Periféricos Interfaces y Buses

I. Arquitectura de E/S

II. Programación de E/S

Aspectos básicos de la programación de E/S. Arquitectura y programación de la E/S en el sistema operativo. Manejadores de dispositivos (drivers) y su programación (interrupciones).

III. Interfaces de E/S de datos

IV. Dispositivos de E/S de datos

V. Buses

VI. Controladores e interfaces de dispositivos de almacenamiento

VII. Sistemas de almacenamiento

II. Programación de E/S

Bloque III

- **Drivers en MaRTE OS**

Nociones básicas de Marte OS

Se pueden hacer aplicaciones en Ada y C

En la instalación se elige un directorio `export` donde se va a colocar el ejecutable del target para que sea cargado, por ejemplo por *etherboot*:

- en el servidor en `/etc/dhcpd.conf` se encuentran la MAC e IP del host y del target, el nombre del fichero y su path

Suponemos que el directorio de instalación es `/marte`

- aquí se encuentran los fuentes y algunas utilidades de compilación

En `/marte/utils` se encuentra el fichero `globals.pl` en el que hay parámetros de configuración creados al instalar MaRTE

Nociones básicas de Marte OS (cont.)

En `/marte/utils` se encuentran algunas utilidades de compilación:

- `mkdrivers`: compila sólo los drivers
- `mkmarte`: compila el Kernel de MaRTE salvo el runtime
- `mkrtsmarteuc`: compila el runtime de MaRTE para la plataforma seleccionada
- `mgcc`: compila el programa de aplicación C y crea el ejecutable
- `mgnatmake`: compila el programa de aplicación en Ada y crea el ejecutable

Los ejecutables se tienen que llevar al directorio `export` en el que el target espera encontrar el fichero para cargarlo

Drivers en MaRTE OS

Tipos de drivers

- sólo existen los drivers de caracteres

Identificación de ficheros de dispositivo

- números mayores y números menores asignados en una tabla de dispositivos (cualquier variación en esta tabla exige la recompilación del kernel)

Instalación de los drivers

- instalación estática a través de la correspondiente modificación de la tabla de dispositivos

Los drivers se pueden escribir tanto en Ada como en C

- si se escriben en C necesitan un envoltorio Ada

Drivers en MaRTE OS (cont.)

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

Cada driver debe ir localizado en un subdirectorio de

- `marte/x86_arch/drivers/`

Identificación de drivers

La identificación de los drivers se hace con los números mayores y menores con un significado similar al que hemos visto para Linux

La implementación se realiza mediante tres tablas:

- Tabla de descriptores de fichero: permite el acceso al dispositivo desde las aplicaciones
- Tabla de ficheros de dispositivo: contiene la información de dispositivo instalado (asocia el número mayor y menor)
- Tabla de drivers: cada driver es identificado por el número mayor y contiene toda la funcionalidad del driver

Las dos últimas permiten la instalación de driver y dispositivos

Tabla de descriptores de ficheros

Esta tabla es gestionada automáticamente por el sistema

Table 1: The_FD_Table

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

Se crea una entrada en cada `open`

Tabla de ficheros de dispositivos

Esta tabla es configurada por el usuario

Table 2: The_Device_Files_Table

Device_File_ Number	File_Name Path	Major_ Number Major	Minor_ Number Minor	Device_Used Boolean
1	?	1	1	False
4	test_driver1	5	1	True
Configuration_parameters. Device_Files_Mx	?	1	1	False

Linux: cada entrada es equivalente a un `mknod`

Tabla de drivers

Esta tabla es configurada por el usuario

Table 3: The_Driver_Table

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	null	null	null	null	null	null	null
5	test_ create	test_ remove	test_ open	test_ close	test_ read	test_ write	test_ 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`

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 desintalación de módulos
- *open/close y read/write*
  funciones de entrada/salida básicas de POSIX gestionadas internamente por el sistema de archivos
- *ioctl*
  función que permite transmitir una orden al dispositivo

API de los puntos de entrada (C)

```
int my_driver_create (void);

int my_driver_remove (void);

int my_driver_open (int file_descriptor, int file_access_mode);

int my_driver_close (int file_descriptor);

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

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

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

Creación de un driver C

Incluso para drivers C hay que escribir una especificación Ada

- puede usarse como plantilla el fichero `drivers/demo_driver_c/demo_driver_c_import.ads`

Además, hay que crear un `GNUmakefile`

- `mkdrivers` ejecutará `make` en el directorio del driver
- después `mkdrivers` copiará en `lib/libdrivers.o` los objetos de los directorios de los drivers

Ficheros implicados en la instalación de un driver

`/marte/x86_arch/arch_dependent_files/marte-kernel-devices_table.ads`

- registra todas las funciones de los drivers
- es un fichero único para todos los drivers

`/marte/x86_arch/arch_dependent_files/marte-configuration_parameters.ads`

- parámetros de configuración del kernel
- se pueden configurar entre otras cosas algunos parámetros correspondientes a los drivers, como los rangos de números mayores y menores

`/marte/x86_arch/drivers/my_driver/*`

- ficheros con el código de mi driver en el subdirectorio `my_driver`

Instalación de un driver

El fichero `marte-kernel-devices_table.ads` tiene dos tablas:

- `The_Driver_Table`: drivers incluidos en el sistema
  - esta tabla asocia números mayores a los drivers y contiene los puntos de entrada
- `The_Device_Files_Table`: registra los dispositivos y asocia el driver que los controla

Instalación de un driver (cont.)

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 unidades de manera definida por el driver

Ejemplo de fichero de 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 Demo_Driver_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;
```


marte-kernel-devices_table.ads (cont.)

```
   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,
```

marte-kernel-devices_table.ads (cont.)

```
        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),
```


marte-kernel-devices_table.ads (cont.)

```
7 => (Name   => "Demo Driver C   ",
        Create => Demo_Driver_C_Import.Create_Ac,
        Remove => Demo_Driver_C_Import.Remove_Ac,
        Open   => Demo_Driver_C_Import.Open_Ac,
        Close  => Demo_Driver_C_Import.Close_Ac,
        Read   => Demo_Driver_C_Import.Read_Ac,
        Write  => Demo_Driver_C_Import.Write_Ac,
        Ioctl  => Demo_Driver_C_Import.Ioctl_Ac,
        Delete => null,
        Lseek  => null),
others => ("                  ", null, null,
           null, null, null, null, null, null, null)
);
```

marte-kernel-devices_table.ads (cont.)

```
   function To_Path (Str : String) return Path;

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

end Marte.Kernel.Devices_Table;
```


Resumen del proceso de instalación de un driver

Realizar el código del driver:

- en C: además se necesita un fichero Ada para importar las funciones C y compilar las funciones C con `mkdrivers`
- en Ada: se hace uso del paquete `Drivers_Marte` en `/marte/misc` que contiene la interfaz de las funciones y tipos

Modificar `marte-kernel-devices_table.ads` para *"instalar"* el driver

Realizar el código de la aplicación

Compilar el driver con `mkkernel`, `mkrtskerneluc`, o `mkdrivers`

Compilar y enlazar la aplicación: `mgcc` (C) o `mgnatmake` (Ada)

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 que es 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
- las funciones están en <stdlib.h>
- para reservar memoria

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

- para liberar memoria

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

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

```
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;
}
```

Ejemplo de un driver C: demo_driver_c.c

```
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;
}
```


Ejemplo de un driver C: demo_driver_c.c

```
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;
}
```

Ejemplo de un driver C: demo_driver_c.c

```
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;
}
```


Ejemplo de un driver C: demo_driver_c_import.ads

```
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);
```

Ejemplo de un driver C: demo_driver_c_import.ads

```
--  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);
```


Ejemplo de un driver C: demo_driver_c_import.ads

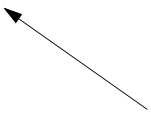
```
--  Read
function Read (Fd         : in File_Descriptor;
               Buffer_Ptr : in Buffer_Ac;
               Bytes      : in Buffer_Length) return Int;
pragma Import (C, Read, "demo_c_read");
function Address_To_Read_Ac is new Ada.Unchecked_Conversion
    (System.Address, Read_Function_Ac);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

Ejemplo de un driver C: demo_driver_c_import.ads

```
   --  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;
```


Ejemplo de un driver C: GNUmakefile

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

DEFAULT: demo_driver_c.o

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



Programa de prueba

Dado que MaRTE OS no tiene procesos, no podemos poner dos programas a ejecutar (p.e., uno de lectura y otro de escritura)

- la concurrencia se debe hacer con threads (o tareas en el caso de Ada)

El programa de prueba hace lo siguiente:

- abre el dispositivo `/dev/demo_c`
- realiza un lazo hasta una condición de parada
  - escribe en el dispositivo un string dado por el usuario
  - lee seguidamente el buffer del dispositivo
- cuando sale del lazo cierra el dispositivo

Sincronización

Para el acceso a secciones críticas

- mútexes y semáforos
- con un manejador de interrupción se deben inhibir interrupciones

Para la sincronización de espera

- semáforos y variables condicionales
- un manejador de interrupción
  - puede despertar a un thread en espera en un semáforo
  - dispone de una función especial para despertar a un thread

Si el driver se escribe en Ada la sincronización se puede hacer con los objetos protegidos

Sincronización (cont.)

Semáforos definidos en <semaphore.h>

```
int sem_init (sem_t *sem, int pshared, unsigned int value);

int sem_destroy (sem_t *sem);

int sem_wait (sem_t *sem);

int sem_trywait (sem_t *sem);

int sem_timedwait (sem_t *sem, const struct timespec *abs_timeout);

int sem_post (sem_t *sem);

int sem_getvalue (sem_t *sem, unsigned int *value);
```

Mutexes, variables condicionales, threads y funciones de planificación definidos en <pthread.h>

Sincronización (cont.)

Mutexes:

```
int pthread_mutexattr_init (pthread_mutexattr_t *attr);

int pthread_mutexattr_destroy (pthread_mutexattr_t *attr);

int pthread_mutex_init (pthread_mutex_t *m,
     const pthread_mutexattr_t *attr);

int pthread_mutex_destroy (pthread_mutex_t *m);

int pthread_mutex_lock (pthread_mutex_t *m);

int pthread_mutex_trylock (pthread_mutex_t *m);

int pthread_mutex_unlock (pthread_mutex_t *m);

int pthread_mutex_timedlock (pthread_mutex_t *m,
          const struct timespec *abs_timeout);
```

Temporización

MaRTE OS dispone de los temporizadores de POSIX que están pensados para enviar una señal a un thread

También dispone de los relojes de POSIX y de las funciones de dormir de alta resolución

Esto permite la creación de threads periódicos también en los drivers; en `<time.h>`

```
struct timespec {
  time_t tv_sec;
  int tv_nsec;
};

int nanosleep (const struct timespec *rqtp, struct timespec *rmtp);
int clock_nanosleep (clockid_t clock_id, int flags,
        const struct timespec *rqtp, struct timespec *rmtp);
```


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>`

Interrupciones

Hay facilidades para interrupciones hardware en <intr.h>

- instalación y desintalación de manejadores de interrupción

```
int posix_intr_associate (intr_t intr,
     int (*intr_handler) (void * area, intr_t intr),
     volatile void * area,
     size_t areasize);

int posix_intr_disassociate (intr_t intr,
        int (*intr_handler) (void * area, intr_t intr));
```

- bloqueo y desbloqueo de interrupciones

```
int posix_intr_lock (intr_t intr);
int posix_intr_unlock (intr_t intr);
```

- espera temporizada a un manejador

```
int posix_intr_timedwait (int flags,
     const struct timespec *abs_timeout,
     intr_t *intr,
     int (**intr_handler) (void * area, intr_t intr) );
```


Threads del kernel

No existen threads especiales en MaRTE OS

Se puede usar un thread "normal" para esperar a un manejador de interrupción

- éste después puede activar a cualquier otro que estuviera esperando en un punto de entrada del driver
- la espera al manejador se hace
  - con `posix_intr_timedwait`
  - con un semáforo

Threads de MaRTE OS

Creación y destrucción de threads

```
int pthread_create (pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start_routine)(void *), void *arg);

void pthread_exit (void *value_ptr);
```

Creación de los atributos

```
int pthread_attr_init (pthread_attr_t *attr);
int pthread_attr_destroy (pthread_attr_t *attr);
```


Threads de MaRTE OS (cont.)

Atributos generales

```
int pthread_attr_setstacksize(pthread_attr_t *attr,
            size_t stacksize);

int pthread_attr_getstacksize(const pthread_attr_t *attr,
            size_t *stacksize);

int pthread_attr_setstackaddr(pthread_attr_t *attr,
            void * stackaddr);

int pthread_attr_getstackaddr(pthread_attr_t *attr,
            void ** stackaddr);

int pthread_attr_setdetachstate(pthread_attr_t *attr,
            int detachstate);

int pthread_attr_getdetachstate(const pthread_attr_t *attr,
            int *detachstate);
```

Threads de MaRTE OS (cont.)

Atributos de planificación

```
int pthread_attr_setinheritsched (pthread_attr_t *attr,
                  int inheritsched);
int pthread_attr_getinheritsched (pthread_attr_t *attr,
                  int *inheritsched);

int pthread_attr_setschedpolicy (pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy (pthread_attr_t *attr, int policy);

int pthread_attr_setschedparam (pthread_attr_t *attr,
                const struct sched_param *param);
int pthread_attr_getschedparam (const pthread_attr_t *attr,
                struct sched_param *param);

int pthread_setschedparam (pthread_t thread, int policy,
                const struct sched_param *param);
int pthread_getschedparam (pthread_t thread, int *policy,
                struct sched_param *param);

int pthread_setschedprio (pthread_t thread, int prio);
```


Bibliografía

[1] Jonathan Corbet, Greg Kroah-Hartman, Alessandro Rubini, "Linux Device Drivers", 3rd Ed., O'Reilly, 2005.

[2] P. J. Salzman, M. Burian, O. Pomerantz, "The Linux Kernel Module Programming Guide", Ver. 2.6.4, 18-5-2007: http://tldp.org/LDP/lkmpg/2.6/html/

[3] MaRTE OS: http://marte.unican.es/

[4] Francisco Guerreira, "Entorno para la instalación y utilización de manejadores de dispositivos en MaRTE OS", Proyecto fin de carrera, http://marte.unican.es/projects/driversframework/memoria_drivers.pdf