EJERCICIO 12

POR POLLING

```
//se definen como constantes las direcciones de los puertos de E/S,
//asociada al dispositivo
#define REG_DATOS         0x1000
#define REG_CONTROL       0x1001
#define numIRQ            27

//Prototipo de funciones usadas en el programa
int inicializar_dispositivo (int set_irq);
unsigned char leer_byte();//E/S programada
int desactivar_dispositivo (int set_irq);

int main()//Función principal
{
      unsigned char v[1024];
      int cont_byte=0;

      inicializar_dispositivo(0); //E/S programada
      while(cont_byte<1024)
      {
            v[cont_byte] = leer_byte();
            cont_byte++;
      }
      desactivar_dispositivo(0); //E/S programada
      return 0;
}
// Implementación de funciones
//  Inicialización: se contempla el hecho de que se quiera inicializar el
//  dispositivo con uso de interrupciones(set_irq=1) o E/S programada(set_irq=0).
int inicializar_dispositivo (int set_irq){
      int fallo, dev = 0;

      if (set_irq==0) { //Activo dispositivo, interrupción desactivada
            outb(inb (REG_CONTROL) | 0x01, REG_CONTROL);
      }
      else{
            fallo = request_irq (numIRQ, rutinaISR, "/dev/dispositivo");
            if (fallo) { //fallo!=0
                  printf ("Error al establecer la interrupción\n");
                  dev = -1;
            } else {
                  outb(inb (REG_CONTROL) |  0x03, REG_CONTROL);
                  enable_irq(numIRQ);
            }
      }
      return dev;
}
unsigned char leer_byte (){
      int cont_bit=0;
      unsigned char dato=0;

      while (cont_bit<8)
      {
            // Se espera flanco de bajada de R
            while (inb (REG_DATO) & 0x02 !=0);
            dato = (dato<<1) | inb(REG_DATO)& 0x01;
            cont_bit++;
            // Se espera flanco de subida de R para evitar que se lea 2 veces el mismo bit
            while (inb (REG_DATO)  & 0x02 ==0);
      }
      return dato;
```

```
}
void desactivar_dispositivo (int set_irq){
	if (set_irq==0){ //Desactivo dispositivo
		outb(inb (REG_CONTROL) & 0xFE, REG_CONTROL);
	}
	else{ //Desactivo dispositivo, deshabilito interrupción
		outb(inb (REG_CONTROL) & 0xFC, REG_CONTROL);
		disable_irq(numIRQ);
	}
}
```

POR INTERRUPCIONES:

```
#define REG_DATOS		0x1000
#define REG_CONTROL		0x1001
#define numIRQ			27

//Prototipo de funciones usadas en el programa
int inicializar_dispositivo (int set_irq);
int desactivar_dispositivo (int set_irq);
void  rti (); //E/S interrupcion

//Declaro variables globales que se usaran en la RTI
unsigned char v[1024];
int character_leido=0;
unsigned char dato=0;
int cont_bit=0;
int cont_byte=0;

int main()
{
	int ret;
	ret = inicializar_dispositivo(1); //E/S interrupción
	if(ret!=0){
		printf (“No se pudo iniciar la operación\n”);
	}
	return 0;
}

void  rti ()
{
	disable();
	dato = (dato<<1) | inb(REG_DATO)& 0x01;
	cont_bit++;
	if(cont_bit==8){
		v[cont_byte] = dato;
		cont_bit=0;
		dato=0;
		cont_byte++;
		if(cont_byte==1024)
		{
			desactivar_dispositivo(1);
		}
	}
	enable();
}
```

EJERCICIO 30

Inicialización:

```
#define REG_EST-CONTR_TEMP 0xA0
#define REG_DATOS_TEMP 0xA1
#define T_ESPERA 100
...

int inicializar_sensor_temperatura (int modo) {
	char reg_estado-control;

	// Comprobar si no está READY, en cuyo caso hacer RESET y esperar a READY:
	if ( (inb (REG_EST-CONTR_TEMP) & 0x40) != 0) {
		outb (inb (REG_EST-CONTR_TEMP) | 0x02,  REG_EST-CONTR_TEMP);
		while((inb (REG_EST-CONTR_TEMP)&0x40) != 0);
	}

	// Indicarle el modo de funcionamiento (interrupciones o polling):
	// Se pone el bit menos significativo a 0:
	reg_estado-control = inb (REG_EST-CONTR_TEMP) & 0xFE;

	// Se establece el bit menos significativo para el modo de funcionamiento:
	// Se supone que 'modo' vale 0 (polling) o 1 (interrupciones). Aun así, nos quedamos
	// con el bit menos significativo para evitar errores.
	outb ( reg_estado-control | (modo & 0x01),  REG_EST-CONTR_TEMP);

	// Devolver si ha habido error:
	return (inb (REG_EST-CONTR_TEMP) & 0x10 )>>4);
}
```

Control por *polling*:

```
#define REG_EST-CONTR_DIV 0xF0

void modificar_divisor_reloj (int sentido);
unsigned char leerTemperatura () ;

void main  () {
	unsigned char temp;
	int dev;
	dev = inicializar_sensor_temperatura (0);

	// Inicilaizar el sensor de temperatura:
	if (dev != 0) {
		// Ha habido un error. Hay que salir.
		printf("Error…\n");
	}
	else
	{
		while (1)
		{
			// Leer la temperatura:
			temp = leerTemperatura();

			// Ajustar divisor frecuencia:
			if (reg_datos > 150) {
				// Aumentar el divisor de reloj:
				modificar_divisor_reloj (0);
			} else if (reg_datos < 100) {
				// Disminuir el divisor de reloj:
				modificar_divisor_reloj (1);
			}
		}
	}
}
```

```
unsigned char leerTemperatura()
{
	// 1- Esperar si está "Busy":
	while (inb (REG_EST-CONTR_TEMP) & 0x08);

	// 2- Realizar una petición de dato (señal REQ):
	outb (inb (REG_EST-CONTR_TEMP) | 0x04, REG_EST-CONTR_TEMP);

	// 3- Esperar que nos avise de dato nuevo on 'NEW':
	while (inb (REG_EST-CONTR_TEMP) & 0x20));

	// 4- Leer registro datos:
	 return (inb (REG_DATOS_TEMP));
}

void modificar_divisor_reloj (int sentido) {
	char reg_estado-control;

	// 1- Esperar a que no haya operación prioritaria ejecutándose (PRIOR):
	while (inb (REG_EST-CONTR_DIV) & 0x40));

	// 2- Indicar si se va a aumentar o disminuir (UP/DOWN):
	// 2.1 – Se pone el bit UP/DOWN a cero:
	reg_estado-control = inb (REG_EST-CONTR_DIV) & 0xFD;
	// 2.2 – Se modifica el bit UP/DOWN:
	outb (reg_estado-control | (sentido <<1), REG_EST-CONTR_DIV);

	// Activar el bit MOD:
	outb (inb (REG_EST-CONTR_DIV) | 0x04, REG_EST-CONTR_DIV);
}
```