Enunciado. Sea un dispositivo conectado a un computador que utiliza E/S separada y que mapea el dispositivo a partir de la dirección base 0x80. Los registros son los siguientes:

7	6	5	4	3	2	1	0
			IRQ				P

Registro de Control (dir 0x80)

P: Envío de petición de lectura (1).

IRQ: Configura el dispositivo para ser utilizado mediante interrupciones.

7	6	5	4	3	2	1	0
DR		ERR	E2	E1	E0		BS

Registro de Estado (dir 0x81)

BS: Bit de ocupado (Busy). Antes de escribir en el registro de control debe verificarse que el bit BS está a 0.

ERR: El dispositivo lo establece a 1 si petición de lectura no se ha podido realizar. En ese caso, los bits E2-E0 especifican un código de 3 bits del error que se ha producido.

DR: Bit de dato disponible (Data Ready). El dispositivo lo establece a 1 cuando la petición de lectura se ha procesado correctamente.

7	6	5	4	3	2	1	0

Registro de Datos 1 (parte alta, dir 0x82)

7	6	5	4	3	2	1	0

Registro de Datos 2(parte baja, dir 0x83)

a) Realice una función que lea un dato de 16 bits del dispositivo utilizando E/S Mapeada en C bajo Linux suponiendo que ya está preconfigurado para no utilizar Interrupciones (IRQ → 0)

b) Realice un programa que lea un dato de 16 bits del dispositivo utilizando interrupciones suponiendo que el dispositivo se configura con el vector de interrupción 8.
Tenga en cuenta que la interrupción se dispara por activación del bit DR y/o el bit ERR (ambos bits son fuente de interrupción).

Solución.

a)

```
unsigned short leeDato()
{
      // Espera activa hasta que no esté ocupado.
      while (  ( inb(0x81) & 1 ) != 0 );

      // Envia la petición de lectura
      outb (  inb(0x80) | 1 , 0x80);

      // Espera activa hasta que finalice la petición de lectura o se produzca error.
      while(  ( inb(0x81) & 0xA0 ) == 0);

      // Si se produjo error
      if ( (  inb(0x81) & 0x20 ) != 0)
      {     // Muestra el código del error que se ha producido
            printf("Se ha producido un error con código %d", (inb(0x81) >> 2) & 0x07);
            return 0;
      }
      // Si no se produjo error, se activó DR por lo que se realizó la lectura con éxito.
      else{
            return ((inb(0x82)<<8)|(inb(0x83)));
      }
}
```

b)

```
unsigned short dato;
unsigned char codError;

int main()
{     int res;
      // Registrar la interrupción.
      res = request_irq( 8, miRSI, "/dev/midispositivo");
      if (res != 0)
      {     printf("Error registrando la interrupción");
            return 0;
```

```
        }

        // Espera activa hasta que no esté ocupado para poder configurar el dispositivo.
        while (  ( inb(0x81) & 1) != 0);

        // Activa el uso de interrupciones en el dispositivo
        outb( inb(0x80) | 0x10, 0x80);

        // Habilita la interrupción 8 en la máscara de interrupciones.
        enable_irq(8);

        // Vuelve a esperar a que no esté ocupado antes de enviar la petición de lectura
        while (  (  inb(0x81) & 1 ) != 0);

        // envía la petición de lectura
        outb (  inb(0x80) | 1 ,  0x80);

        // En este punto puede dedicarse a ejecutar instrucciones
        //   no relacionadas con el dispositivo.
        …
}

void  miRSI()
{
        unsigned char estado;

        // Deshabilita interrupciones globales para no sea interrumpida
        disable();

        // Lee el registro de estado para determinar la fuente de interrupción.
        estado = inb(0x81);

        // Si se produjo error, guarda el código en la variable 'codError'
        if ( ( estado & 0x20 ) != 0)
                codError = (  inb(0x81) >> 2 ) & 7;
        // Si no se produjo error (interrupción por activación de DR), guarda en la variable
        'dato'
        else
                dato = (inb(0x82)<<8)|inb(0x83);

        // Habilita las interrupciones globales nuevamente.
        enable();
}
```