Entrada/Salida

Ignacio Vissani

DC - FCEyN - UBA

Sistemas Operativos, 2c-2015

Interfaz de E/S

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

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

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 *driver*
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 *driver*

Funciones del *kernel* para *drivers*

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

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

1. Esperar a que el disco esté en estado `HDD_STAT_RDY`
2. Indicar el sector inicial a ser leído en `HDD_SECTOR`
3. Indicar la cantidad de sectores **consecutivos** a ser leídos en `HDD_COUNT`
4. Poner el valor `HDD_OP_READ` en `HDD_OPERATION`
5. Esperar a que el disco pase al estado `HDD_STAT_BSY`
6. Esperar a que el disco vuelva al estado `HDD_STAT_RDY`
7. A partir de este momento se pueden hacer `HDD_COUNT * (SECTOR_SIZE/4)` lecturas de `HDD_DATA`.

Implementación de *read*

En el pizarrón

En el pizarrón

Tarea: Implementar el *write*.

Mejorando la *performance*

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.

Re-implementación de *read* usando interrupciones

En el pizarrón

En el pizarrón

Tarea: Re-implementar el *write*.

RAID por *software*

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*.