

Entrada/Salida

Alexis Tcach

DC - FCEyN - UBA

Sistemas Operativos, 1c-2016

Interfaz de E/S

Nos informan que un SO provee la siguiente API para operar con un dispositivo de E/S genérico.

<code>int open(int device_id)</code>	Abre el dispositivo
<code>int close(int device_id)</code>	Cierra el dispositivo
<code>int read(int device_id, int* data)</code>	Lee el dispositivo <code>device_id</code>
<code>int write(int device_id, int* data)</code>	Escribe el valor en el dispositivo <code>device_id</code>
<code>int seek(int device_id, int offset)</code>	Avanza/retrocede el <i>seek pointer</i>

Todas las operaciones retornan la constante `IO_OK` si fueron exitosas o la constante `IO_ERROR` si ocurrió algún error.

Programación de un Driver

Para ser cargado como un *driver* válido por el sistema operativo, éste debe implementar los siguientes procedimientos:

Función	Invocación
int driver_init()	Durante la carga del <i>driver</i>
int driver_open()	Al solicitarse un open
int driver_close()	Al solicitarse un close
int driver_read(int *data)	Al solicitarse un read
int driver_write(int *data)	Al solicitarse un write
int driver_seek(int offset)	Al solicitarse un seek
int driver_remove()	Durante la descarga del <i>driver</i>

Funciones del *kernel* para *drivers*

Para la programación de un *driver*, se dispone de las siguientes funciones:

<code>void OUT(int IO_address, int data)</code>	Escribe data en el registro de E/S
<code>int IN(int IO_address)</code>	Retorna el valor almacenado en el registro de E/S.
<code>void * kalloc(int size)</code>	Pide memoria en espacio de kernel
<code>int kfree(void *)</code>	Libera memoria pedida con <i>kalloc</i>
<code>copy_from_user(void * from, void * to, int size)</code>	Copia de la memoria en espacio de usuario a la memoria en espacio de <i>kernel</i> .
<code>copy_to_user(void * from, void * to, int size)</code>	Copia de la memoria en espacio de <i>kernel</i> a la memoria en espacio de usuario.

Un *driver* para un HDD

Se tiene un HDD conectado a una computadora. Se sabe que el HDD posee 5 registros:

HDD_STATUS	[HDD_STAT_RDY HDD_STAT_BSY]
HDD_SECTOR	Sector a ser leído o escrito
HDD_OPERATION	[HDD_OP_READ HDD_OP_WRIT]
HDD_DATA	Contiene la información a ser leída o escrita
HDD_COUNT	Cantidad de sectores a ser leídos o escritos

Un *driver* para un HDD

Se tiene un HDD conectado a una computadora. Se sabe que el HDD posee 5 registros:

HDD_STATUS	[HDD_STAT_RDY HDD_STAT_BSY]
HDD_SECTOR	Sector a ser leído o escrito
HDD_OPERATION	[HDD_OP_READ HDD_OP_WRITE]
HDD_DATA	Contiene la información a ser leída o escrita
HDD_COUNT	Cantidad de sectores a ser leídos o escritos

Escribir un driver para este dispositivo para un sistema **monoproceso**.

Pasos para hacer una lectura

- ❶ Esperar a que el disco esté en estado `HDD_STAT_RDY`
- ❷ Indicar el sector inicial a ser leído en `HDD_SECTOR`
- ❸ Indicar la cantidad de sectores **consecutivos** a ser leídos en `HDD_COUNT`
- ❹ Poner el valor `HDD_OP_READ` en `HDD_OPERATION`
- ❺ Esperar a que el disco pase al estado `HDD_STAT_BSY`
- ❻ Esperar a que el disco vuelva al estado `HDD_STAT_RDY`
- ❼ A partir de este momento se pueden hacer $\text{HDD_COUNT} * (\text{SECTOR_SIZE}/4)$ lecturas de `HDD_DATA`.

En el pizarrón

En el pizarrón

Tarea: Implementar el *write*.

Mejorando la *performance*

Para poder manejar interrupciones, necesitamos dos funciones (provistas por el *kernel*) adicionales:

- `int request_irq(int irq, void* handler)`
Permite asociar el procedimiento handler a la interrupción irq. Retorna `IRQ_OK` ó `IRQ_ERROR` si no se pudo asociar.
- `int free_irq(int irq)`
Libera la interrupción irq del procedimiento asociado.

En el pizarrón

En el pizarrón

Tarea: Re-implementar el *write*.

Nos piden implementar *por software* RAID 1 mediante el desarrollo de un *driver* para nuestro SO monoproceso.

Este *driver* debe manejar dos discos cuyos identificadores son HDD1_DEV_ID y HDD2_DEV_ID.

En el pizarrón

En el pizarrón

Tarea: Implementar el *write*.