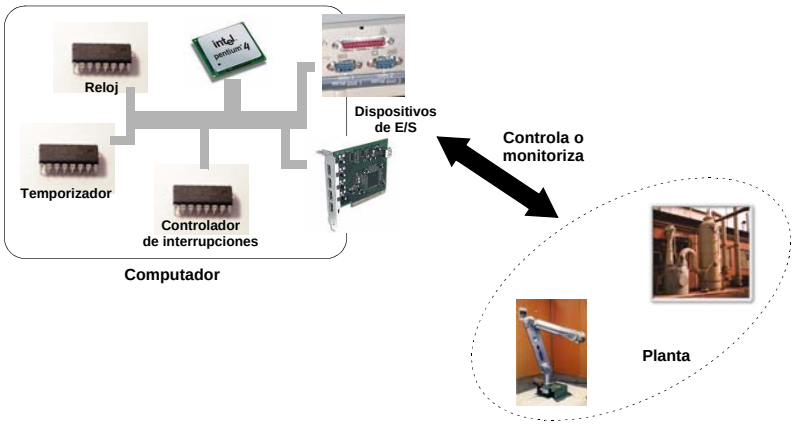
Master en Computación

Plataformas de Tiempo Real

Bloque II: Manejadores de dispositivos y drivers

Tema 4. Programación de E/S en MaRTE OS

4.1 Relación entre el computador y su entorno



Los dispositivos de entrada/salida ponen en contacto al computador con su entorno

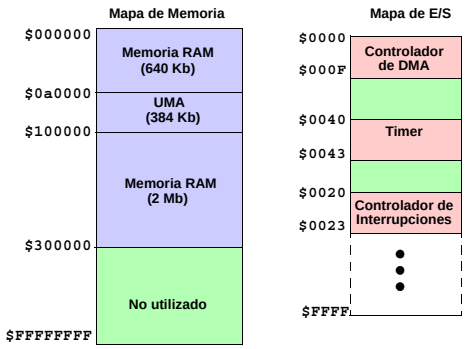
Una parte fundamental del software de un sistema empotrado será el código encargado de controlar los dispositivos de E/S

- **el código encargado de controlar un dispositivo se denomina *manejador de dispositivo* (o "driver")**

Acceso al hardware

En un PC los mapas de memoria y de E/S son independientes

- **instrucciones ensamblador diferentes para acceder a la memoria (`MOV`, `XCHG`, etc.) o a los registros de E/S (`IN`, `OUT`, etc.)**
- **la línea `MEM/IO` del bus de control selecciona el mapa**



Los registros de los dispositivos pueden encontrarse tanto mapeados en memoria como en el mapa de E/S

El acceso a un registro mapeado en memoria se realiza utilizando un puntero que apunte a la dirección de memoria deseada

```
// registro en la dirección de memoria 0xFE000100
char *punt_reg = (char *)0xFE000100;
*punt_reg = 0x2A; // escribe 0x2A en el registro
```

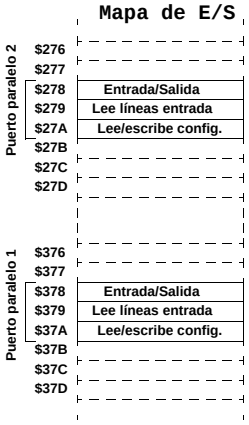
El acceso a los registros de un dispositivo en el mapa de E/S se consigue con las funciones definidas en `<sys/pio.h>`

```
// lee un byte de la direc. de E/S indicada por port
char inb(unsigned short port)
// escribe un byte en la direc. indicada por port
outb(unsigned short port, char val)
```

- **también hay funciones leer y escribir 2 bytes (`inw`, `outw`) y 4 bytes (`inl`, `outl`)**

Ejemplo de dispositivo en E/S: puerto paralelo

Dirección de E/S puerto1/puerto2	Registro	Modo
$378/$278	Entrada de Datos	Lect.
$378/$278	Salida de Datos	Esc.
$379/$279	Lee valor líneas auxiliares de entrada	Lect.
$37A/$27A	Lee configuración y líneas auxiliares de salida	Lect.
$37A/$27A	Establece configuración y valor de líneas auxiliares de salida	Esc.



Puerto paralelo: Registro de configuración

Permite leer y escribir el estado de las 4 líneas auxiliares de salida

También permite configurar el funcionamiento de la puerta:

- **puerta de salida (`Output=>0`) o de entrada (`Output=>1`)**
- **interrupción habilitada (`IRQ enable=>1`)**

X	X	$\overline{\text{Output}}$	IRQ enable	$\overline{\text{Select}}$	Init	$\overline{\text{Auto}}$	$\overline{\text{Strobe}}$

Ejemplo: habilita las interrupciones del puerto paralelo 1, dejando los demás bits del registro de control como estaban:

```
char valor_anterior = inb(0x37A);
outb(0x37A, 0x10 | valor_anterior);
```


4.2 Modelos de drivers en MaRTE OS

Existen dos modelos fundamentales de integración de los drivers con el sistema operativo y el resto de la aplicación:

- **El dispositivo se *mapea como un fichero* (p.e. `/dev/lpt0`)**
  - **se accede a él mediante las funciones de acceso a ficheros (`open()`, `write()`, `read()`, ...) y la función `ioctl()`**
    - **el driver implementa esas funciones**
  - **estrategia habitual en sistemas tipo UNIX**
- **El driver se implementa como una *librería de funciones* específicas:**
  - **`lee_canal_AD()`, `configura_puerto_serie()`, ...**

MaRTE OS permite ambas estrategias

Ventajas e inconvenientes de los modelos

- Dispositivo mapeado como un fichero:
  - ✓ Interfaz estándar de llamadas al sistema (`open()`, `write()`, `read()`, `close()`, `ioctl()`)
  - ✓ Facilita (hasta cierto punto) el intercambio de un dispositivo por otro sin modificar la aplicación
  - ✗ Poca flexibilidad en las llamadas, se acaba abusando de `ioctl()`
- Librería de funciones específicas
  - ✓ Interfaz flexible y fácil de utilizar
  - ✗ No estándar

En este curso nos centraremos en el modelo basado en *dispositivos mapeados como ficheros*

4.3 Drivers en MaRTE OS: Aspectos generales

Tipos de dispositivos

- sólo existen los dispositivos de caracteres

Identificación de drivers

- Números mayores y menores con un significado similar al que tienen en Linux

Instalación de los drivers

- Instalación estática a través de la modificación de la tabla de dispositivos
- Requiere la recompilación del kernel

Los drivers se pueden escribir tanto en Ada como en C

- si se escriben en C necesitan un envoltorio Ada

Los drivers pueden usar todas las funciones POSIX

- threads, mutexes, variables condicionales, temporizadores, semáforos, ...
- excepto en las rutinas de servicio de interrupción, donde no se pueden usar operaciones bloqueantes
  - esto incluye escribir en otros drivers (p.e., no usar la entrada/salida estándar)
  - alternativamente existe una operación `printc` para escribir en consola directamente

El código de cada driver debe ir localizado en un subdirectorio de

- `marte/x86_arch/drivers/`

Puntos de entrada de los drivers

Puntos de entrada de los drivers

- ***create/remove***
  funciones de instalación/desinstalación del driver, externas al sistema de archivos
  - **Linux: suplen las operaciones de instalación y desinstalación de módulos**
- ***open/close y read/write***
  funciones de entrada/salida básicas de POSIX
- ***ioctl***
  función que permite transmitir un comando al dispositivo

Ejemplo: puntos de entrada del driver `my_drv`

```
int my_drv_create (void);
int my_drv_remove (void);

int my_drv_open (int file_descriptor,
                 int file_access_mode);
int my_drv_close (int file_descriptor);

ssize_t my_drv_read (int file_descriptor,
                     void *buffer,
                     size_t bytes);
ssize_t my_drv_write (int file_descriptor,
                      void *buffer,
                      size_t bytes);

int my_drv_ioctl (int file_descriptor,
                  int request,
                  void *argp);
```


4.4 Drivers y sistema de ficheros en MaRTE OS

El sistema de ficheros en MaRTE OS utiliza tres tablas:

- ***Tabla de drivers*** **(editada por el usuario):**
  - **Vincula cada driver (número mayor) con las funciones que constituyen sus puntos de entrada**
- ***Tabla de ficheros de dispositivo*** **(editada por el usuario):**
  - **Define los ficheros de dispositivo existentes (sus nombres)**
  - **Vincula cada fichero de dispositivo con el número mayor de su driver**
  - **Asocia el número menor de los ficheros de dispositivo**
- ***Tabla de descriptores de fichero*** **(gestionada por MaRTE OS):**
  - **Mantiene los descriptores de fichero correspondientes a los dispositivos abiertos en cada momento por la aplicación**

Tabla de drivers

Definición de los drivers:

- **Vincula cada driver (número mayor) con las funciones que constituyen sus puntos de entrada**

Major	Create Create_Procedure_Ac	Remove Remove_Procedure_Ac	Open Open_Procedure_Ac	Close Close_Procedure_Ac	Read Read_Procedure_Ac	Write Write_Procedure_Ac	Ioctl Ioctl_Procedure_Ac
1	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
5	my_drv_create	my_drv_remove	my_drv_open	my_drv_close	my_drv_read	my_drv_write	my_drv_ioctl
Configuration_Parameters.Devices_Mx	null	null	null	null	null	null	null

- **Linux: es equivalente al uso de `register_chrdev_region`, `cdev_alloc` y `cdev_add`**

Tabla de ficheros de dispositivos

Define los ficheros de dispositivo existentes

Device_File_Number	File_Name Path	Major_Number Major	Minor_Number Minor	Device_Used Boolean
1	"/dev/stdin"	1	0	True
2	"/dev/stdout"	2	0	True
3	"/dev/stdin"	3	0	True
4	"/dev/my_drv1"	5	0	True
5	"/dev/my_drv2"	5	1	True
Configuration_parameters.Device_Files_Mx	-	-	-	False

- **Linux: cada entrada es equivalente a un `mknod`**

Tabla de descriptores de ficheros

Mantiene los *descriptores de fichero* correspondientes a los dispositivos abiertos en cada momento por la aplicación

- ***gestionada automáticamente por el SO***

File descriptor	File_Open_Status File_Access_Mode	Device_File_Assigned Device_File_Number	Specific_Data Integer	Fd_Used Boolean
...	...	...	...	True
3	Read_Write	4	0	True
Configuration_parameters.Open_Files_Mx-1	?	?	?	False

Se crea una entrada en cada open

- **la llamada retorna el índice de la fila de la tabla utilizada**

```
int fd = open("/dev/my_drv1", O_RDWR);
```

3

4.5 Creación e instalación de un driver C

```
Puede usarse como plantilla el driver existente en:
  marte/x86_arch/drivers/demo_driver_c/
```

1. Crear un directorio en `marte/x86_arch/drivers/`
2. Escribir uno o más ficheros con las funciones del driver
3. Escribir un fichero Ada de interfaz del driver
   - puede usarse como plantilla `demo_driver_c_import.ads`
4. Crear un `Makefile` en el directorio del driver
5. Modificar las tablas de drivers y de ficheros de dispositivos
6. Recompilar MaRTE OS:
   - `mkmarte`: la primera vez (después de cambiar las tablas)
   - `mkdrivers`: tras cada cambio en el código del driver
     - ejecuta `make` en el directorio del driver y copia los `"*.o"`

Modificación de las tablas de drivers y de ficheros de dispositivos

El fichero `marte-kernel-devices_table.ads` (directorio `marte/x86_arch/arch_dependent_files/`) define las tablas:

- `The_Driver_Table`: *Tabla de drivers*
  - Vincula cada driver (número mayor) con las funciones que constituyen sus puntos de entrada
- `The_Device_Files_Table`: *Tabla de ficheros de dispositivo*
  - registra los ficheros de dispositivo y asocia el driver que los controla

Para incluir un nuevo driver:

- Añadir una entrada a `The_Driver_Table`, eligiendo un número mayor no usado
- Añadir uno o varios ficheros de dispositivos (uno por número menor) a `The_Device_Files_Table`
  - el nombre es un string cualquiera (se puede usar el prefijo `"/dev"` por analogía con Linux)
  - elegir como número mayor el asociado al driver
  - el número menor es cualquiera; se usará para identificar las (posibles) distintas instancias de un dispositivo

Fichero marte-kernel-devices_table.ads

```
--  Typical standard devices(std input, output, error)
with Keyboard_Functions;              -- standard input
with Text_And_Serial_Console_Import;  -- std output, error

--  User's drivers "withs" (add with{your_driver}')
with My_Drv_C_Import;

--  MaRTE OS "withs" (Do not edit)
with Marte.Kernel.File_System_Data_Types;
use Marte.Kernel.File_System_Data_Types;

package Marte.Kernel.Devices_Table is

  pragma Elaborate_Body;
```

Añadido por el usuario


```
   The_Driver_Table :
     Kernel.File_System_Data_Types.Driver_Table_Type :=
     (1 => (Name   => "Keyboard        ",
            Create => Keyboard_Functions.Create'Access,
            Remove => null,
            Open   => null,
            Close  => null,
            Read   => Keyboard_Functions.Read'Access,
            Write  => null,
            Ioctl  => Keyboard_Functions.Ioctl'Access,
            Delete => null,
            Lseek  => null),
      2 => (Name   => "Text/Serial      ",
            Create => Text_And_Serial_console_Import.Create_Ac,
            Remove => Text_And_Serial_console_Import.Remove_Ac,
            Open   => Text_And_Serial_console_Import.Open_Ac,
            Close  => Text_And_Serial_console_Import.Close_Ac,
            Read   => null,
            Write  => Text_And_Serial_console_Import.Write_Ac,
            Ioctl  => Text_And_Serial_console_Import.Ioctl_Ac,
            Delete => null,
            Lseek  => null),
      3 => (Name   => "Text/Serial      ",
            Create => Text_And_Serial_console_Import.Create_Ac,
```



```
            Remove => Text_And_Serial_console_Import.Remove_Ac,
            Open   => Text_And_Serial_console_Import.Open_Ac,
            Close  => Text_And_Serial_console_Import.Close_Ac,
            Read   => null,
            Write  => Text_And_Serial_console_Import.Write_Error_Ac,
            Ioctl  => Text_And_Serial_console_Import.Ioctl_Ac,
            Delete => null,
            Lseek  => null),

      7 => (Name   => "My Driver        ",
            Create => My_Drv_C_Import.Create_Ac,
            Remove => My_Drv_C_Import.Remove_Ac,
            Open   => My_Drv_C_Import.Open_Ac,
            Close  => My_Drv_C_Import.Close_Ac,
            Read   => My_Drv_C_Import.Read_Ac,
            Write  => My_Drv_C_Import.Write_Ac,
            Ioctl  => My_Drv_C_Import.Ioctl_Ac,
            Delete => null,
            Lseek  => null),

      others => ("                ", null, null,
                 null, null, null, null, null, null, null)
      );
```

Driver añadido por el usuario:
- Nombre:"My Driver"
- Número mayor: 7

Fichero marte-kernel-devices_table.ads (cont.)

```
The_Device_Files_Table :
Kernel.File_System_Data_Types.Device_Files_Table_Type :=
(
   --    File path           Major Minor Used   Type     Del   Count
 1  => ("/dev/stdin         ", 1,    0,  True,  Device, False, 0),
 2  => ("/dev/stdout        ", 2,    0,  True,  Device, False, 0),
 3  => ("/dev/stderr        ", 3,    0,  True,  Device, False, 0),
 12 => ("/dev/my_drv        ", 7,    0,  True,  Device, False, 0),
 others =>
       ("                   ", 1,    0,  False, Device, False,0)
 );

end Marte.Kernel.Devices_Table;
```

Fichero de dispositivo añadido por el usuario:
- **Nombre:"/dev/my_drv"**
- **Driver asociado: 7**
- **Número menor: 0**

4.6 Operaciones disponibles para drivers

Operaciones de <drivers/drivers_marte.h>

```
/* retorna el número mayor del driver asociado al
 * descriptor de fichero, o -1 si hay error */
int get_major (int filedes);

/* retorna el número menor del driver asociado al
 * descriptor de fichero, o -1 si hay error  */
int get_minor (int filedes);

/* Obtiene el dato específico del dispositivo */
int driver_getspecific (int filedes, int *data);

/* Cambia el dato específico del dispositivo */
int driver_setspecific (int filedes, int data);

/* Obtiene el modo de acceso */
int get_fd_access_mode (int filedes);
```


Acceso a la memoria

MaRTE OS maneja un *espacio de direcciones único*

- **no diferencia la memoria del kernel de la de usuario**
- **en los puntos de entrada no es necesario el uso de funciones especiales para el intercambio de datos entre el driver y la aplicación**

Memoria dinámica

- **Funciones estándar C (en <stdlib.h>)**
- **para reservar memoria**

```
void *malloc (size_t size)
```

- **para liberar memoria**

```
void free (void *buff);
```

Acceso al hardware

El acceso a los registros de un dispositivo se consigue con las funciones de `<sys/pio.h>`

- **funciones de entrada y salida de bytes**

```
inb(port)
inb_p(port)
outb(port, val)
outb_p(port, val)
```

- **también hay funciones para 16 y 32 bits**

Hay facilidades para uso del bus PCI en `<sys/pci.h>`

4.7 Ejemplo de un driver C: demo_driver_c.c

```
#include <stdio.h>
#include <drivers/drivers_marte.h>

#define BUFFER_SIZE 30
char demo_buffer [BUFFER_SIZE] = "";

int demo_c_create ()
{
  return 0;
}


int demo_c_remove ()
{
  return 0;
}
```


Ejemplo de un driver C: demo_driver_c.c (cont.)

```
int demo_c_open (int file_descriptor, int file_access_mode)
{
  printf ("Demo C Drv:Open dev file %d (Maj:%d, Min:%d) Mod %d\n",
          file_descriptor,
          get_major(file_descriptor),
          get_minor(file_descriptor),
          file_access_mode);
  return 0;
}


int demo_c_close (int file_descriptor)
{
  printf ("Demo C Drv: Close dev file %d (Major:%d, Minor:%d)\n",
          file_descriptor,
          get_major(file_descriptor),
          get_minor(file_descriptor));
  return 0;
}
```

```
ssize_t demo_c_read (int file_descriptor, void *buffer,
                     size_t bytes)
{
  size_t i, bytes_read = 0;

  if (bytes < BUFFER_SIZE)
    bytes_read = bytes;
  else
    bytes_read = BUFFER_SIZE;

  for (i=0; i<bytes_read; i++)
    ((char *)buffer)[i] = demo_buffer[i];

  printf ("Demo C Diver: read %d bytes\n", bytes_read);
  return bytes_read;
}
```



```
ssize_t demo_c_write (int file_descriptor, void *buffer,
                      size_t bytes)
{
  size_t i, bytes_written = 0;

  if (bytes < BUFFER_SIZE)
    bytes_written = bytes;
  else
    bytes_written = BUFFER_SIZE;

  for (i=0; i<bytes_written; i++)
    demo_buffer[i] = ((char *)buffer)[i];

  printf ("Demo C Diver: written %d bytes\n", bytes_written);
  return bytes_written;
}
```



```
int demo_c_ioctl (int file_descriptor, int request, void* argp)
{
  printf ("Demo C Diver: Ioctl. Request:%d, argp:%s\n",
          request, (char *) argp);
  return 0;
}
```

```
with Drivers_MaRTE;  use Drivers_MaRTE;
with System, Ada.Unchecked_Conversion;
package Demo_Driver_C_Import is
   --  Create
   function Create return Int;
   pragma Import (C, Create, "demo_c_create");
   function Address_To_Create_Ac is new Ada.Unchecked_Conversion
       (System.Address, Create_Function_Ac);
   Create_Ac : Create_Function_Ac :=
       Address_To_Create_Ac (Create'Address);

   --  Remove
   function Remove return Int;
   pragma Import (C, Remove, "demo_c_remove");
   function Address_To_Remove_Ac is new Ada.Unchecked_Conversion
       (System.Address, Remove_Function_Ac);
   Remove_Ac : Remove_Function_Ac :=
       Address_To_Remove_Ac (Remove'Address);
```



```
   --  Open
   function Open (Fd   : in File_Descriptor;
                  Mode : in File_Access_Mode) return Int;
   pragma Import (C, Open, "demo_c_open");
   function Address_To_Open_Ac is new Ada.Unchecked_Conversion
       (System.Address, Open_Function_Ac);
   Open_Ac : Open_Function_Ac :=
       Address_To_Open_Ac (Open'Address);


   --  Close
   function Close (Fd : in File_Descriptor) return Int;
   pragma Import (C, Close, "demo_c_close");
   function Address_To_Close_Ac is new Ada.Unchecked_Conversion
       (System.Address, Close_Function_Ac);
   Close_Ac : Close_Function_Ac :=
       Address_To_Close_Ac (Close'Address);
```



```
   --  Read
   function Read (Fd         : in File_Descriptor;
                  Buffer_Ptr : in Buffer_Ac;
                  Bytes      : in Buffer_Length) return Int;
   pragma Import (C, Read, "demo_c_read");
   function Address_To_Read_Ac is new Ada.Unchecked_Conversion
       (System.Address, Read_Function_Ac);
   Read_Ac : Read_Function_Ac :=
       Address_To_Read_Ac (Read'Address);

   --  Write
   function Write (Fd         : in File_Descriptor;
                   Buffer_Ptr : in Buffer_Ac;
                   Bytes      : in Buffer_Length) return Int;
   pragma Import (C, Write, "demo_c_write");
   function Address_To_Write_Ac is new Ada.Unchecked_Conversion
       (System.Address, Write_Function_Ac);
   Write_Ac : Write_Function_Ac :=
       Address_To_Write_Ac (Write'Address);
```

```
   --  Ioctl

   function Ioctl (Fd             : in File_Descriptor;
                   Request        : in Ioctl_Option_Value;
                   Ioctl_Data_Ptr : in Buffer_Ac) return Int;
   pragma Import (C, Ioctl, "demo_c_ioctl");
   function Address_To_Ioctl_Ac is new Ada.Unchecked_Conversion
       (System.Address, Ioctl_Function_Ac);
   Ioctl_Ac : Ioctl_Function_Ac :=
       Address_To_Ioctl_Ac (Ioctl'Address);

end Demo_Driver_C_Import;
```



```
MGCC = ../../../utils/mgcc

DEFAULT: demo_driver_c.o

demo_driver_c.o: demo_driver_c.c Makefile
	$(MGCC) -c $(CFLAGS) demo_driver_c.c
```