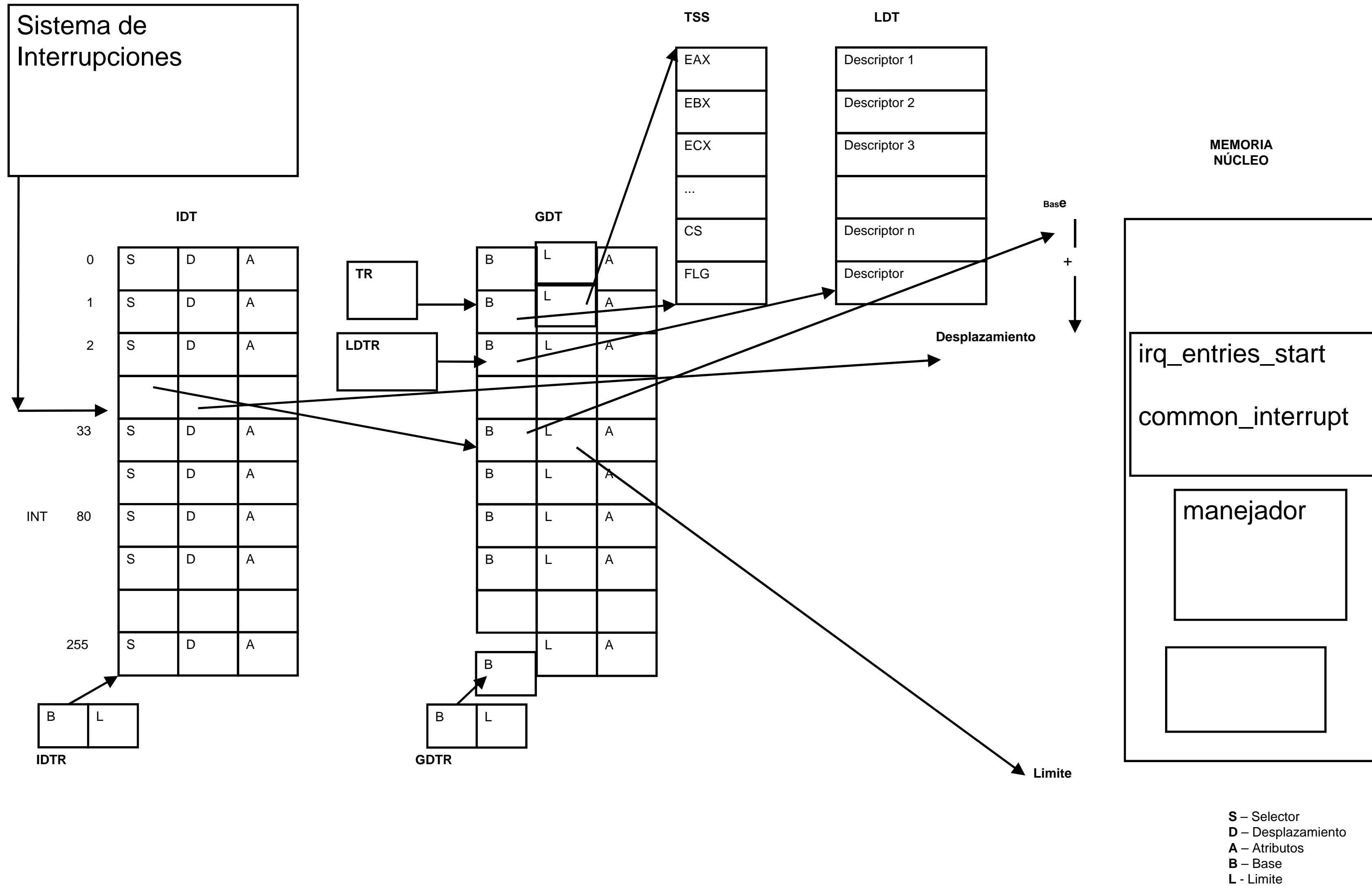
Tabla descriptora de interrupciones IDT

- Es una tabla que junto con la tabla global de descriptores GDT, nos va a llevar a las rutinas de manejo de interrupciones Definida en arch/x86/kernel/traps.c, como gate_desc idt_table[256]. Y cada entrada de la tabla es una estructura definida en,
- arch/x86/include/asm/desc_defs.h
- Como:

```
/* 16byte gate */
struct gate_struct64 {
       u16 offset_low;
       u16 segment;
       unsigned ist : 3, zero0 : 5, type : 5, dpl : 2, p : 1;
       u16 offset_middle;
       u32 offset_high;
       u32 zero1;
} __attribute__((packed));
```

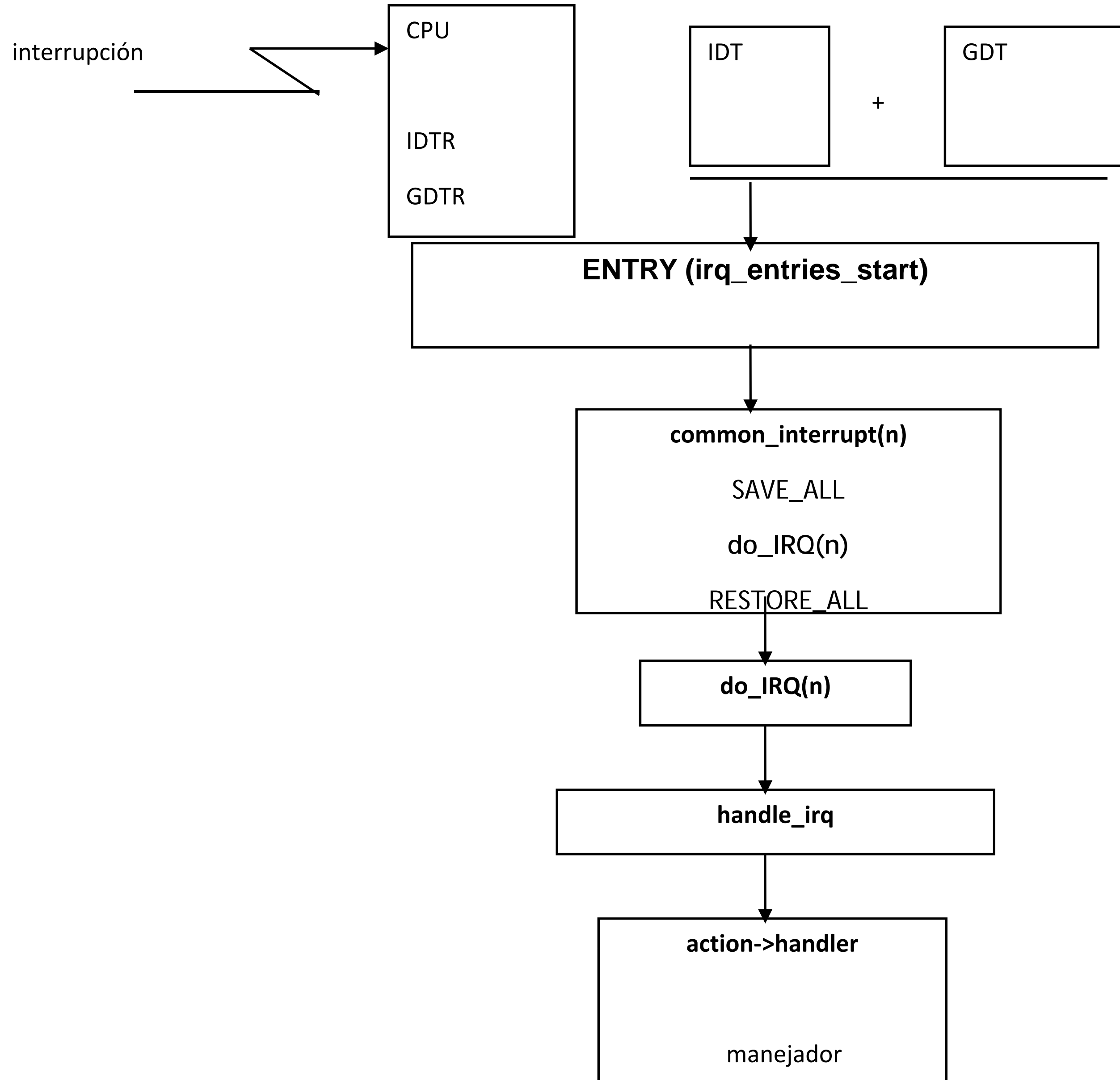


GDT y Salto a la función manejadora



S – Selector
D – Desplazamiento
A – Atributos
B – Base
L - Limite

Flujo general



Flujo general

- 1. El decodificador de interrupciones recibe una interrupción (la patilla IRQ a nivel alto), este interrumpe a la CPU, que automáticamente mediante la entrada que corresponde a esta interrupción y con la ayuda de la IDT y la GDT nos lleva a **ENTRY (irq_entries_start)**, que salta a **common_interrupt**.

- 2. **common_interrupt** despues de guardar el estado del proceso (**save_all**) llama a **do_irq** pasándole el número de interrupción para ejecutar la función manejadora de la interrupción y posteriormente recupera el estado del proceso interrumpido **RESTART_ALL** .

- 3. **do_IRQ** llama a **handle_IRQ**.

3.3.1 Flujo general

- 4. **handle_IRQ** llama a las funciones necesarias para atender la interrupción **action->handler** dentro del manejador .
- 5. **action->handler** se ejecuta la función manejadora de la interrupción dentro del manejador (driver) del dispositivo que interrumpió.
- 6. **ret_from_intr** restaura el estado del proceso interrumpido RESTORE_ALL y continúa con la ejecución normal.

Flujo proceso a proceso

- La función **set_intr_gate** inicializar la IDT, colocando el desplazamiento que nos lleva al punto de entrada dentro del fichero entry_32.S
- **irq_entries_start** nos lleva a **common_interrupt** pasándole como parámetro el número de interrupción en el stack cambiado de signo.

```
ENTRY (irq_entries_start)
/* guarda el número de la interrupción en el stack, forma
   de pasar parámetros a una función, y en negativo para
   que no exista conflicto con el número de señales */
        pushl $-(vector)
        jmp common_interrupt
```

- donde vector corresponde con el índice del vector

common_interrupt

common_interrupt es una función en ensamblador que realiza los tres pasos fundamentales para atender una interrupción:

- **SAVE_ALL** almacenar el estado del proceso que se interrumpe.
- **call do_IRQ** llamar a la función que atiende la interrupción pasándole en eax el número de la interrupción.
- **RESTORE_ALL** saltando a **ret_from_intr** restablece el proceso interrumpido

do_IRQ

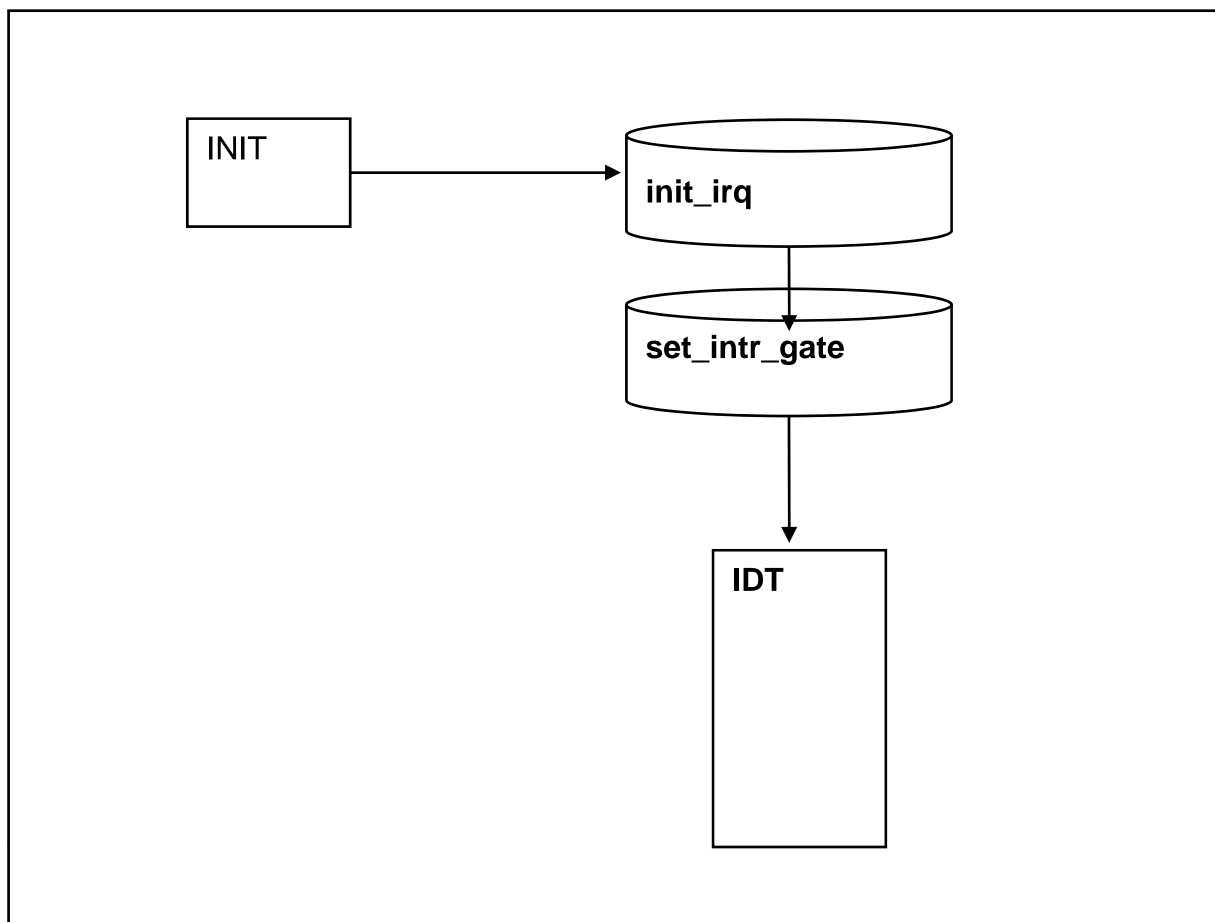
```
fastcall unsigned int do_IRQ(struct pt_regs *regs)
{
  /* el numero de interrupción se encuentra en eax */
     int irq = ~regs->orig_eax;
/* accede a la posición del vector irq_desc para esta interrupción */
      struct irq_desc *desc = irq_desc + irq;
/* macro que incrementa un contador con el número de interrupciones
      anidadas */
     irq_enter();

/* llamada a la función con el número de irq y un puntero a la
      estructura con los registros */
     desc->handle_irq(irq, desc);

 /* macro que decrementa el contador */
    irq_exit();
    return 1;
}
```

init_IRQ().

- llamada por el programa de inicio del sistema **init start_kernel ()**, en **init/main.c**



Esta función pone a punto la tabla descriptora de interrupciones (IDT) y las estructuras de datos internas. Inicializa el vector de interrupciones hardware irq_desc[0..15], inicializa la tabla IDT, llamando a la función **set_intr_gate()**.

```
unsigned long __init  init_IRQ (unsigned long memory) {
   int i;
 pre_intr_init_hook();    /* inicializa el vector irq_desc[0..15]  */
   /* Para todas la interrupciones inicializa la IDT.*/
      for (i = 0; i < (NR_VECTORS - FIRST_EXTERNAL_VECTOR); i++) {
            int vector = FIRST_EXTERNAL_VECTOR + i;
            if (i >= NR_IRQS)
                  break;
            if (vector != SYSCALL_VECTOR)
                  set_intr_gate(vector, interrupt[i]);

 /* interrupt[i] desplazamiento a la primera función donde salta  cuando
   llega  la interrupción */
}
```

set_intr_gate

- Por medio de la función **set_intr_gate** se rellena la tabla de descriptores de interrupciones (**IDT**). Básicamente esta es la parte en la que se realiza el "cableado" de las interrupciones hardware dentro del sistema, a partir de este momento, cualquier interrupción hardware será atendida. Se encuentra en el fichero arch/i386/kernel/traps.c.

```
void set_intr_gate(unsigned int n, void *addr)
{
  _set_gate(n, DESCTYPE_INT,addr, KERNEL_CS);
}
```

n va apuntando a las distintas entradas de la IDT.

DESCTYPE_INT tipo de descriptor interrupcion

addr desplazamiento que le va a llevar a la función irq_entries_start.

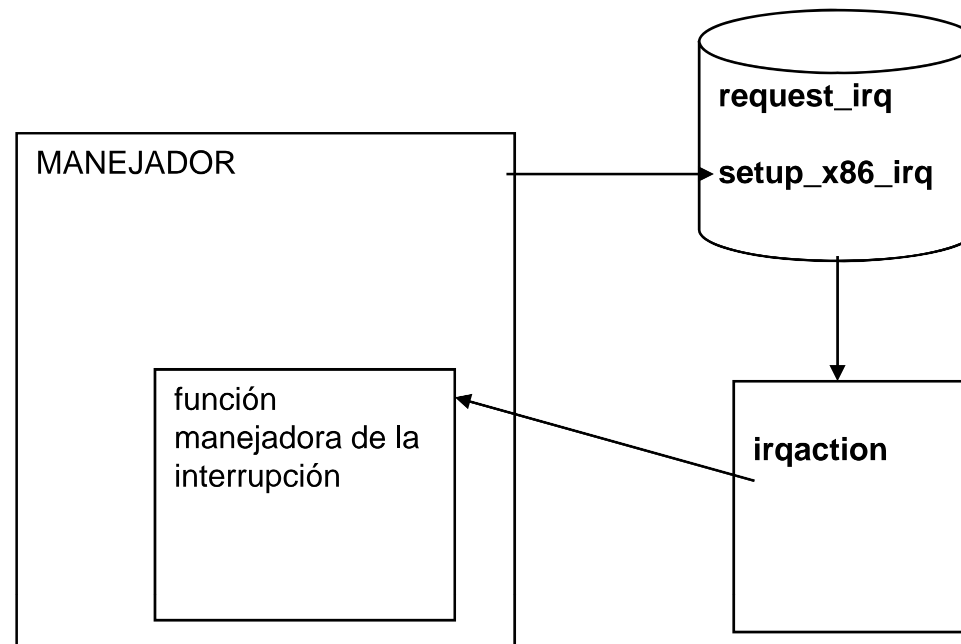
KERNEL_CS selector que apunta a la entrada de la GDT donde está la base del código del núcleo.

set_gate

```
static inline void _set_gate(int gate, unsigned int type, void *addr, unsigned
    short seg)
{       __u32 a, b;
     pack_gate(&a, &b, (unsigned long)addr, seg, type, 0);
    write_idt_entry(idt_table, gate, a, b);
}
static inline void pack_gate(__u32 *a, __u32 *b,
      unsigned long base, unsigned short seg, unsigned char type, unsigned char
    flags)
{
      *a = (seg << 16) | (base & 0xffff);
      *b = (base & 0xffff0000) | ((type & 0xff) << 8) | (flags & 0xff);
}
static inline void write_dt_entry(struct desc_struct *dt,
                        int entry, u32 entry_low, u32 entry_high)
{
      dt[entry].a = entry_low;
      dt[entry].b = entry_high;
}
```

request_irq

- Los manejadores de dispositivos usan las funciónes **request_irq** y **setup_x86_irq** para colocar la dirección de la función manejadora de interrupción propia.



request_irq

Crea un nuevo nodo en la lista irqaction con los valores suministrados para una IRQ

```
int request_irq(unsigned int irq, irq_handler_t handler,
          unsigned long irqflags, const char *devname, void *dev_id)
{
  struct irqaction * action;
  int retval;
 /* comprueba los valores de entrada irq y handler */
  if ((irqflags & IRQF_SHARED) && !dev_id)  /* interrupción compartida
          return -EINVAL;
  if (irq >= NR_IRQS)      /*se sobrepasa el nº de IRQ.
          return -EINVAL;
  if (!handler)      /* existe una función  manejadora
          return -EINVAL;

/* con la función kmalloc asigna memoria dinamicamente para el nuevo nodo
   */
  action = (struct irqaction *)
    kmalloc(sizeof(struct irqaction), GFP_KERNEL);
  if (!action)
   return -ENOMEM;
```

request_irq

```
/* llena los campos de action con la nueva función
  action->handler = handler;
  action->flags = irqflags;
  cpus_clear (action->mask);0;
  action->name = devname;
  action->next = NULL;
  action->dev_id = dev_id;
/* iañade el nuevo nodo a la lista con setup_x86_irq
  retval = setup_x86_irq(irq, action);

  if (retval)
    kfree(action);
  return retval;
 }
```

setup_irq recibe un nodo **irq_action** y un número de interrupción **n** y rellena la estructura **irq_desc**

```
int setup_x86_irq (unsigned int irq, struct irqaction * new)
{
   int shared = 0;
   struct  irqaction    * old,   **p;

p = &desc->.action;
   old = *p /* nodo de la lista no vacío, existe ya una interrupción */
If(old) {
       /* No pueden compartir interrupciónes a menos que estén de acuerdo */
     if (!((old->flags & new->flags) & IRQF_SHARED) ||
                ((old->flags ^ new->flags) & IRQF_TRIGGER_MASK)) {
                   old_name = old->name;
                   goto mismatch;
     }
/* añade una nueva interrupción al final de la cola  irq   coloca a uno  shared
    */
do  {
        p = &old->next ;
        old = *p;
  } while (old);
  shared = 1;
```

```
}
/* p ahora apunta al último elemento de la lista campo “next” si la IRQ
   no es
 * compartida o a irq_desc[irq].action,  si no es compartida. */
*p = new
/* si no se habián establecido acciones en esta irq inicializarla */
if  (!shared) {
desc->chip && desc->chip->set_type
desc->status &= ~(IRQ_AUTODETECT | IRQ_WAITING |
   IRQ_INPROGRESS);
desc->depth = 0;
 desc->status &= ~(IRQ_DISABLED;
 desc->chip->startup  /* permite interrumpir por irq */
desc->irq_count = 0;
 desc->irqs_unhandled = 0;
}
return 0;
}
```

free_irq

Llamada por los manejadores para quitar un nodo de la lista irqaction, es la acción inversa de request_irq

```
void free_irq(unsigned int irq, void *dev_id)
{
  struct irqaction * action, **p;
  unsigned long flags;
/* Obtiene la entrada correspondiente en irq_desc y avanza por la lista hasta
   encontrar el dispositivo para realizar las acciones */
p = &desc->action;
      for (;;) {
            struct irqaction *action = *p;
            if (action) {
                  struct irqaction **pp = p;
                  p = &action->next;
                  if (action->dev_id != dev_id)  /* es el ID del dispositivo */
                        continue;
                  /* encuentra el dispositivo quita el nodo de la lista */
                  *pp = action->next;
```

```
/* si la cola tenía un solo elemento la entrada irq es deshabilitada */
    if (!desc->action) {
            desc->status |= IRQ_DISABLED;
            if (desc->chip->shutdown)
                    desc->chip->shutdown(irq);
            else
                    desc->chip->disable(irq);
    }
    spin_unlock_irqrestore(&desc->lock, flags);
/* borra el registro del manejador en el sistema de ficheros /proc */
  unregister_handler_proc(irq, action);
/* se asegura que no está siendo usada por otra CPU */
  synchronize_irq(irq);
   if (action->flags & IRQF_SHARED)
            handler = action->handler;
/* Elimina el elemento de la lista y libera memoria */
    kfree(action);
    return;
}
```