

# ***Arquitectura de Computadores***

## **Tema 3: Sistema de Entrada/Salida**

### ***ANEXO: Manejo de Periféricos en C***

**Grado en Ingeniería Informática**

- 1. Acceso a un puerto en C bajo Linux (E/S separada).**
- 2. Operaciones a nivel de bits.**
- 3. Manejo de Interrupciones.**

- **Funciones de acceso a puertos de E/S en C bajo Linux**

- Envío de datos (escritura en el puerto)

- Registro de 8 bits

- `void outb (unsigned char valor, unsigned short puerto) ;`

- Registro de 16 bits

- `void out (unsigned short valor, unsigned short puerto) ;`

- Recepción de datos (lectura en el puerto)

- Registro de 8 bits

- `unsigned char inb (unsigned short portid) ;`

- Registro de 16 bits

- `unsigned short in (unsigned short portid) ;`

**puerto:** Dirección del puerto al que acceder.

**valor:** Dato a enviar al dispositivo (escribir en el puerto).

- **Máscaras de bits**

- En el Lenguaje C el tipo de datos más pequeño es ***char*** (8 bits)
- Cuando se accede a un registro de E/S en C se leen o escriben 8 bits (al menos) generalmente sin signo (***unsigned char***).
  - También es posible acceder a 16 bits (***short int***) con o sin signo.
- Habitualmente es necesario leer o modificar el valor de un subconjunto de bits del registro accedido.
  - Para ello, se realizan **operaciones lógicas** que toman como primer operando el registro accedido y como segundo un valor (una secuencia de bits) denominado “***máscara***”.
  - Para utilizar máscaras se utilizarse la representación hexadecimal (utilizando la notación 0x) aunque también puede utilizarse la decimal, pero no existe la notación en binario.

- Ejemplos básicos:

### con AND lógico:

Dada una secuencia de 4 bits,  
establecer el 1<sup>er</sup> bit a cero.

$$\begin{array}{rcccc}
 & b3 & b2 & b1 & b0 \\
 \& & & & \\
 & 1 & 1 & 1 & \mathbf{0} \text{ (máscara)} \\
 \hline
 & b3 & b2 & b1 & \mathbf{0}
 \end{array}
 \quad (\text{AND})$$

Para leer el valor de un determinado bit, se puede usar el operador ‘&’ y una máscara con todos sus bits a 0, excepto el bit que ocupa la posición que se pretende leer.

### con OR lógico

Dada una secuencia de 4 bits,  
establecer el 2<sup>o</sup> bit a uno.

$$\begin{array}{rcccc}
 & b3 & b2 & b1 & b0 \\
 | & & & & \\
 & 0 & 0 & \mathbf{1} & 0 \text{ (máscara)} \\
 \hline
 & b3 & b2 & \mathbf{1} & b0
 \end{array}
 \quad (\text{OR})$$

Para leer el valor de un determinado bit, se puede usar el operador ‘|’ y una máscara con todos sus bits a 1, excepto el bit que ocupa la posición que se pretende leer.

### Ejemplo 1: consulta del valor de un registro

```
#define DIRECCION_REG_ESTADO 0x00A0
...
int main ()
{
    char registro_estado;
    ...
    registro_estado = inb (DIRECCION_REG_ESTADO);
    // Se comprueba el bit menos significativo si
    // está a cero o uno:
    if (registro_estado & 0x01)
    {
        ...
    }
    ...
}
```

### Ejemplo 2: modificación del valor de un registro

```
#define DIRECCION_REG_CONTROL 0x00A1
...
int main ()
{
    char registro_control;
    ...
    // Cambia los dos últimos bits a 0 del reg. control:
    registro_control = inb (DIRECCION_REG_CONTROL);
    registro_control = registro_control | 0xFD;
    outb (registro_control, DIRECCION_REG_CONTROL);
    ...
}
```

En lenguaje C: 1, 0x01, 0x0001 y 0x00000001, es lo mismo.

- **Establecimiento de un manejador de interrupciones (rutina de servicio de interrupción):**

```
int request_irq (int numirq, void (*handler), char *device [, void *dev_id]) ;
```

- **numirq:** número de interrupción a utilizar.
- **handler:** manejador de la interrupción. Puntero a la rutina de tratamiento de la interrupción.
- **device:** cadena con el nombre del dispositivo (fichero de dispositivo).
- **dev\_id:** usado para compartir líneas de interrupción. Si no se comparte no se indica y, en caso contrario, suele pasarse un puntero al driver del dispositivo.
- **Retorno:** Si 0 → correcto.

- **Liberación de una interrupción:**

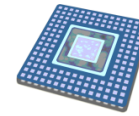
```
void free_irq(int numirq [, void *dev_id]) ;
```

- **numirq:** número de interrupción a liberar.
- **dev\_id:** usado para compartir líneas de interrupción.



- **Habilitar/Deshabilitar interrupciones:**
  - Habilitar una interrupción específica:  
***void enable\_irq(int irq);***  
*irq*: número de la interrupción a habilitar.
  - Deshabilitar interrupción específica:  
***void disable\_irq(int irq);***  
*irq*: número de la interrupción a deshabilitar.
  - Deshabilitar interrupciones generales:  
***void disable();***
  - Habilitar interrupciones generales:  
***void enable();***

### 3. Manejo de Interrupciones: Ejemplo



```
void main ()
{
    int a;
    a = request_irq (num_IRQ, rutina_int, "/dev/midisp");
    if (a != 0)
        printf ("ERROR\n");
    else {
        inicializacion_dispositivo ();
        enable_irq (num_IRQ); // Habilitar interrupción específica
    }
}
```

```
void rutina_int ()
{
    disable(); // Evitar que ser interrumpido
    ...       // Realizar la transferencia
    enable();  // Reactivar las interrupciones
}
```

```
void inicializacion_dispositivo ()
{
    // Configurar el dispositivo
    // Habilitar interrup en el dispositivo
}
```