EAFITOS
Una herramienta para aprender sistemas operativos
(versión 2.0)

Juan David Ibáñez Palomar

5 de abril de 1999

AVISO LEGAL

a mi abuelo Victoriano

Índice General

Índice de Figuras

Índice de Tablas

Prefacio

Este documento, junto con el software que le acompaña, es una herramienta creada para el aprendizaje de Sistemas Operativos, su diseño e implementación. Su nombre es EAFITOS cuyo significado es "EAFIT Operating System", EAFIT es la universidad donde nació, situada en Medellín (Colombia).

Ya existen otros Sistemas Operativos creados con los mismos fines. Los cuales pueden ser divididos en dos grupos según funcionen directamente sobre una máquina real (ejemplo, Minix) o sobre una máquina imaginaria que se simula sobre otro SO (ejemplo, Nachos).

Al desarrollar Eafitos se decidió hacerlo sobre una máquina imaginaria ya que eso reduce mucho los costes de implementación y simplifica el producto final, el cual será más fácil de comprender y modificar por parte de los estudiantes.

Historia

La primera versión

Los textos de esta sección son fragmentos extraídos de la documentación de la primera versión de Eafitos.

Origen

El Sistema Operativo EAFITOS fue diseñado e implementado con el fin de permitir que los estudiantes del curso de pregrado ST075, de la Universidad de EAFIT (Medellín, Colombia) tuvieran una herramienta (simple pero completa) para comprender, visualizar, evaluar y experimentar los conceptos abstractos que se enseñan en un curso básico de S.O. La primera versión de este producto fue desarrollada por aproximadamente 10 grupos de 3 estudiantes en promedio durante el primer semestre de 1995 y ocupa menos de un disquete de 1.44 Megas incluyendo programas fuentes y ejecutables. Posteriormente, en los cursos siguientes se han ido mejorando algunas de sus funciones básicas como manejo de ventanas simples, procesos concurrentes, semáforos, etc... Pero quizá lo más importante es que se ha ido incrementando poco a poco la calidad del mismo y en consecuencia el número de "bugs".

Objetivos

Los principales objetivos que se tuvieron en cuenta en el diseño del S.O. EAFITOS fueron:

1. Ser tan simple que pueda ser entendido en pocas sesiones de clase con el fin de que el estudiante pueda evaluarlo, modificarlo y/o extenderlo como parte del trabajo normal del curso ST075.
2. Correr en el computador personal de cada estudiante sin obligarlo a hacer particiones de su disco ni ampliar el hardware.
3. Implementar conceptos básicos y fundamentales del S.O. que no tiene DOS-PC como son multiprogramación, planificación de procesos, hilos, mecanismos de sincronización, seguridad, adecuado nivel de protección de archivos, protección de memoria, etc..
4. Mostrar claramente la diferencia entre la especificación de un S.O. (llamadas al sistema) y la implementación de las mismas, procurando que la interfaz se mantenga cuando la implementación cambie.
5. Facilitar a comprender los conceptos básicos de diseño como complejidad, eficiencia, simplicidad, elegancia, robustez, etc.. y las relaciones entre ellos.
6. Ser fácilmente transportable a otras plataformas de S.O.
7. Poder usarse como herramienta en otros cursos de la carrera de Ingeniería de Sistemas como son Teoría de Archivos, Bases de Datos, Arquitectura de Computadores, Telemática, Compiladores, etc..

La versión 2.0

En el verano de 1998 se me concedió una beca del programa de Cooperación Interuniversitaria, que otorga la AECI (Agencia Española para la Cooperación Internacional) a estudiantes españoles para realizar trabajos en universidades Latino-Americanas.

El destino de la beca fue la universidad EAFIT de Medellín (Colombia) y el objetivo de la misma era continuar con el desarrollo del sistema operativo educacional EAFITOS. Mi tutor en Colombia fue el profesor Omar García Palencia. Decidimos reescribir EAFITOS desde cero, para conseguir un software que se adecuara mejor a sus objetivos educativos.

No fue posible terminar el trabajo durante los dos meses que duró mi estancia en Colombia, así que lo terminé en España, aprovechándolo además como proyecto final de carrera (mi director de proyecto fue el profesor Rafael Mayo Gual).

La nueva versión de Eafitos se adecua mejor a sus objetivos de diseño. Por ejemplo, es más portable, funciona en DOS y en GNU/Linux y portarlo a otro sistema no debería implicar la modificación de una sola línea de código fuente;

también es más fácil de comprender y modificar, gracias en parte a un diseño más cuidado y a una documentación de mayor calidad.

Pero además de esto, un objetivo fundamental de la nueva versión de Eafitos era acercarse un poco más a los sistemas operativos reales, ya que la primera versión tenía algunas diferencias graves e innecesarias. Por ejemplo, la nueva versión elimina los registros de tipo cadena del procesador e implementa las llamadas al sistema como tales (no como instrucciones del procesador).

Algunas preguntas y sus respuestas

¿Por qué orientado a objetos?

El programa, de por sí, requiere un enfoque orientado a objetos (para la abstracción de sistemas de ficheros, por ejemplo). Esto se puede implementar mediante punteros con un lenguaje más tradicional, como C, de hecho eso es lo que hacen muchos núcleos de sistema operativo reales, Linux por ejemplo. Pero resulta mucho más natural hacerlo con un lenguaje orientado a objetos.

Además, estos lenguajes suelen ofrecer otras características que hacen más fácil la vida de los estudiantes y desarrolladores. Así, se simplifica el diseño y se incrementa la legibilidad del código.

¿Por qué C++?

La versión original de Eafitos estaba escrita en ANSI C, igual que la mayoría de sistemas operativos reales. Pero C no nos vale porque no es orientado a objetos.

Probablemente otros lenguajes hubieran servido a nuestros propósitos, de hecho, inicialmente se consideró la posibilidad de escribir la nueva versión en Java. Pero pensamos que era mejor utilizar un lenguaje compilado para que a ningún estudiante se le quedase "pequeño" su ordenador.

Existen otros lenguajes compilados que podrían haber servido, podríamos haberlos evaluado con detenimiento pero eso hubiera significado mucho tiempo. Así que elegimos C++, porque sirve perfectamente a nuestros propósitos y porque pasar de C a C++ resulta ser lo más natural.

¿Por qué Curses?

La versión original de Eafitos utilizaba la librería de DOS *conio* para implementar el editor de textos y el depurador. Pero esta librería tan solo está disponible, que yo sepa, en DOS.

Para mejorar la portabilidad del código se decidió utilizar Curses, una líbreria que ha sido portada a un gran número de sistemas distintos. En concreto, se ha utilizado *NCurses* en GNU/Linux y *PDCurses* en DOS.

Acerca de los responsables del proyecto

El profesor Omar García Palencia es quien empezó todo esto. Es profesor de Ingeniería de Sistemas en la Universidad EAFIT de Medellín (Colombia). Puede ser localizado en:

Correo electrónico : *ogarcia@sigma.eafit.edu.co*
Páginas web : *http://sigma.eafit.edu.co/~ogarcia*

El autor de este documento y de la versión 2.0 de Eafitos es Juan David Ibáñez Palomar. Soy Ingeniero en Informática de la Universidad Jaume I de Castellón (España). Puedes encontrarme en:

Correo electrónico : *al004151@alumail.uji.es*
Páginas web : *http://www4.uji.es/~al004151*

Parte I

Usar y programar Eafitos

Capítulo 1

Manual de usuario

1.1 Instalación de Eafitos

Para poder ejecutar Eafitos tan solo necesitas el fichero ejecutable, el cual debes haberlo obtenido junto con este documento.

Pero antes de ejecutar Eafitos debes tener en cuenta una cosa. Eafitos necesita de unos ficheros (los discos duros virtuales) que se alojaran en el directorio "\EAFITOS" o "$HOME/EAFITOS" según utilices DOS o GNU/Linux. Asegurate de tener dicho directorio creado antes de ejecutar Eafitos.

Compilación

Para compilar Eafitos en DOS necesitarás Borland C++ 4.5, es posible que funcione con otras versiones (posteriores a la versión 3.1), pero no puedo garantizarlo ya que no lo he probado.

Para compilarlo en GNU/Linux necesitarás el compilador *egcs* y la librería *NCURSES*. Si dispones de esto, es posible que incluso puedas compilarlo en otras versiones de UNIX.

La compilación en otros entornos será posible siempre y cuando dispongas de alguna líbreria de Curses y de un compilador de C++ que implemente, al menos, la revisión número 3.0 del borrador oficial de ANSI C++.

1.2 Gestión de los discos

La información en Eafitos se almacena en unos discos virtuales que en realidad son ficheros del Sistema Operativo real sobre el que se ejecuta Eafitos.

Cuando ejecutas Eafitos aparece un menú con cuatro opciones. La primera te

permitirá arrancar el intérprete de comandos para empezar a trabajar, pero antes de hacer esto debes tener al menos un disco creado y formateado. Para eso tienes las opciones 2 y 3, primero crea los discos que quieras (hasta un máximo de cinco) con la opción 2 y después les das formato; ahora ya puedes iniciar el intérprete. La última opción, la cuarta, es, como su nombre indica, para terminar la ejecución de Eafitos.

1.3 El intérprete de comandos

Ahora vamos a ver qué es lo que se puede hacer con Eafitos. Para ello estudiaremos su intérprete de comandos.

Primero, has de saber que el intérprete distingue entre mayúsculas y minúsculas (como UNIX y al contrario que DOS), por lo que 'Salir' no es lo mismo que 'salir'.

Ahora veamos la lista de comandos que nos proporciona:

- **salir** Termina el intérprete y regresa al menú inicial.
- **disco** ***numero*** Cambia el disco actual al indicado y el directorio actual pasa a ser el directorio raíz del nuevo disco.
- **creadir** ***nombre*** Crea el directorio *nombre*.
- **cd** ***directorio*** Cambia el directorio actual al indicado.
- **poner** ***fichero*** Pasa el fichero indicado del SO anfitrión a Eafitos.
- **obtener** ***fichero*** Pasa el fichero indicado de Eafitos al SO anfitrión.
- **borrar** ***fichero*** Borra el fichero indicado (también sirve para direcotorios).
- **dir** Lista el contenido del directorio actual.
- **ver** ***fichero*** Muestra el contenido del fichero indicado (como *type* de DOS o *cat* de UNIX).
- **compilar** ***fichero*** Compila el fichero indicado (aquí entra en juego el compilador), el ejecutable que se genera recibe el nombre *fichero.exe*.

Cuando escribes cualquier otra cadena, si corresponde a un fichero ejecutable lo ejecuta y si no da un mensaje de error. Los programas de usuario no admiten parámetros en la línea de comandos. Cualquier parámetro de más en la línea de comandos se ignora.

Capítulo 2

Manual del programador

En este capítulo veremos todo lo que se necesita saber para escribir programas para el sistema operativo Eafitos, y algo más.

La CPU de la máquina virtual sobre la que se ejecuta Eafitos entiende un código máquina muy sencillo, con un conjunto de instrucciones muy reducido que se describirá más adelante. Pero el programador de Eafitos no tiene que tratar directamente con el código máquina, ya que se ha desarrollado un lenguaje ensamblador y un compilador que hacen más fácil la vida del programador.

En las Secciones 2.1 y 2.2 se describe dicho lenguaje y se explica la arquitectura del procesador; en la Sección 2.3 se describen los servicios que nos proporciona el núcleo de Eafitos; finalmente, en la Sección 2.4 se estudia un pequeño programa que servirá de ejemplo.

2.1 Estructura de un programa

Un programa de Eafitos consta de dos partes. Primero una declaración de variables, que es opcional, y a continuación el código.

En la Figura 2.1 vemos dos pequeños ejemplos que nos ayudarán a comprender mejor dicha estructura.

Lo primero que se observa en ambos ejemplos es una línea empezada por un punto y coma. Se trata de un comentario. Los comentarios empiezan por punto y coma y acaban en el final de la línea. Se pueden poner en cualquier lugar del programa.

Lo que diferencia a ambos ejemplos es que el segundo no tiene una zona de declaración de variables. La zona de declaración de variables, si existe, debe estar siempre al inicio del programa y empezar por la palabra DATOS (o DaTos, el compilador no es sensible a mayúsculas/minúsculas). En ella existen una lista de variables, primero el nombre de la variable y a continuación su valor inicial. Existen dos tipos de variable:

```
; Ejemplo número 1
DATOS
        numero  #3
        cadena  "hola"
CODIGO
        cargar_i @0, #2
etiq    cargar_i @1, numero
        cargar32 @1, @1
        sumar @0, @1, @3
```

```
; Ejemplo número 2
CODIGO
        cargar_i @0, #2
        cargar_i @1, #3
        sumar @0, @1, @3
```

Figura 2.1: Ejemplos de programas de Eafitos

- De tipo numérico. Ocupan 32 bits. El valor inicial que toma la variable debe ir precedido por una almohadilla (#).
- De tipo cadena. Ocupan tanto como la longitud de la cadena más uno (por el carácter de final de cadena). La cadena aparece entre comillas.

Las variables siempre deben inicializarse, aunque después no se vaya a utilizar el valor inicial que contienen. Más adelante, en el código, referenciaremos las variables por su nombre, esto no nos dará el valor que contienen sino la dirección de memoria que representan.

A continuación va el código, el cual debe empezar por la palabra CODIGO. El código no es más que una lista de instrucciones, una en cada línea, primero aparece el nombre de la instrucción y después sus operandos separados por comas. Además, opcionalmente, puede aparecer al principio una etiqueta que se podrá utilizar como destino en los saltos. En el primer ejemplo se observa una, aunque en este caso no se llega a utilizar.

En cuanto a los operandos, podemos observar tres formas diferentes de indicarlos:

- Registros. Primero se escribe un carácter arroba (@) y después el número del registro
- Inmediato. Primero aparece un carácter almohadilla (#) y después el valor del dato.
- Referencia a variable. Tan solo aparece el nombre de la variable.

2.2 Arquitectura del procesador

En esta sección estudiaremos el juego de instrucciones con cierto detalle, y veremos todos los aspectos del procesador que interesan al programador de Eafitos.

2.2.1 Registros del procesador

Este es un procesador basado en registros generales, tiene un total de 16 numerados del 0 al 15, que el programador utilizará explícitamente. Además, cuenta con los típicos registros de contador de programa y puntero a pila, que se modifican implícitamente con las instrucciones de salto y las de gestión de la pila respectivamente.

Por otro lado, carece de registro de estado. Los saltos condicionales se realizan en función del valor de los registros generales.

2.2.2 Formato de las instrucciones

La longitud de las instrucciones es variable. El primer byte identifica de que instrucción se trata (por lo tanto, pueden haber hasta 256 instrucciones). A continuación vienen los operandos, que pueden ser de dos tipos:

Registro Referencia a alguno de los 16 registros del procesador. Ocupa un byte.

Inmediato Es un dato de 32 bits.

Existen seis tipos distintos de instrucciones según su número y tipo de operandos. Son:

- Sin operandos. Solo ocupan un byte.
- Con un operando de tipo registro. Ocupan dos bytes.
- Dos operandos de tipo registro. Tres bytes.
- Tres operandos de tipo registro. Cuatro bytes.
- Un operando de tipo inmediato. Cinco bytes.
- Un operando de tipo inmediato y otro de tipo registro. Seis bytes.

2.2.3 Modos de direccionamiento

Existen cuatro modos de direccionamiento:

Directo a registro El valor se encuentra almacenado en el registro. Un registro, un byte.

Inmediato El valor esta contenido en la propia instrucción. Un dato inmediato, cuatro bytes.

Indirecto mediante registro El registro especifica la dirección de memoria donde se encuentra el valor. Un registro, un byte.

Directo a memoria La propia instrucción especifica la dirección de memoria donde se encuentra el valor. Un dato inmediato, cuatro bytes.

2.2.4 Descripción individual de todas las instrucciones

Aritmético/Lógicas

Pueden ser instrucciones de dos o de tres operandos, según el operador que representan sea unario o binario respectivamente. En cualquier caso, el modo de direccionamiento utilizado para todos los operandos es el directo a registro.

Los primeros operandos (uno o dos según sea una operación unaria o binaria) son los registros fuente sobre los que se realiza la operación. El último es el registro destino donde se almacena el resultado.

Existen seis instrucciones en esta categoría:

- **sumar** ***fuente1***, ***fuente2***, ***destino***

  Descripción: suma los dos operandos fuente y almacena el resultado en el destino.

  Modos de direccionamiento de los operandos: directo a registro.

- **restar** ***fuente1***, ***fuente2***, ***destino***

  Descripción: Resta el segundo operando fuente del primero y almacena el resultado en el destino

  Modos de direccionamiento de los operandos: directo a registro.

- **and** ***fuente1***, ***fuente2***, ***destino***

  Descripción: aplica la operación Y lógica a los dos operandos fuente y almacena el resultado en el destino.

  Modos de direccionamiento de los operandos: directo a registro.

- **or** ***fuente1***, ***fuente2***, ***destino***

  Descripción: aplica la operación O lógica a los dos operandos fuente y almacena el resultado en el destino.

  Modos de direccionamiento de los operandos: directo a registro.

- **copiar** ***fuente***, ***destino***

  Descripción: copia del registro fuente al destino.

  Modos de direccionamiento de los operandos: directo a registro.

- **not** ***fuente***, ***destino***

  Descripción: almacena en destino el resultado de aplicar la negación lógica al registro fuente.

  Modos de direccionamiento de los operandos: directo a registro.

De acceso a memoria

Tienen siempre dos operandos. El primero es siempre directo a registro. El segundo puede ser cualquiera de los cuatro modos de direccionamiento, excepto el directo a registro.

- **cargar32** ***destino*, *fuente***

  Descripción: lee de memoria (fuente) un dato de 32 bits y lo almacena en un registro (destino).

  Modo de direccionamiento del destino: directo a registro.

  Modo de direccionamiento del fuente: indirecto mediante registro.

- **guardar32** ***fuente*, *destino***

  Descripción: escribe en memoria (destino) un dato de 32 bits que está almacenado en un registro(fuente).

  Modo de direccionamiento del fuente: directo a registro.

  Modo de direccionamiento del destino: indirecto mediante registro.

- **cargar8** ***destino*, *fuente***

  Descripción: lee de memoria (fuente) un dato de 8 bits y lo almacena en un registro (destino).

  Modo de direccionamiento del destino: directo a registro.

  Modo de direccionamiento del fuente: indirecto mediante registro.

- **guardar8** ***fuente*, *destino***

  Descripción: escribe en memoria (destino) un dato de 8 bits que está almacenado en un registro(fuente).

  Modo de direccionamiento del fuente: directo a registro.

  Modo de direccionamiento del destino: indirecto mediante registro.

- **cargar_i** ***destino*, *fuente***

  Descripción: lee de memoria (fuente) un dato que almacena en un registro(destino).

  Modo de direccionamiento del destino: directo a registro.

  Modo de direccionamiento del fuente: inmediato.

- **guardar_i** ***fuente*, *destino***

  Descripción: escribe en memoria (destino) un dato que esta almacenado en un registro(fuente).

  Modo de direccionamiento del fuente: directo a registro.

  Modo de direccionamiento del destino: directo a memoria.

De gestión de la pila

En esta categoría encontramos dos instrucciones que nos permiten almacenar y recuperar datos en la pila del sistema.

La pila se utiliza para el paso de parámetros cuando realizamos llamadas al sistema. También se podría usar para implementar subrutinas.

Tan solo tienen un operando de tipo registro, el otro está implícito en el registro especial *sp* (puntero a pila). Por lo tanto solo ocupan dos bytes.

La pila crece hacia arriba, esto es, cuando almacenamos un dato se incrementa el puntero a la pila y cuando lo extraemos se decrementa.

- **apilar *fuente***

  Descripción: almacena en la pila el operando *fuente*.
  Modo de direccionamiento del operando: directo a registro.

- **desapilar *destino***

  Descripción: extrae un dato de la pila y lo almacena en *destino*.
  Modo de direccionamiento del operando: directo a registro.

De control de flujo

Aquí están los saltos, que pueden tener uno o dos operandos según sean incondicionales o condicionales. Todo salto tiene siempre como último operando el destino del salto, que lo normal es que se trate de una etiqueta. Los saltos condicionales tienen, además, un operando de tipo registro, en función de cuyo valor se realizará o no el salto. En total son cuatro:

- **saltar *destino***

  Descripción: salta incondicionalmente a la dirección *destino*.
  Modo de direccionamiento del destino: inmediato.

- **saltar0 *fuente*, *destino***

  Descripción: salta a la dirección *destino* solo si el operando *fuente* es cero.
  Modo de direccionamiento del fuente: directo a registro.
  Modo de direccionamiento del destino: inmediato.

- **saltarP *fuente*, *destino***

  Descripción: salta a la dirección *destino* solo si el operando *fuente* es mayor que cero.
  Modo de direccionamiento del fuente: directo a registro.
  Modo de direccionamiento del destino: inmediato.

- **saltarN *fuente*, *destino***

  Descripción: salta a la dirección *destino* solo si el operando *fuente* es menor que cero.
  Modo de direccionamiento del fuente: directo a registro.
  Modo de direccionamiento del destino: inmediato.

Especiales

Aquí hay dos, y ninguna de ellas tiene operandos:

- **nop**

  Descripción: instrucción de no operación, no hace nada.

- **ser_sis**

  Descripción: llamada al sistema, pasa el control al núcleo. Los parámetros se pasan a través de la pila, ver la Sección 2.3 para más detalles.

2.3 Llamadas al sistema

En esta sección vamos a ver como realizar una llamada al sistema y qué servicios proporciona el núcleo de Eafitos al programador.

2.3.1 Realizando una llamada

El núcleo proporciona un conjunto de servicios que el programador puede utilizar mediante la instrucción *ser_sis*. Pero antes de ejecutar dicha instrucción el programador debe almacenar en la pila una serie de parámetros, adecuadamente ordenados. El número y significado de dichos parámetros dependerán del servicio que se solicite, en cualquier caso siempre existe al menos uno. El último parámetro que se introduce en la pila identifica el servicio concreto que requerimos del núcleo. Cada parámetro se almacena directamente en la pila si cabe (si es de 32 o menos bits), pero existen casos en los que se almacena un puntero a la zona de memoria donde se encuentra el parámetro, esto sucede cuando ocupa más de cuatro bytes, por ejemplo en el caso de cadenas de caracteres. Además, el núcleo siempre devuelve un resultado en el registro 0.

2.3.2 Descripción individual de los servicios del sistema

Gestión de hilos

- **Crear hilo**

Código: 1

Parámetro: Dirección en la que empezará a ejecutarse el nuevo hilo.

Resultado: El identificador del nuevo hilo o -1 si la operación ha fracasado.

Descripción: Crea un nuevo hilo del proceso en ejecución.

- **Terminar hilo**

  Código: 2

  Parámetro: identificador del proceso que se quiere eliminar.

  Resultado: Éxito (0) o fracaso (-1) de la operación.

  Descripción: termina el hilo que se especifica.

- **Terminar**

  Código: 3

  Resultado: Éxito (0) o fracaso (-1) de la operación.

  Descripción: termina el hilo actual (no tiene parámetros).

Sistema de ficheros

- **Cambiar unidad**

  Código: 10

  Parámetro: Identificador de la nueva unidad.

  Resultado: Éxito (0) o fracaso (-1) de la operación.

  Descripción: Cambia la unidad de disco actual en la cual se está ejecutando el hilo.

- **Cambiar directorio**

  Código: 11

  Parámetro: Ruta al nuevo directorio (puntero).

  Resultado: Éxito (0) o fracaso (-1) de la operación.

  Descripción: Cambia el directorio de actual en el que se está ejecutando el hilo. El nuevo directorio se especifica con una cadena, que puede ser una ruta absoluta, si empieza por '/', o relativa al directorio actual si no.

- **Crear fichero**

  Código: 12

  Parámetro 1: Nombre del fichero (puntero).

  Parámetro 2: Tipo del fichero, normal (0) o directorio (1).

  Resultado: Identificador del nuevo fichero o -1 si la operación ha fracasado.

  Descripción: Crea el fichero especificado en la unidad de disco actual. Si el fichero ya existe da un error.

- **Abrir fichero**

  Código: 13
  Parámetro: Nombre del fichero (puntero).
  Resultado: Identificador del fichero o -1 si la operación ha fracasado.
  Descripción: Abre el fichero especificado.

- **Cerrar fichero**

  Código: 14
  Parámetro: Identificador del fichero.
  Resultado: Éxito (0) o fracaso (-1) de la operación.
  Descripción: Cierra el fichero especificado.

- **Borrar fichero**

  Código: 15
  Parámetro: Nombre del fichero (puntero).
  Resultado: Éxito (0) o fracaso (-1) de la operación.
  Descripción: Borra el fichero especificado.

- **Leer fichero**

  Código: 16
  Parámetro 1: Identificador del fichero.
  Parámetro 2: Dirección de memoria donde se almacenará la información.
  Parámetro 3: Número de bytes a leer.
  Resultado: Número de bytes leídos o -1 si la operación ha fracasado.
  Descripción: Lee un número de bytes dado del fichero especificado.

- **Escribir fichero**

  Código: 17
  Parámetro 1: Identificador del fichero.
  Parámetro 2: Dirección de memoria origen de la información.
  Parámetro 3: Número de bytes a escribir.
  Resultado: Número de bytes escritos o -1 si la operación ha fracasado.
  Descripción: Lee un número de bytes dado del fichero especificado.

- **Saltar fichero**

  Código: 18
  Parámetro 1: Identificador del fichero.
  Parámetro 2: Desplazamiento.

Parámetro 3: Posición base.

Resultado: Nueva posición dentro del fichero o -1 si la operación ha fracasado.

Descripción: Modifica la posición actual dentro del fichero especificado, la nueva posición se obtiene sumando el desplazamiento a la posición base, que puede ser el principio del fichero (0) o la posición actual (1).

Ejecución de programas

- **Ejecutar**

  Código: 20

  Parámetro: Nombre del fichero (puntero).

  Resultado: Identificador del nuevo hilo o -1 si la operación ha fracasado.

  Descripción: Ejecuta el fichero especificado.

Entrada/Salida

- **Obtener carácter**

  Código: 30

  Resultado: Devuelve el carácter leído.

  Descripción: Lee un carácter de la entrada estándar.

- **Imprimir carácter**

  Código: 31

  Parámetro: Carácter que se quiere imprimir.

  Resultado: Devuelve siempre 0.

  Descripción: Imprime un carácter en la salida estándar.

2.4 Ejemplo

En la Figura 2.2 tenemos el típico programa de ejemplo que escribe el mensaje "¡Hola mundo!" en la pantalla.

En primer lugar está la declaración de variables, en este caso solo hay una. Su nombre es (*mensaje*), es de tipo cadena y representa el mensaje que vamos a imprimir (observar que está acabado en `\n`, símbolo que el compilador traduce a una nueva línea, el compilador también entiende el símbolo `\t` que representa un tabulador, como en C).

Después se encuentra el código, que se divide en dos partes, un bloque de inicialización y un bucle.

```
DATOS
        mensaje         "¡Hola mundo!\n"
CODIGO
        ; inicializamos los valores de los registros
        cargar_i @1, #31        ; código de llamada al sistema
        cargar_i @2, mensaje    ; el mensaje
        cargar_i @3, #1         ; desplazamiento para recorrer la cadena

bucle   cargar8 @4, @2
        saltar0 @4, fin

        apilar @4       ; llamada al sistema
        apilar @1
        ser_sis

        sumar @2, @3, @2
        saltar bucle

fin
```

Figura 2.2: Programa de ejemplo, escribe tres veces un mensaje.

Primero cargamos el registro número 1 con el número del servicio del sistema que vamos a utilizar, el que imprime un carácter por las salida estándar Después cargamos el registro 2 con la dirección de memoria donde se almacena la cadena. Finalmente cargamos en el registro 3 una cantidad fija que sumaremos al registro 2 para recorrer la cadena.

La etiqueta *bucle* identifica la dirección de memoria donde empieza el bucle. Empieza cargando en el registro 4 el carácter que vamos a imprimir y comprobamos que sea distinto de cero (el cero es el valor que representa el final de una cadena). Después se realiza la llamada al sistema, primero apilamos los dos parámetros necesarios (el carácter que queremos imprimir, registro 4, y el número del servicio del sistema, registro 1) y después ejecutamos la llamada. Finalmente se incrementa el registro 2 para que apunte al siguiente carácter de la cadena y se salta al principio del bucle. El bucle terminará cuando se encuentre el carácter de final de cadena (valor 0), lo cual provoca un salto al final del programa (etiqueta `fin`).

2.5 Programación avanzada

2.5.1 Manejo de Ficheros

Todos los procesos tienen asociados una tabla de ficheros. Cuando se abre o crea un fichero se utiliza una entrada de la tabla. Las llamadas abrir y crear devuelven un identificador (un índice dentro de la tabla) con el que después se podrá utilizar para trabajar con el fichero.

Las llamadas al sistema relacionadas con el sistema de ficheros pueden ser utilizadas con cualquier identificador de fichero. Pero además, las dos primeras entradas (identificadores 0 y 1) se usan de forma implícita en las llamadas al sistema de entrada/salida; en concreto, el fichero 0 es la entrada estándar, se utiliza en la llamada **Obtener Carácter**, y el fichero 1 es la salida estándar, se utiliza en la llamada **Imprimir Carácter**. Por defecto la entrada estándar es el teclado y la salida estándar es la pantalla.

Redireccionar la entrada/salida estándar

Cuando se abre o se crea un fichero el identificador devuelto se corresponde siempre con el de la primera entrada libre de la tabla. Así, para redireccionar la entrada estándar lo que tenemos que hacer es cerrar el fichero número 0 e inmediatamente después abrir (o crear) el fichero que queremos sea la nueva entrada estándar.

El fichero *ejemplos/es* es un programa de ejemplo que ilustra como redireccionar la salida estándar.

Acceso a dispositivos de tipo carácter

Los dispositivos de tipo carácter se pueden acceder como si se tratara de ficheros normales. Cada dispositivo de tipo carácter tiene un nombre especial que podemos utilizar para abrirlo. El nombre del teclado es "–teclado–" y el nombre de la pantalla es "–pantalla–".

Parte II

Eafitos por dentro

Capítulo 3

Introducción

3.1 Diseño global

La diferencia más importante de Eafitos con los sistemas operativos reales es el hecho de que Eafitos se ejecuta sobre una máquina virtual en lugar de directamente sobre el hardware. Esto simplifica enormemente las cosas y tiene otras consecuencias como ahora veremos.

Los núcleos de los sistemas operativos reales suelen estar escritos en C y en lenguaje ensamblador. Entonces, hay que compilar el código fuente para generar el binario que se ejecutará en la máquina. De este modo, el núcleo es un programa más (aunque con unas características particulares) que, igual que los programas de usuario, reside en la memoria del sistema y cuyas instrucciones de código máquina ejecuta el procesador.

En Eafitos esto no es así. Eafitos está completamente escrito en C++ y se ejecuta sobre un sistema operativo anfitrión (DOS o UNIX). Así, mientras que los programas de usuario residen en la memoria virtual de la máquina virtual y son ejecutados por su procesador, el núcleo y el resto de herramientas (compilador, intérprete de comandos,...) se ejecutan directamente sobre el sistema operativo anfitrión. Hacerlo como en la realidad, es decir, escribir el núcleo (y el resto de herramientas) en un lenguaje dado y compilarlo para generar código máquina que el procesador virtual pudiese entender, hubiese tenido un coste tremendo ya que hubiese implicado crear un compilador muchísimo más complejo o escribir el código en ensamblador.

La Figura 3.1 muestra un esquema en el que nos apoyaremos para acabar de entender el diseño de Eafitos.

En el esquema, los programas de usuario están dibujados en el interior de la máquina virtual para subrayar el hecho de que residen en la memoria virtual. El núcleo de Eafitos es el encargado de gestionar la máquina virtual y de proporcionar a los programas de usuario una serie de facilidades. La flecha que va de la máquina virtual al núcleo hace referencia a las llamadas al sistema y a la

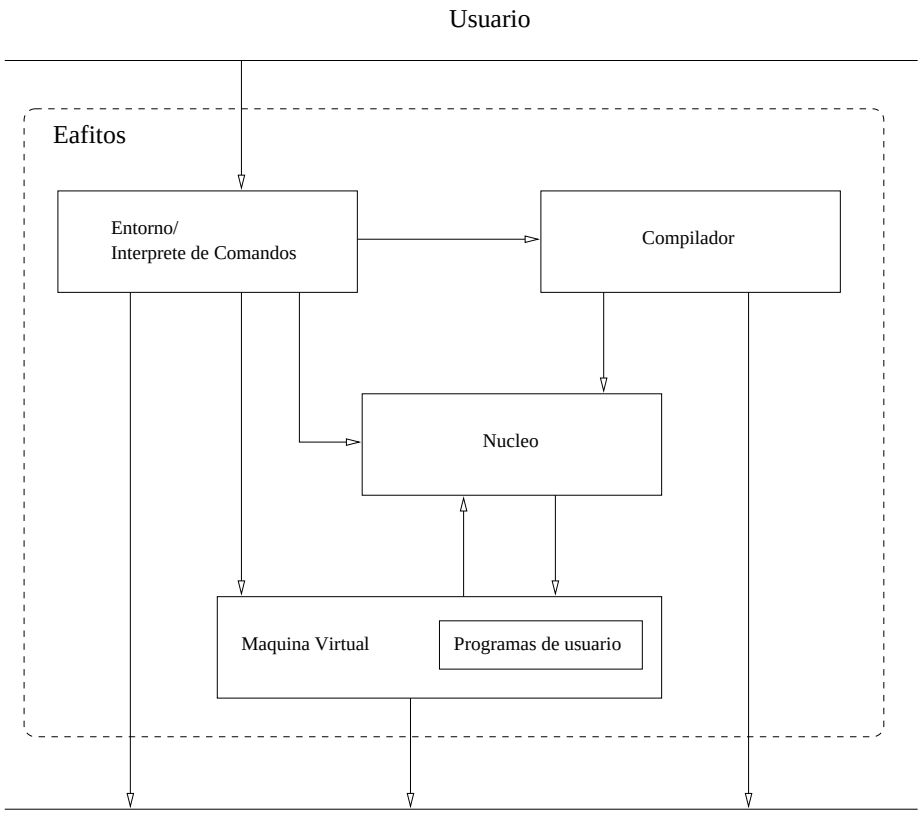

Figura 3.1: Diseño global de Eafitos.

interrupción de reloj (ver Capítulo 5).

Tres de las cuatro partes de Eafitos utilizan recursos directamente del sistema operativo anfitrión. El entorno y el intérprete de comandos lo hacen para interactuar con el usuario, el compilador para presentar mensajes de error o de éxito y la máquina virtual para implementar sus dispositivos.

El usuario interactúa directamente tan solo con el entorno (menús iniciales para la gestión de los discos) y el intérprete de comandos. Es pues el intérprete de comandos el que se ejecuta la mayor parte del tiempo, el control solo pasa a la máquina virtual cuando se ejecuta algún programa.

3.2 Estructura del código fuente

Dentro del directorio *programa* tenemos el código fuente que se distribuye en los siguientes directorios:

- *include* Todos los ficheros de cabecera residen en este directorio.
- *mv* Aquí están los ficheros de implementación de la máquina virtual.
- *nucleo* Aquí están los ficheros de implementación del núcleo. Este directorio tiene, además, los siguientes subdirectorios:
  - *es* Aquí encontrarás los ficheros que implementan la entrada/salida.
  - *sf* Aquí residen los ficheros que implementan el sistema de ficheros.
  - *ejec* Aquí encontrarás los ficheros que implementan los formatos de ficheros ejecutables.
- *entorno* En este directorio están los ficheros que implementan el intérprete de comandos, el compilador y el sistema de menús para la gestión de los discos.

El fichero que contiene la función `main` es *programa/eafitos2.cpp*.

3.3 Cuestiones de implementación

3.3.1 Implementación de las clases principales

La Figura 3.1 muestra a Eafitos dividido en cuatro partes, aunque podemos considerar que solo hay tres si unimos el compilador al entorno y al intérprete de comandos, ya que así es como está implementado.

Disponemos de tres clases principales: `MaqVirtual`, `Nucleo` y `Entorno` cuyos miembros (atributos y métodos) son estáticos y públicos. Es aconsejable que mires los ficheros de cabecera donde están definidos estas tres clases: *mv.h*, *nucleo.h* y *entorno.h*.

Estas clases, fundamentalmente hacen la función de contenedoras de otros objetos, los cuales pueden ser referenciados con, por ejemplo, **`Nucleo::xxx.yyy().`** De hecho, el código contiene muchas de estas referencias. Es algo así como si fueran variables globales, esto simplifica mucho el código.

Además, estas clases principales contienen, entre otras cosas, código de inicialización y terminación que se estudiará con más detalle en los próximos capítulos.

La función `main`

Mira también la función **`main`** en el fichero *eafitos2.cpp*. Todo lo que hace es iniciar la máquina virtual y en núcleo, ceder el control al entorno y terminar el núcleo y la máquina virtual.

3.3.2 Gestión de errores

En principio la gestión de errores estaba implementada como se suele hacer en C, haciendo que los métodos devolvieran un código de error (normalmente -1) cuando se detectase un error, y evaluando el código devuelto por los métodos.

Pero más adelante decidí utilizar los mecanismos de gestión de excepciones de C++ (**`throw`**, **`try`** y **`catch`**) porque facilitan muchísimo el trabajo. En general la sentencia **`throw`** siempre devuelve una cadena, y los bloques **`try`** - **`catch`** se limitan o a ignorar el error o a imprimir un mensaje de error y abortar la operación. Esto es cierto excepto en un caso particular en la clase **`Nucleo`**, donde están definidas dos clases (**`NoHayHilos`** y **`NoHayHilosListos`**) que se utilizan para poder distinguir la causa de la excepción y poder dar así un tratamiento específico.

Ver include/nucleo.h

En cualquier caso, lo que te quiero decir es que si vas a modificar Eafitos debes utilizar también los mecanismos de gestión de excepciones de C++, vale la pena.

Capítulo 4

La Máquina Virtual

La Máquina Virtual simula un ordenador real. Consta de varios componentes: la memoria, la unidad de gestión de memoria, el procesador y los dispositivos (actualmente un teclado, una pantalla y uno o más discos duros). En la Figura 4.1 se puede observar un dibujo que refleja la estructura de la máquina virtual. Cómo se puede ver, no hay relación entre los dispositivos y el procesador; en los sistemas reales existen diversos mecanismos (interrupciones, acceso directo a memoria, puertos de entrada/salida,...) que en Eafitos se han obviado para simplificar.

Ver include/mv.h

4.1 La memoria

La memoria está implementada como la clase **`MemFis.`** Fundamentalmente consta de un vector de bytes, que representa a la memoria en sí, y de los métodos que permiten acceder a ella, a nivel de byte, **`leerByte`** y **`escribirByte`** , o a nivel de palabra (cuatro bytes), **`leerPalabra`** y **`escribirPalabra.`**

Ver include/memfis.h

4.2 Unidad de gestión de memoria

La unidad de gestión de memoria, o MMU (Memory Management Unit), proporciona facilidades al núcleo para la gestión de memoria y es la responsable de calcular las direcciones físicas de memoria a partir de direcciones lógicas.

Ver include/mmu.h

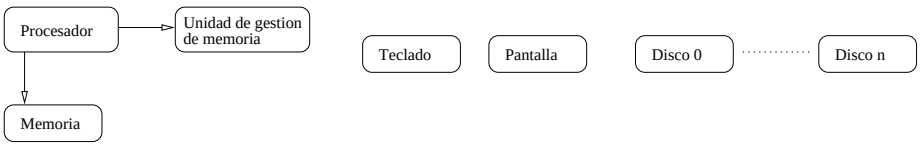

Figura 4.1: Estructura de la máquina virtual.

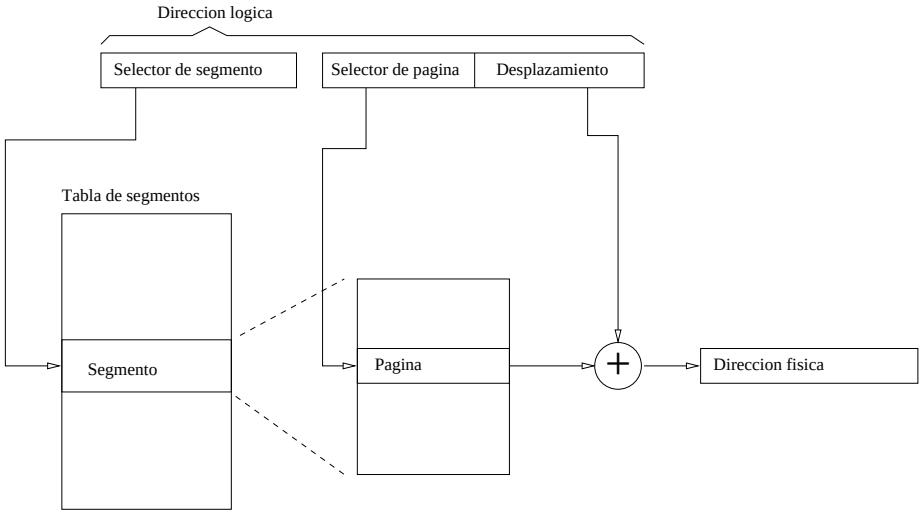

Figura 4.2: Traducción de una dirección lógica a una dirección física

La gestión de memoria de la máquina virtual sigue un modelo que combina segmentación y paginación. Existe una única tabla de segmentos (el vector **`MMU::segmentos`**) y cada segmento define un conjunto de páginas (**`Segmento::paginas`**).

4.2.1 Cálculo de la dirección física

Ver MMU::logicaAFisica en include/mmu.cpp

En la Figura 4.2 puedes ver un dibujo ilustrativo. El *selector de segmento* indica qué segmento utilizar para el cálculo. Una vez tenemos el segmento se obtiene la página lógica mediante el *selector de página*. La página lógica nos da la página física. Ahora calculamos la dirección física con la siguiente ecuación:

$$(\textit{página física} * \textit{tamaño de página}) + \textit{desplazamiento} = \textit{dirección física}$$

donde *tamaño de página* es una constante.

Pero, ¿de donde sale la dirección lógica? El *selector de segmento* es un registro especial de la CPU. Existen dos de estos registros, se utiliza uno u otro dependiendo de la instrucción que se ejecute; es por lo tanto una selección implícita, un mecanismo transparente para el programador. En la Sección 4.3 veremos esto con más detalle.

Por otro lado, el *selector de página* y el *desplazamiento* constituyen las direcciones de 32 bits que el programador maneja; dichas direcciones están contenidas en registros de manipulables por el programador: los registros generales, el contador de programa y el puntero de pila. El número de bits que ocupan ambos campos, *selector de página* y *desplazamiento*, depende del *tamaño de página*, el cual está definido por la constante **`TAM_PAGINA.`** En principio el valor de dicha constante es 1024, por lo tanto el *selector de página* ocupa 10 bits y el

desplazamiento ocupa 22. Puedes modificar el *tamaño de página* con la única condición de que sea potencia de 2.

4.2.2 Reservar y liberar memoria

Además de la tabla de segmentos, la unidad de gestión de memoria contiene un vector (**`MMU::paginas`**) que contiene información sobre cada página física. En concreto, la información almacenada nos indica si la página está libre u ocupada. Este vector, junto con el atributo **`MMU::nPaginasLibres`** (indica el número de páginas libres que quedan), le sirve a la MMU para ayudarse en la gestión de la memoria.

Solicitando memoria

El método **`MMU::asignar`** sirve para solicitar una determinada cantidad de memoria. La MMU reserva un segmento y las páginas físicas necesarias (si existe suficiente memoria libre) y devuelve el selector del segmento que se ha reservado.

También se puede incrementar el *uso* (**`Segmento::uso`**) de un segmento que ya esté ocupado con el método **`MMU::reservar,`** esto permite la compartición de memoria.

Liberando memoria

El método **`MMU::liberar`** reduce el uso del segmento indicado. Si el segmento ya no se utiliza (su uso es cero) libera la memoria que tiene asociada y el propio segmento.

4.3 El procesador

La clase **`CPU`** representa al procesador. El objetivo esencial de esta unidad es el de ejecutar instrucciones.

Ver include/cpu.h

4.3.1 Contexto

El procesador de la máquina virtual tiene un conjunto de registros, algunos de los cuales ya vimos en la Sección 2.2.1 desde el punto de vista del programador. Ahora vamos a verlos todos con más detalle.

Los registros están definidos en la clase **`Contexto,`** ya que forman el contexto de ejecución de un hilo.

Ver include/contexto.h

Por un lado tenemos 16 registros generales (**`Contexto::registros`**) para uso del programador. También hay un contador de programa (**`Contexto::pc`**) y un puntero de pila (**`Contexto::sp`**).

Finalmente, tenemos dos registros especiales, **`Contexto::codigo`** y **`Contexto::pila;`** estos registros son selectores de segmento. El registro de *pila* se utiliza en las instrucciones de acceso a la pila y el de *codigo* en el resto de instrucciones. El programador no tiene forma alguna de modificar estos registros, ni explicita ni implícitamente.

4.3.2 Acceso a memoria

Ver mv/cpu.cpp

En la clase **`CPU`** hay varios métodos definidos para acceder a memoria. Estos métodos permiten leer y escribir bytes y palabras, son **`CPU::leerByte`**, **`CPU::escribirByte`**, **`CPU::leerPalabra`** y **`CPU::escribirPalabra`**.

Lectura de instrucciones y operandos

Por otro lado los métodos **`CPU::leerOpByte`** y **`CPU::leerOpPalabra`** leen de la memoria utilizando el *contador de programa* como dirección. Estos métodos se utilizan para leer las instrucciones y sus operandos inmediatos. Además de leer de memoria actualizan automáticamente el *contador de programa*.

La pila

A la pila se accede mediante los métodos **`CPU::apilar`** y **`CPU::desapilar`**. Es en estos métodos donde está definido su comportamiento. En concreto, la pila está implementada para crecer hacia "arriba", es decir, cuando se apila un dato el *puntero de pila* (**`CPU::contexto.sp`**) se incrementa y cuando se desapila se decrementa.

4.3.3 Ejecución de instrucciones

Los métodos principales del procesador son **`ejecutarPaso`** y **`ejecutar`**. El primero ejecuta una instrucción mientras que el segundo llama al primero para ejecutar instrucciones hasta que no quede ninguna, es decir, hasta que todos los hilos hayan terminado. [1]

Las excepciones que se puedan provocar como consecuencia de la ejecución de una instrucción (ej: fallo de página) se capturan en **`CPU::ejecutar`** y provocan la terminación del hilo.

[1] La razón para distinguir entre dos métodos es la de facilitar la construcción de un depurador. Así, sería fácil ejecutar instrucciones "paso a paso".

El método **CPU::ejecutarPaso** es muy sencillo, aunque extenso. Para empezar se ejecuta el método **Nucleo::reloj,** el cual cede el control al núcleo. Esto simula las interrupciones de reloj que se producen cada cierto tiempo en las máquinas reales, solo que en este caso se "producen" antes de ejecutar cada instrucción. ¿Y qué es lo que hace el núcleo en esas *interrupciones de reloj*? Eso se estudiará cuando veamos el núcleo, en los siguientes capítulos.

Una vez hecho esto, se pasa a ejecutar una instrucción. Primero se adquiere el código de instrucción; después se decodifica, lo cual consiste en saltar dentro del **switch** al lugar adecuado; finalmente se adquieren los operandos y se ejecuta. Mejor le hechas un vistazo al código.

4.4 Dispositivos

4.4.1 El teclado

El teclado virtual sencillamente nos proporciona la posibilidad de leer un carácter del teclado real.

Ver include/teclado.h

4.4.2 La pantalla

La pantalla virtual sencillamente nos proporciona la posibilidad de escribir un carácter en la pantalla real.

Ver include/pantalla.h

4.4.3 Los discos

Este es el dispositivo más complicado. Al contrario que el teclado y la pantalla, la máquina virtual puede contener varios discos, tantos como indica la constante **NUM_DISCOS.**

Ver include/disco.h

Un disco es, en la máquina real, un fichero que reside en el directorio "\EAFITOS" o "$HOME/EAFITOS", dependiendo de si el Sistema Operativo anfitrión es DOS o GNU/Linux; el nombre del fichero es DISCO_*n*, donde *n* es el número del disco (de 0 a **NUM_DISCOS** -1).

El disco se divide en bloques de igual tamaño. El número de bloques y su tamaño puede variar de un disco a otro, dichos valores están especificados al principio del fichero real. Tras estos dos valores vienen los bloques en sí, en la Figura 4.3 tienes un esquema. Una vez creado el disco no se puede cambiar ni su tamaño ni su número de bloques; si lo cambias desde el SO anfitrión perderás la información que contenga esa unidad.

Creación de nuevos discos

Para crear nuevos discos disponemos del método **Disco::crearDisco,** el

Ver mv/disco.cpp

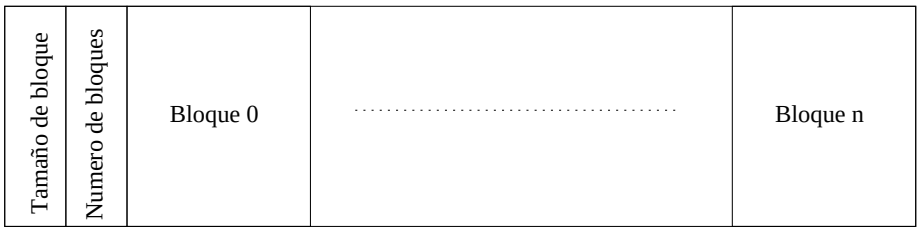

Figura 4.3: Esquema de un disco.

cual recibe como parámetros el número del disco, *tamaño de bloque* y el *número de bloques* que queremos tenga el nuevo disco.

4.5 Arranque y parada

La memoria, la unidad de gestión de memoria y el procesador no necesitan ser inicializados. Ni el teclado ni la pantalla, ya que estos dispositivos son "estáticos", es decir, siempre hay uno y solamente uno de cada. Por el contrario, puede haber un número variable de discos que hay que detectar e inicializar en el arranque.

Ver mv/mv.cpp

El método **`MaqVirtual::iniciar`** es el responsable del arranque de la máquina y el método **`MaqVirtual::terminar`** lo es de su parada. El primero inicializa el vector **`MaqVirtual::discos`** utilizando el constructor de la clase **`Disco.`** El segundo (**`terminar`**) "limpia" dicho vector.

4.6 Modificar la máquina virtual

4.6.1 Añadir instrucciones al procesador

Primero hay que añadir la instrucción al tipo enumerado **`Instrucciones`** y después modificar el método **`CPU::ejecutarPaso`** añadiendo una nueva entrada al **`switch`** que haga lo que corresponda.

Ten en cuenta que para poder utilizar las nuevas instrucciones en el lenguaje ensamblador también tendrás que modificar el compilador. Además, si no has colocado todas las nuevas instrucciones al final de **`Instrucciones,`** tendrás que eliminar los ejecutables y volver a compilar los programas, ya que de lo contrario es probable que no funcionen.

4.6.2 Añadir nuevos dispositivos

Simplemente escribe un fichero de cabecera *nombre.h* y uno de implementación *nombre.cpp* donde se defina e implemente la clase que representa al dispositivo;

después añade una instancia, o varias, según corresponda, de dicha clase a la maquina virtual (**`MaqVirtual`**). Finalmente, añade el código de inicialización/creación que sea necesario, fíjate en el caso de los discos como ejemplo.

Capítulo 5

El núcleo: Introducción

El núcleo se divide en cinco partes: el manejador de memoria, el manejador de hilos y procesos, los dispositivos de tipo carácter, los dispositivos de tipo bloque y el sistema de ficheros virtual. Dichos elementos están contenidos en la clase **`Nucleo`**, y los estudiaremos por separado en los capítulos siguientes.

Ver include/nucleo.h

5.1 Iniciar y terminar

La inicialización y la terminación del núcleo consiste en la inicialización y terminación de sus partes. Los métodos **`iniciar`** y **`terminar`** son los encargados de hacer esta tarea.

5.2 Llamadas al sistema

Las llamadas al sistema proporcionan ciertos servicios que el programador puede utilizar. Además de las cinco partes del núcleo arriba señaladas, existe un interfaz de llamadas al sistema, el cual está implementado en el método **`Nucleo::llamada`**. La Figura 5.1 muestra un esquema de esto, como se puede observar, **`llamada`** es una especie de multiplexor que, en función del valor almacenado en la cima de la pila, llama a otro método de algún otro subsistema del núcleo.

Ver nucleo/nucleo.cpp

5.2.1 Añadir nuevas llamadas

Añadir una nueva llamada al sistema es fácil. Primero hay que definir una nueva constante para la nueva llamada, esto se hace al principio del fichero "include/nucleo.h"; después se añade una entrada dentro del bloque **`switch`** en el método **`Nucleo::llamada`** que se encuentra en el fichero "nucleo/nucleo.cpp", en dicho código lo único que se debe hacer es desapilar los parámetros y llamar

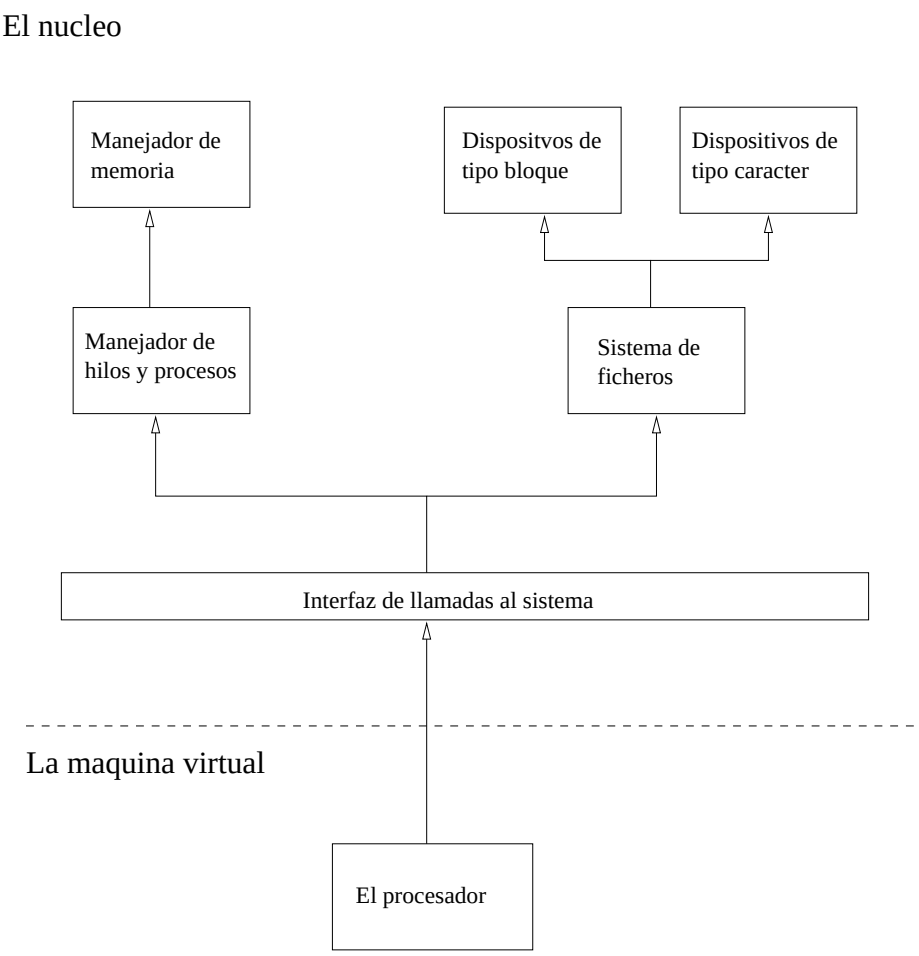

Figura 5.1: Esquema básico del núcleo y las llamadas al sistema. El dibujo está muy simplificado, tan solo pretende ilustrar la relación entre el procesador, el interfaz de llamadas y el resto del núcleo. Las relaciones que muestran las flechas internas del núcleo son incompletas.

al método del núcleo que corresponda; finalmente, quedarán por hacer las modificaciones necesarias al resto del núcleo para que la llamada haga lo que se quiera.

5.3 Interrupciones de reloj

Los sistemas reales disponen de una interrupción de reloj que produce el hardware siempre a intervalos de tiempo regulares. Al producirse dicha interrupción deja de ejecutarse momentáneamente el proceso que se estuviera ejecutando y se ejecuta una rutina de servicio. Dicha rutina se dedica a realizar tareas varias que pueden desencadenar, por ejemplo, el cambio del proceso en ejecución.

En Eafitos esto está simulado de la siguiente manera: antes de ejecutar cada instrucción el procesador de la máquina virtual [1] llama al método **`Nucleo::reloj`**, esto es "la generación de la interrupción de reloj"; y el método **`Nucleo::reloj`** es "la rutina de servicio de la interrupción de reloj".

El método **`Nucleo::reloj`** lo que hace es llamar a otros tres métodos:

- **`DispositivosCaracter::gestionarColas`**
- **`DispositivosBloque::gestionarColas`**
- **`ManejadorProcesos::planificador`**

Lo qué hacen estos métodos lo veremos en los capítulos siguientes.

[1] Ver Sección 4.3.

Capítulo 6

El núcleo: gestión de memoria

La parte del núcleo responsable de la gestión de memoria está representada por la clase **`ManejadorMemoria.`** El manejador de memoria se apoya en la unidad de gestión de memoria de la máquina virtual para realizar su tarea, de hecho la mayor parte del trabajo lo realiza la *MMU*.

Ver include/memoria.h

6.1 Reservar y liberar memoria

Esta es la tarea principal del manejador de memoria. Para realizarla disponemos de tres métodos, **`asignar,`** **`reservar`** y **`liberar.`** Estos métodos lo único que hacen es llamar a sus equivalentes en la unidad de gestión de memoria. El primero, **`asignar,`** reserva la memoria solicitada y devuelve el *selector de segmento* reservado; el método **`reservar`** incrementa el uso del segmento especificado; finalmente, **`liberar`** reduce el uso del segmento especificado y, si ya no se utiliza (su uso es cero), libera al segmento y a la memoria asociada.

6.2 Copiar desde y hacia espacio de usuario

En los sistemas operativos reales suelen existir unas funciones que copian datos de memoria de usuario a memoria del núcleo y viceversa. En Eafitos también, solo que la memoria del núcleo no reside en la memoria de la máquina virtual, sino directamente en la memoria del sistema operativo anfitrión.

Estos métodos son **`aUsuario`** y **`deUsuario`** , los cuales copian, respectivamente, datos de memoria del núcleo a memoria de usuario y viceversa. Para se ello utilizan los métodos **`CPU::escribirByte`** y **`CPU::leerByte`** .

Ver nucleo/memoria.cpp

6.3 Modificar el modelo de memoria

Modificar el modelo de memoria implica no solo modificar el manejador de memoria, sino sobre todo modificar la unidad de gestión de memoria en la máquina virtual, ya que la mayor complejidad reside en la *MMU*. De hecho, dependiendo de los cambios que se quieran realizar, es probable que no sea necesario modificar el manejador de memoria en nada. El modelo de memoria no está pensado especialmente para que sea fácilmente modificable, pero algunas cambios sencillos no deberían ser difíciles de hacer, ya que los mecanismos básicos de paginación y segmentación están implementados.

Capítulo 7

El núcleo: hilos y procesos

Eafitos distingue perfectamente entre hilos y procesos. Los hilos son los elementos activos del sistema operativo mientras que los procesos son un conjunto de recursos. Un hilo tiene siempre asociado un único proceso, mientras que un proceso puede tener asociados uno o más hilos.

Cada uno de estos elementos, hilos y procesos, están definidos como clases, **`Hilo`** y **`Proceso`** respectivamente. Existen también otras clases relacionadas directamente con la gestión de hilos y de procesos, tales como **`Cola`** y **`ManejadorProcesos.`**

7.1 Hilos

7.1.1 Estados

Un hilo puede estar en uno de tres estados, tal como muestra la Figura 7.1.

Ver include/hilo.h

Cuando se crea un hilo este se encuentra inicialmente en el estado *Listo*, es decir se encuentra disponible para ser ejecutado. Pueden haber varios hilos en estado *Listo*.

En un momento dado el hilo que se encuentra en *Ejecución* en estos momentos pasará a estado *Listo* y uno de los hilos que se encuentra en *Listo* pasará a *Ejecución*. Solo puede haber un hilo en *Ejecución*, que es aquel que está siendo ejecutado en estos momentos por la CPU. La política según la cual se realiza este paso de *Listo* a *Ejecución* y viceversa se implementa en el planificador, que estudiaremos más adelante.

En un momento dado el hilo que se encuentra en *Ejecución* puede realizar una petición de entrada/salida, esto hará que dicho hilo pase a estado *Suspendido* y que otro hilo en *Listo* pase a *Ejecución*. Cuando la petición de entrada/salida se haya satisfecho el hilo que la realizó pasará a *Listo*.

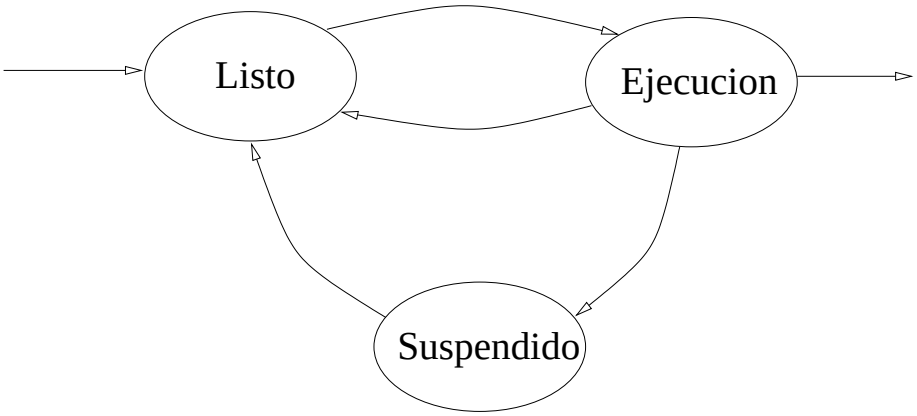

Figura 7.1: Estados de un hilo

Existen dos formas de que un hilo termine: de forma natural, cuando acaba su ejecución (como muestra la flecha que sale de *Ejecución* hacia ninguna parte); y de forma "violenta" cuando algún otro hilo le "mata", esta segunda forma no aparece reflejada en la Figura ya que puede producirse independientemente del estado en el que se encuentre el hilo.

El atributo **`Hilo::estado`** guarda el estado actual del hilo que puede ser **`LISTO`**, **`EJECUCION`** o **`SUSPENDIDO`**. Los métodos **`iniciar`**, **`ejecutar`**, **`dormir`**, **`suspender`**, **`reactivar`** y **`terminar`** implementan las transiciones de un estado a otro, es decir, representan los arcos de la Figura 7.1.

Ver núcleo/hilo.cpp

Cambio de contexto

Las transiciones a y desde el estado de *Ejecución* implican el cambio del contexto del hilo, el cual, como ya vimos en la Sección 4.3.1, está formado por los registros del procesador.

Cuando un hilo sale de *Ejecución* se guarda su contexto, es decir, se copia **`CPU::contexto`** en **`Hilo::contexto`**. Y al revés, cuando un hilo pasa de *Listo* a *Ejecución* se carga su contexto, es decir, se copia **`Hilo::contexto`** en **`CPU::contexto`**.

7.1.2 Las colas de hilos

Eafitos puede tener un número variable de hilos ejecutándose, desde 0 hasta el valor de la constante **`N_HILOS`**. Existe una tabla (un vector) que contiene una entrada para cada hilo, su definición está en la clase **`Cola`**. El atributo **`Cola::hilos`** es la tabla de hilos, fíjate que está declarada como **`static`**, lo que significa que solo existe una tabla en todo el sistema. Cada entrada de la tabla puede estar libre u ocupada, una entrada está libre si el atributo

Ver include/cola.h

Hilo::estado tiene el valor **LIBRE;** eso significa que dicho atributo puede tomar, en realidad, cuatro valores: **LIBRE, LISTO, EJECUCION** y **SUSPENDIDO.**

La clase **Cola,** además de contener la tabla de hilos, implementa una cola de hilos. Dicha cola es en realidad una lista circular doblemente enlazada. El atributo **primero** nos identifica el hilo que es cabeza de la cola, y los atributos **anterior** y **siguiente** de la clase **Hilo** apuntan, como ya habrás supuesto, al hilo anterior y siguiente en la cola. Los métodos **actual, insertar** y **extraer** nos permiten gestionar la cola.

Ver nucleo/cola.cpp

Las colas de hilos sirven para mantener ordenados a los hilos cuando se encuentran en estado *Listo* o *Suspendido*.

7.2 Procesos

7.2.1 Relación con los hilos

Por un lado en la clase **Hilo** se define el atributo **proceso** el cual referencia (es un puntero) al proceso asociado; por otro lado, la clase **Proceso** tiene un atributo llamado **uso** que nos indica el número de hilos asociados al proceso. Como se puede ver existe una relación 1 a n entre los procesos y los hilos.

Ver include/proceso.h

También se ve que los hilos quedan así agrupados, perteneciendo a un mismo grupo aquellos que están asociados al mismo proceso. Pero el proceso no es lo único que comparten los hilos del mismo "grupo". Cada hilo tiene dos segmentos de memoria asociados, uno para el código y los datos estáticos y otro para la pila. Todos los hilos de un mismo proceso comparten el mismo segmento de código y datos estáticos, pero el segmento de pila no se comparte, es independiente.

7.2.2 Vida

Un proceso nace cuando se ejecuta un programa (ver Sección 7.4); su uso crece cada vez que se crea un hilo (ver Sección 7.3.1) y decrece cada vez que termina un hilo (ver Sección 7.3.2); cuando su uso llega a cero el proceso termina.

El método **Proceso::iniciar** es llamado cada vez que se crea un hilo y el método **Proceso::terminar** cada vez que uno termina [1].

Ver nucleo/proceso.cpp

7.2.3 Recursos

Un hilo puede utilizar los recursos del proceso asociado. Básicamente el proceso nos proporciona acceso al sistema de ficheros y a la entrada/salida, de hecho, la mayor parte de las llamadas al sistema pasan por la clase **Proceso.**

[1] Evidentemente, nos referimos siempre a hilos asociados al proceso.

Unidad y directorio

Los atributos **`Proceso::unidad`** y **`Proceso::directorio`** indican la unidad y el directorio actuales, respectivamente. Ambos parámetros se utilizan en las llamadas al sistema relacionadas con el sistema de ficheros, la operación requerida se realizará siempre sobre la unidad actual, mientras que el directorio actual se utiliza para encontrar un fichero cuando se utilizan rutas relativas en lugar rutas de absolutas.

Los métodos **`cambiarUnidad`** y **`cambiarDirectorio`** permiten modificar estos atributos, implementan las llamadas al sistema equivalentes.

Sistema de ficheros

El atributo **`ficherosLocales`** es una tabla que contiene los ficheros abiertos por el proceso. Estos ficheros pueden referenciar a un fichero normal (ver Capítulo 9) o a un controlador de dispositivo de tipo carácter (ver Capítulo 8). Cada uno de estos ficheros se identifica por su posición en la tabla (vector). Los métodos **`crear`**, **`abrir`**, **`cerrar`**, **`borrar`**, **`leer`**, **`escribir`** y **`saltar`** implementan las llamadas al sistema equivalentes, nos permiten trabajar con el sistema de ficheros. El método **`ejecutar`** sirve para ejecutar programas.

Entrada/Salida estándar

Como en UNIX, en Eafitos los dispositivos se tratan como si fueran ficheros, pero la implementación es distinta (más sencilla) que la de UNIX. Esto permite acceder a ellos con los mismos métodos con los que accedemos a los ficheros normales.

Además, están definidas una entrada y una salida estándar, que son los ficheros locales número 0 y 1 respectivamente. Específicamente para acceder a la entrada/salida estándar, los procesos disponen de los métodos **`leerCaracter`** e **`imprimirCaracter`**, los cuales, como sus nombres indican, leen y escriben un solo carácter de la entrada y en la salida estándar respectivamente.

7.3 El manejador de hilos y procesos

Ver include/procesos.h

Este manejador está definido como la clase **`ManejadorProcesos`**. Fíjate primero en que dicha clase desciende de la clase **`Cola`**, esto significa que **`ManejadorProcesos`** tiene acceso a la tabla de hilos; en concreto, este manejador gestiona los hilos que se encuentran en estado *Listo* y el hilo que está en *Ejecución*. Además, contiene la tabla de procesos (atributo **`procesos`**) e implementa las llamadas al sistema de creación/terminación de hilos y ejecución de programas.

7.3.1 Creación de hilos

Como ya vimos en la Sección 2.3 la llamada al sistema **`CREAR_HILO`** crea un hilo que es una copia del hilo en *Ejecución*. El nuevo hilo estará asociado al mismo proceso que el actual y ambos compartirán la memoria de código y datos (selector de segmento **`Contexto::código,`** pero la memoria de pila (selector de segmento **`Contexto::pila`**) será diferente.

Esta llamada al sistema está implementada por el método **`ManejadorProcesos::crearHilo,`** el cual a su vez hace uso de uno de los dos métodos **`Hilo::iniciar`** [2] . Te aconsejo en este punto que le peches un vistazo al código que está bien comentado.

7.3.2 Terminación de hilos

Existen dos llamadas al sistema relacionadas, una termina el hilo actual y otra termina un hilo cualquiera (su identificador se pasa como parámetro).

Ambas llamadas son implementadas por el método **`terminarHilo,`** el cual, lo único que hace, además de un par de comprobaciones, es llamar al método **`Hilo::terminar`** encargado, como su nombre indica, de terminar el hilo. Una vez más, te invito a que le peches un vistazo al código.

7.3.3 El planificador

Este es el responsable de la política por la cual un hilo pasa de *Listo* a *Ejecución* y viceversa.

El planificador está implementado en el método **`planificador,`** el cual hace uso de los métodos **`Hilo::ejecutar`** e **`Hilo::dormir.`**

La política implementada es *Round-Robin*, es decir, cada cierto tiempo el hilo en ejecución pasa a ocupar el último lugar de la cola de hilos listos, y el primer hilo de la cola pasa a ejecución. El tiempo se mide en número de instrucciones ejecutadas, la constante **`TAJADA`** nos da dicho número.

7.4 Ejecución de programas

7.4.1 Formatos de fichero ejecutable

Los programas residen en ficheros con un formato dado, hay que leer dichos ficheros e interpretarlos.

[2] Aquí, como en otros sitios, hago uso del polimorfismo. Hay dos métodos iniciar porque hay dos formas de crear un hilo, a partir de un hilo ya existente o como uno nuevo e independiente (cuando se ejecuta un programa, ver Sección 7.4).

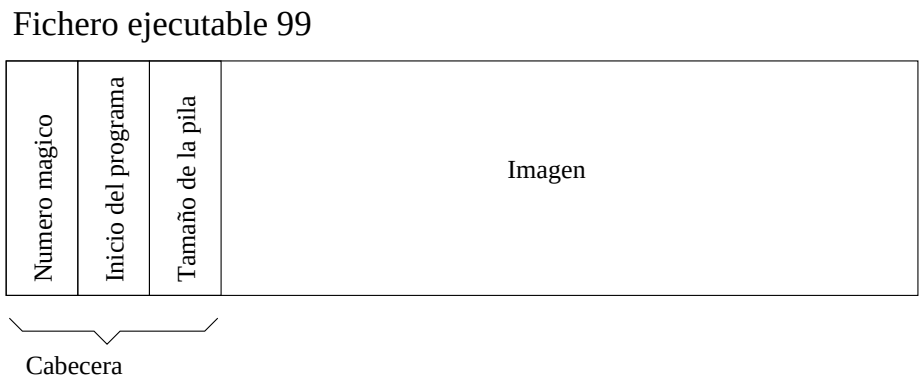

Figura 7.2: Formato de un fichero ejecutable 99.

Eafitos tiene definido un formato de fichero ejecutable concreto llamado 99 [3] , el cual está definido por la clase **`Ejecutable99.`** Pero además, Eafitos dispone de los recursos necesarios para poder definir nuevos formatos y añadirlos fácilmente al núcleo.

Crear nuevos formatos de ejecutables

Ver include/ejec.h

Para crear un nuevo formato de ejecutable debes crear una nueva clase que herede de la clase base **`Ejecutable.`** El fichero de implementación (*.cpp*) de la nueva clase deberá residir en el directorio "nucleo/ejec/". Lo único que debe implementar la nueva clase es el método **`ejecutar.`**

Una vez hecho esto tan solo queda registrar el nuevo formato en el núcleo, esto consiste en ocupar una entrada del vector **`ejecutables,`** el método **`ManejadorProcesos::iniciar`** se encarga de hacerlo.

Si vas a crear un nuevo formato te aconsejo que estudies el formato que ya existe, el 99.

Formato 99

Ver include/ejec99.h

La clase **`Ejecutable99,`** que desciende de **`Ejecutable,`** es la que implementa el formato 99. En la Figura 7.2 puedes ver la estructura de un fichero con este formato.

En primer lugar se encuentra la cabecera que contiene información varía y después la imagen del ejecutable que habrá que cargar en memoria.

La cabecera empieza con el número mágico que identifica al fichero como ejecutable 99, el número es 2323 [4] . A continuación viene la dirección donde empieza el código, es decir, el valor con el que hay que inicializar el contador de programa

[3] Por el año 1999. Lo mio no es poner nombre a las cosas.
[4] Dos veces mi edad en el momento de escribir esto.

(**Contexto::pc**). Por último se encuentra el tamaño mínimo que debe tener la pila.

7.4.2 Ejecución

El método **ManejadorProcesos::ejecutar** es el responsable de realizar esta operación. La ejecución de un programa provoca la creación de un hilo y de un proceso, por ello, lo primero que hace **ejecutar** es buscar entradas libres en las tablas de hilos y de procesos. Tras esto abre el fichero que se supone contiene la imagen del ejecutable. A continuación recorre el vector **ManejadorProcesos::ejecutables** llamando cada vez al método **Ejecutable::ejecutar,** hasta que el fichero es reconocido, entonces se crea el hilo y el procesos llamando al método **Hilo::iniciar** . Si el fichero no se reconoce como ejecutable se genera un error.

Ver **ejecutar** en nucleo/procesos.cpp

Capítulo 8

El núcleo: Entrada/Salida

Cada tipo de dispositivo de la máquina virtual debe tener un controlador asociado en el núcleo, o de lo contrario no podrá ser utilizado por los programas de usuario.

Los controladores se pueden clasificar, al igual que los dispositivos, en dos grupos, de carácter o de bloque. Los controladores de tipo carácter leen y escriben en el dispositivo de forma secuencial, cada vez un solo carácter; ejemplos de estos dispositivos son el teclado y la pantalla. Los de tipo bloque leen y escriben información en bloques de datos de longitud fija y su acceso no es secuencial, es directo; un ejemplo de este tipo son los discos.

En la Figura 8.1 puedes ver la jerarquía de clases relacionadas con los controladores.

8.1 Listas de dispositivos

Para cada grupo de controladores tenemos sendas listas, **`DispositivosCaracter`** para los controladores de tipo carácter y **`DispositivosBloque`** para los controladores de tipo bloque. Cada una de estas clases tiene un vector (**`dispositivos`**) cuyas entradas apuntan a los controladores de dispositivos. La Figura 8.2 ilustra estas estructuras.

Inicialización

Ambas clases tienen un método **`iniciar`** [1] que es el encargado de "cargar" los controladores de dispositivo en el atributo **`dispositivos.`**

Ver nucleo/es/bloque.cpp y nucleo/es/caracter.cpp

[1] También tienen un método **`terminar`** pero este no hace nada.

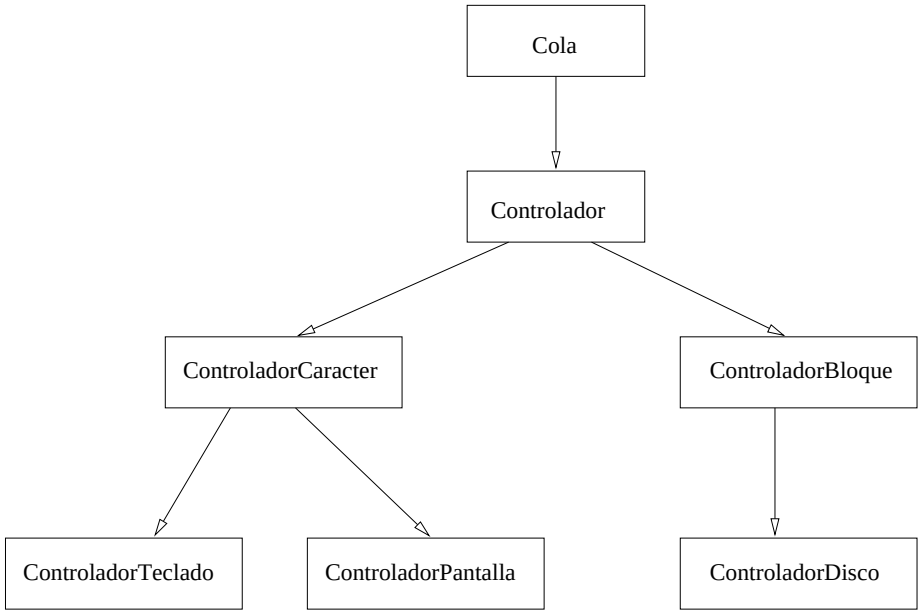

Figura 8.1: Jerarquía de clases para los controladores de dispositivos.

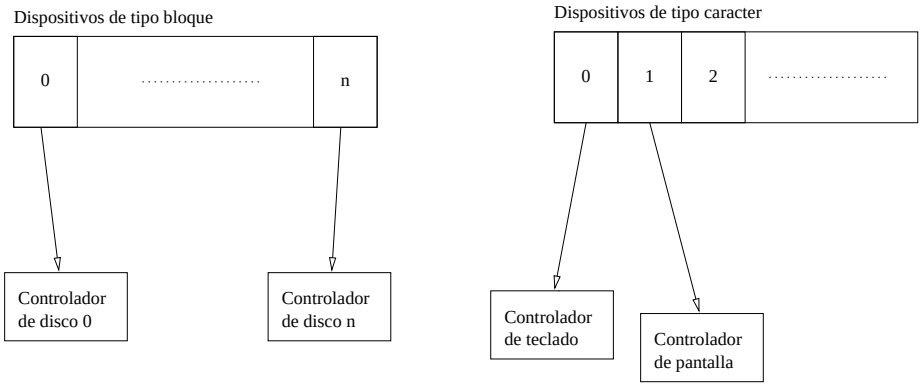

Figura 8.2: Listas de controladores de dispositivos.

Identificación

Una diferencia importante entre los controladores de tipo carácter y los de bloque es la forma en que se identifican. Mientras que los primeros tienen asociado un nombre los segundos se identifican por la posición que ocupan dentro del vector **`dispositivos.`** Esto se puede ver con claridad, además de en el método **`iniciar,`** en el método **`obtenerControlador`** que ambas clases poseen. [2]

8.2 Hilos en estado suspendido

Como puedes ver en la Figura 8.1 la clase base de toda la jerarquía es **`Cola,`** esto significa que cada controlador de dispositivos tiene asociada una cola de hilos, la cual representa a los hilos que, en estado suspendido, están esperando para acceder al dispositivo.

De hecho, la mayor parte de la complejidad en la gestión de las transiciones de estados de los hilos de *Ejecución* a *Suspendido* y de *Suspendido* a *Listo* reside en la clase **`Controlador.`**

Para comprender mejor la forma en que Eafitos simula la entrada/salida, vamos a estudiar qué es lo que sucede desde que un hilo realiza una petición de entrada/salida hasta que esta se satisface.

Las peticiones de entrada/salida nacen siempre con una llamada al sistema y pasan siempre por el sistema de ficheros [3] antes de llegar a uno de los métodos:

- **`ControladorCaracter::leerCaracter`**
- **`ControladorCaracter::escribirCaracter`**
- **`ControladorBloque::leer`**
- **`ControladorBloque::escribir`**

Entonces empieza el trabajo del subsistema de gestión de la entrada/salida. La petición no se satisface inmediatamente sino que provoca la suspensión del hilo que la realizó, esto está implementado en el método **`nuevaPeticion`** de la clase **`Controlador.`** Una vez el hilo está en estado *Suspendido* y esperando en la cola del dispositivo, se procede a registrar la petición que realizó, esto se realiza con los métodos **`Hilo::ponerPeticion`** cuya función es almacenar la información necesaria en el atributo **`Hilo::peticion.`** Una vez realizado esto el control regresa al procesador, el cual seguirá ejecutando instrucciones (de otro hilo que haya pasado a *Ejecución*).

[2] Inicialmente ambos tipos de dispositivos se identificaban por la posición dentro del atributo **`dispositivos.`** Modifiqué el modo de identificación de los controladores de tipo carácter cuando implementé la entrada/salida estándar como ficheros. Así, los controladores de tipo carácter se pueden tratar como ficheros normales, pero no los controladores de tipo bloque.

[3] Esta parte la veremos en el Capítulo 9.

Cada vez que se ejecuta una instrucción se produce, como ya vimos en la Sección 5.3, una llamada al método **`Nucleo::reloj`** el cual a su vez llama a los métodos **`DispositivosCaracter::gestionarColas`** y **`DispositivosBloque::gestionarColas.`** Estos métodos recorren las listas de dispositivos llamando a los métodos **`Controlador::planificador`** de cada controlador de dispositivo. Dicho método decrementa el contador **`Cola::tiempo,`** si llega a cero ejecuta la petición pendiente del hilo que está en la cabeza de la cola (con el método **`Controlador::ejecutarPeticion`**) y lo pasa de *Suspendido* a *Listo*.

8.3 Crear nuevos controladores

Si creas nuevos dispositivos en la máquina virtual tendrás que escribir controladores para ellos, si no no podrás utilizarlos.

Lo primero que debes tener claro es si el dispositivo es de tipo carácter o de tipo bloque. En cualquier caso deberás crear una clase, la cual deberá heredar de **`ControladorCaracter`** o **`ControladorBloque,`** según corresponda.

8.3.1 De tipo carácter

La nueva clase deberá implementar al menos uno de los dos métodos siguientes:

- **`_leerCaracter`** Lee un carácter del dispositivo y lo devuelve.
- **`_escribirCaracter`** Escribe el carácter que se le pasa en el dispositivo.

Finalmente deberás modificar el método **`DispositivosCaracter::iniciar,`** para que el nuevo controlador sea incluido en la lista de controladores de tipo carácter.

8.3.2 De tipo bloque

La nueva clase deberá implementar al menos uno de los dos métodos siguientes:

- **`_leer`** Lee el bloque indicado del dispositivo.
- **`_escribir`** Escribe el bloque indicado en el dispositivo.

También deberás escribir un constructor que se utilizará para iniciar el controlador.

Finalmente deberás modificar el método **`iniciar`** y/o **`iniDisco`** de la clase **`DispositivosBloque`** , para que el nuevo controlador sea incluido en la lista de controladores de tipo bloque.

Capítulo 9

El núcleo: Sistema de ficheros

Lo normal en la mayoría de sistemas operativos, o al menos en los sistemas operativos más utilizados, es que estos tan solo soporten un sistema de ficheros propio y en todo caso algún otro heredado de versiones antiguas del mismo producto. Otros sistemas operativos, sin embargo, tienen soporte para un montón de sistemas de ficheros.

En principio Eafitos solo entiende un sistema de ficheros, pero dispone de los mecanismos necesarios para crear fácilmente nuevos sistemas de ficheros e incorporarlos al núcleo. La Figura 9.1 muestra un esquema global del sistema de ficheros.

9.1 Relación con los procesos

Una de las posibilidades que ofrecen los procesos a los hilos (ver Capítulo 7) es el acceso al sistema de ficheros a través de su atributo **`ficherosLocales`** (ver Sección 7.2), en el dibujo se puede ver que cada entrada de la tabla de ficheros locales apunta a un fichero de la tabla de ficheros globales (**`SFV::ficheros`**).

Ver include/flocal.h

Cada entrada de dicha tabla es una instancia de la clase **`FicheroLocal,`** es a través de esta clase que se accede al sistema de ficheros. Pero, como ya vimos en el Capítulo 7, un fichero local puede apuntar a un controlador de dispositivo de tipo carácter en lugar de a un fichero normal (o no apuntar a ningún sitio, si el fichero no está abierto). El atributo **`FicheroLocal::tipo`** nos indica a qué referencia el fichero:

- **`LIBRE`** El fichero local está libre.
- **`FICHERO`** Referencia a un fichero global del sistema de ficheros virtual. El

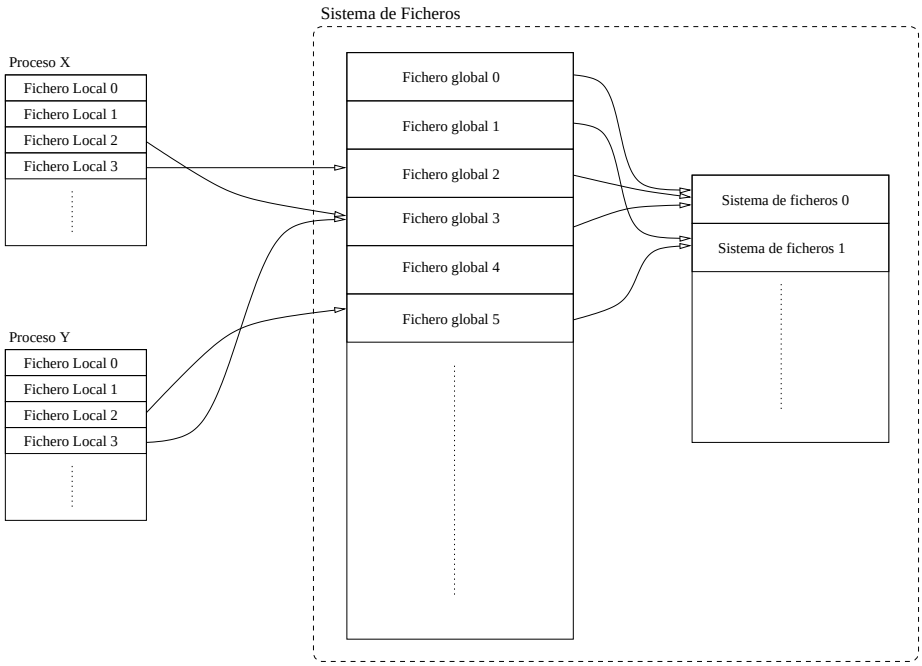

Figura 9.1: Esquema del sistema de ficheros

atributo **id** identifica al fichero global [1] y el atributo **posicion** indica la posición dentro del fichero en la que realizaremos el próximo acceso.

- **DISPOSITIVO** Referencia a un controlador de dispositivo de tipo carácter. El atributo **controlador** apunta al controlador de tipo carácter.

Para acceder a estos recursos, la clase **FicheroLocal** dispone de los métodos **crear, abrir, cerrar, leer, escribir** y **saltar.**

Además de la tabla de ficheros locales los procesos disponen de otros dos atributos relevantes. El atributo **Proceso::unidad** identifica la unidad actual, es un índice dentro de la tabla de sistemas de ficheros; el atributo **Proceso::directorio** identifica el directorio actual, es un índice dentro de la tabla de ficheros globales.

9.2 El sistema de ficheros virtual

La Figura 9.1 muestra dos tablas dentro del sistema de ficheros, ambas tablas forman parte de lo que se llama sistema de ficheros virtual. El sistema de ficheros virtual es un interfaz para los sistemas de ficheros y está implementado en la clase **SFV.**

[1] Los arcos de la Figura 9.1 que van de la tabla de ficheros locales a la de ficheros globales vienen definidos por este atributo, **FicheroLocal::id** .

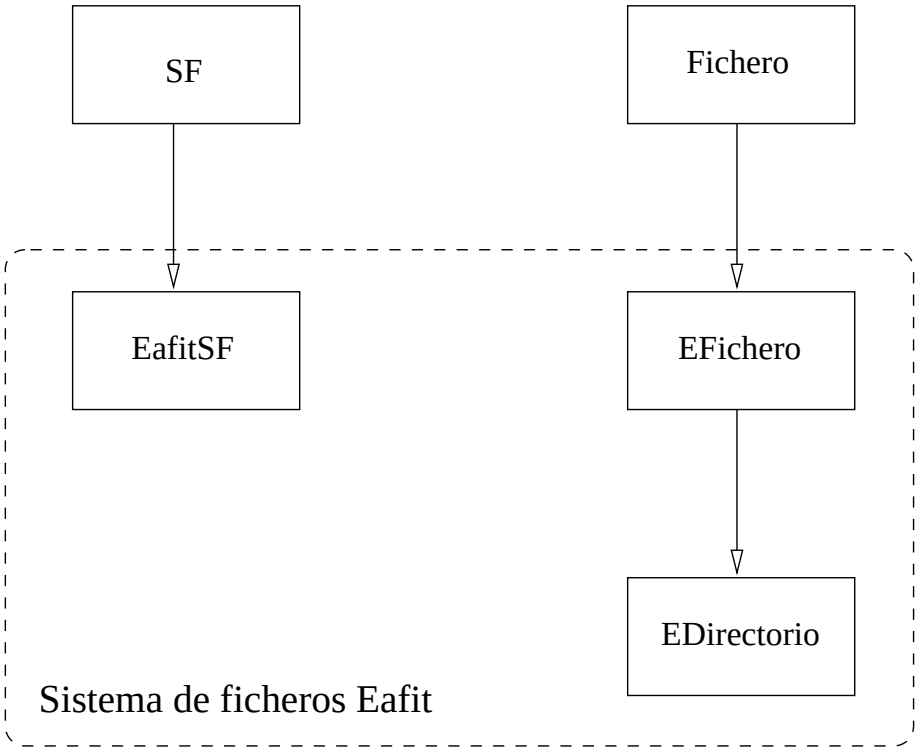

Figura 9.2: Jerarquía de clases para los sistemas de ficheros.

9.2.1 Sistema de ficheros no montado

El sistema de ficheros de Eafitos no está montado [2] (como en DOS y al contrario que UNIX), esto lo simplifica significativamente.

De aquí la necesidad del atributo **`Proceso::unidad,`** que en un sistema de ficheros montado no sería necesario. En el sistema operativo DOS las unidades se referencian mediante letras (C, D, ...), en Eafitos se usan números (0, 1, ...).

9.2.2 Abstracción de los sistemas de ficheros

Igual que los formatos de ficheros ejecutables, los sistemas de ficheros también están abstraídos, esto facilita la creación de nuevos sistemas de ficheros. La Figura 9.2 muestra la jerarquía de clases existente.

[2] Convertir el sistema de ficheros de Eafitos en un sistema montado podría ser una buena práctica.

Creación de nuevos sistemas de ficheros

Para crear un nuevo sistema de ficheros hay que definir e implementar dos clases, una de ellas hereda de **SF** y la otra hereda de **Fichero.** Si vas a emprender esta tarea fíjate en el sistema de ficheros ya existente.

Además de esto lo único que hay que hacer es modificar el método **SFV::iniSF.** Este método se encarga de inicializar un sistema de ficheros partiendo de un controlador de dispositivo de tipo bloque. La inicialización consiste en crear una entrada en la tabla de sistemas de ficheros (**SFV::sf**) y abrir el directorio raíz. El fragmento de código a modificar es:

```
try {
        sf[n] = new EafitSF(cB);
}
catch(...) {
        throw "sistema de ficheros desconocido";
}
```

y el código modificado:

```
try {
        sf[n] = new EafitSF(cB);
}
catch(...) {
        try {
                sf[n] = new NuevoSF(cB);
        }
        catch(...) {
                throw "sistema de ficheros desconocido";
        }
}
```

Donde `NuevoSF` es la clase del nuevo sistema de ficheros que desciende de **SF.** Si el dispositivo controlado por el controlador **cB** no contiene un sistema de ficheros tipo Eafit entonces prueba con el sistema de ficheros Nuevo; de este modo se podrían crear cuantos sistemas de ficheros se quisiera, sin límites.

La clase SF

El atributo **SFV::sf** es un vector de punteros a la clase **SF.** De este modo, cada puntero puede apuntar a una instancia de una clase distinta, siempre y cuando dicha clase herede de **SF.**

El objetivo de esta clase, o mejor dicho, de sus descendientes, es el de tratar directamente con el sistema de ficheros; esta es la clase que entiende cómo esta organizada la información en el disco.

Un sistema de ficheros no es más que un contenedor de ficheros. Dentro de cada sistema de ficheros los ficheros deben quedar identificados unívocamente por un número, y el directorio raíz debe ser siempre el fichero número 0. Por ejemplo, en un sistema de ficheros tipo UNIX (como el que tiene Eafitos) los ficheros se identifican por su inodo, mientras que en el sistema FAT de DOS los ficheros se identifican por el número de su primer bloque; en cualquier caso, siempre se un número.

La clase **SF** tiene varios atributos y métodos que son comunes para cualquier clase que descienda de ella. El atributo **dispositivo** apunta al controlador de dispositivo asociado, el cual proporciona, como ya sabes, un acceso directo por bloques. Los atributos **tamBloque** y **numBloques** contienen el tamaño del bloque y el número de bloques del dispositivo. Como ya vimos anteriormente, cuando se inicializa un sistema de ficheros se abre su directorio raíz; el atributo **dirRaiz** identifica el fichero global que representa al directorio raíz abierto. Además de estos atributos la clase **SF** tiene varios métodos obvios.

Si creas un nuevo sistema de ficheros tendrás que crear una clase que herede de **SF.** Esa nueva clase deberá implementar el método **abrir,** el cual abre el fichero que se le indica (mediante el número que lo identifica). También deberás implementar dos constructores, uno que sirva para inicializar el sistema de ficheros y que tan solo reciba como parámetro el controlador de bloque, y otro que sirva para dar formato a un disco; probablemente también tendrá que escribir un destructor. Y por supuesto tendrás que implementar todo lo que el sistema de ficheros concreto que estas creando requiera.

La clase `Fichero`

De forma similar, el atributo **SFV::ficheros** es un vector de punteros a la clase **Fichero.** Esta clase guarda información sobre cada fichero abierto del sistema. Su atributo **uso** indica cuantos elementos están utilizando este fichero, el atributo **posicion** indica cual es la posición en la que se hará el próximo acceso, el atributo **tamBloque** contiene el tamaño del bloque del sistema de ficheros y **buffer** no es más que un espacio de memoria que usaremos para leer y escribir bloques en el sistema de ficheros.

Además de estos atributos, **Fichero** tiene definidos un conjunto de métodos que deberán ser implementados en las clases descendientes. Los métodos **es, obtenerTamano, finalFichero, abrir, cerrar, borrar, leer, escribir** y **saltar** se deben implementar siempre porque son generales para cualquier fichero. Sin embargo existen otros métodos que son específicos para directorios y por lo tanto solo deberán ser implementados por estos. Estos métodos y su significado se describen a continuación:

- **idFichero** Dentro de un directorio un fichero se identifica unívocamente por su nombre; este método debe devolver, a partir del nombre del fichero, el número que lo identifica dentro del sistema de ficheros.
- **existeFichero** Devuelve cierto si existe y falso si no.
- **crearFichero** Crea y abre el fichero.

- **borrarFichero** Borra el fichero dado, si existe claro.
- **infoFichero** Rellena la estructura **InfoFichero,** esto se utiliza para listar el contenido de un directorio.

9.2.3 Interfaz

Vamos a estudiar ahora con cierto detalle la clase **SFV.** Esta clase tan solo tiene dos atributos que ya hemos visto antes, la tabla de ficheros globales, **ficheros,** y la tabla de sistemas de ficheros, **sf.**

Los métodos **iniciar** y **terminar** son los encargados, como sus nombres indican, de iniciar y terminar el sistema de ficheros. Estos métodos son llamados desde el núcleo. En concreto el método **iniciar** lo que hace es inicializar la tabla de sistemas de ficheros, para cada disco formateado crea inicializa una entrada, para ello se sirve del método **iniSF;** además, para cada sistema de ficheros abre su directorio raíz.

El resto de métodos son usados para acceder a los ficheros e implementan, en parte, las llamadas al sistema equivalentes. Estos métodos son **crear, abrir, cerrar, borrar, leer, escribir, saltar** e **infoFichero;** después de haber leído la sección anterior resulta obvio saber qué es lo que hacen.

Búsqueda de un fichero

Algunos de los métodos de **SFV** (**crear, abrir** y **borrar**) reciben como parámetro, entre otros, el nombre del fichero sobre el que realizar la operación. A partir de este nombre hay que, en primer lugar, obtener el identificador del fichero. El nombre puede ser una ruta absoluta (ejemplo, */bin/fich*) o una ruta relativa al directorio actual (ejemplo, *../fich*). Lo que vamos a ver ahora, sin llegar al detalle, es el algoritmo que permite obtener el identificador del fichero a partir de su nombre.

Antes de seguir adelante te aconsejo que cojas el código y leas lo que sigue comprobando la implementación de alguno de estos métodos. La explicación que viene a continuación se refiere al método **abrir,** la implementación de los otros métodos es muy parecida.

Lo primero que se debe determinar es si la ruta es absoluta o relativa (esto se hace en función del primer carácter del nombre), si la ruta es absoluta se empezará buscando desde el directorio raíz y si es relativa se empezará desde el directorio actual (ambos directorios están abiertos y son conocidos). A continuación nos introducimos en un bucle, con cada iteración del bucle buscamos el siguiente subdirectorio, si el subdirectorio ya está abierto aprovechamos su entrada en la tabla de ficheros globales y si no lo está lo abrimos temporalmente con **SF::abrir;** si el subdirectorio no existe se aborta la operación y se genera un error. La ejecución del bucle termina cuando llegamos al último elemento del nombre del fichero, entonces, si todo ha salido bien habremos obtenido su

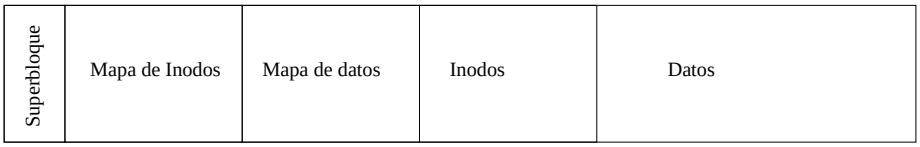

Figura 9.3: Sistema de ficheros Eafit

identificador y podremos abrirlo.

9.3 El sistema de ficheros Eafit

Se trata de un sistema de ficheros tipo UNIX.

9.3.1 Estructura en disco

En la Figura 9.3 puedes ver cómo divide el disco este sistema de ficheros. La clase **`EafitSF,`** que desciende de **`SF,`** es la encargada de gestionar el sistema de ficheros a este nivel.

El superbloque

El primer bloque del disco está ocupado por el superbloque. Este contiene cierta información global empezando por un número que identifica al sistema de ficheros Eafit, este número se escribe cuando se da formato a un disco y se busca al inicializar (si no lo encuentra no inicializa el sistema de ficheros).

Además del número de identificación el superbloque contiene 8 datos más que identifican los números de bloque donde empiezan el resto de zonas del sistema de ficheros (mapas de inodos y de bloques de datos, inodos y datos), y el número de bloques que ocupan. Como puedes ver hay información redundante, esta redundancia evita tener que hacer cálculos al inicializar. El superbloque está definido en la clase **`SuperbloqueESF.`**

Los mapas de inodos y de datos

Los mapas de inodos y de datos nos indican si un inodo (o un bloque de datos) está libre u ocupado. Para hacer esto solamente es necesario un bit pero este sistema de ficheros gasta un byte (para simplificar), algo que podría cambiar en versiones posteriores.

Los inodos

Cada fichero tiene un inodo que recoge cierta información sobre el fichero. Cada inodo ocupa 64 bytes. Los inodos están definidos en la clase **InodoESF,** cuyos campos son:

- **tamano** El tamaño del fichero.
- **tipo** Puede ser **NORMAL** o **DIRECTORIO.**
- **reservado** 27 bytes para uso futuro.
- **bloquesDirectos** Un vector de siete entradas, cada una es un número de bloque, cada uno de los cuales almacena los datos del fichero.
- **bloquesIndirectos** Es el número de un bloque, el cual contiene a su vez más números de bloques donde podremos encontrar más datos del fichero.

Datos

Aquí es donde se almacena el contenido de los ficheros.

9.3.2 Ficheros

Los ficheros están implementados en la clase **EFichero,** básicamente esta clase lo que hace es proporcionar algunos métodos para acceder a los ficheros (leer, escribir,...).

Sin duda lo más complicado de esta parte es la lectura y escritura. Los métodos **leer** y **escribir** hacen uso de los métodos **obtenerBloque** y **reservarBloque,** .

El método **obtenerNBloque** devuelve el número de bloque del sistema de ficheros que corresponde al atributo **poscion,** si la posición está fuera del fichero se genera un error. El método **reservarBloque** es similar a **obtenerNBloque** excepto en que si la posición está fuera del fichero en lugar de generar un error aumenta el tamaño del fichero y si es necesario reserva un nuevo bloque. Como habrás imaginado el primer método se usa para leer y el segundo para escribir.

Directorios

Los directorios están implementados por la clase **EDirectorio,** que desciende de la clase **EFichero** y por lo tanto hereda su funcionalidad.

El contenido de los directorios está dividido en entradas de longitud fija (32 bytes) definidas en la clase **EEntradaDirectorio.** Los campos de cada entrada son:

- **nInodo** Si tiene un valor negativo significa que la entrada está libre, si el valor es positivo identifica al número de inodo del fichero.
- **nombre** El nombre del fichero, máximo 20 caracteres.
- **reservado** Ocho bytes para uso futuro.

Todos los directorios excepto el directorio raíz tienen siempre una entrada ocupada (..), esta entrada hace referencia al directorio padre.

9.4 Recorrido de una llamada al sistema de ficheros

Ya hemos visto todo el sistema de ficheros, esta sección lo que pretende es dar otro enfoque al problema para mejorar la comprensión del sistema. Para ello vamos a ver las clases por las que pasa una llamada al sistema de ficheros. Por ejemplo una llamada para leer uno o más bytes de un fichero ya abierto.

La llamada nace de la instrucción **ser_sis**, cuando el procesador la encuentra realiza la pertinente llamada al núcleo, al método **Nucleo::llamada.** El núcleo recibe la llamada, la decodifica y llama al método que corresponda del proceso actual, en este caso **Proceso::leer.** El proceso repite la llamada al sistema de ficheros virtual, **SFV::leer,** y este llama al fichero, **Fichero::leer.** A partir de aquí la operación se resuelve con una o más llamadas a la clase encargada de gestionar el sistema de ficheros concreto (en nuestro caso **EafitSF**), esto desencadenará a su vez llamadas al controlador de dispoositivo y de ahí al dispositivo. Una vez la operación se haya ejecutado se recorrerá el camino inverso: del fichero al sistema de ficheros virtual, después al proceso, al núcleo y finalmente el control regresa al procesador.

La descripción anterior ha sido realizada sin tener en cuenta que las llamadas al subsistema de entrada/salida provocan, como ya vimos en anteriores capítulos, la suspensión del hilo que la realizó. Por lo tanto la realidad es significativamente más compleja, de todos modos espero que esta descripción haya dado una idea más coherente del sistema de ficheros y de lo que sucede cuando se produce una llamada. Ahora tan solo queda estudiar el código fuente, modificarlo a tu gusto, y si algo no te ha quedado claro... volver a leer esta documentación (o preguntarme a mi ⌣).

Capítulo 10

El entorno

La ejecución comienza en la función **`main`**. Primero inicia la máquina virtual y el núcleo, después cede el control al entorno ejecutando el método **`Entorno::iniciar`** y finalmente termina el núcleo y la máquina virtual.

Ver eafitos2.cpp

El entorno no es más que el sistema de menús que aparece cuando se ejecuta Eafitos, el cual permite crear y formatear discos e iniciar el intérprete de comandos. Está definido e implementado por la clase **`Entorno`** en los ficheros *include/entorno.h* y *entorno/entorno.cpp*, es muy simple, así que no lo voy a comentar.

10.1 El intérprete de comandos

El intérprete de comandos está definido e implementado por la clase. **`InterpreteComandos`**. Consta de un sólo método llamado **`iniciar`**, que es muy sencillo. Básicamente se trata de un bucle infinito, al principio del cual se lee una línea del teclado para analizarla a continuación. En función del comando que haya escrito el usuario se realizará una acción u otra, o ninguna si el comando no está definido. Del bucle se sale cuando el usuario escribe el comando *salir*.

Ver include/ic.h y entorno/ic.cpp

Capítulo 11

El compilador

11.1 Introducción

El compilador es más exactamente un ensamblador, ya que lo que hace es traducir instrucciones del lenguaje ensamblador (ver Sección 2.2) a las instrucciones del código máquina que el procesador entiende. Además, el resultado es un fichero con un formato dado, en concreto el formato 99.

El ensamblador está implementado por la clase **`Ensamblador`** . En las siguientes secciones estudiaremos por separado cada una de las partes de que consta: analizador léxico, analizador sintáctico, analizador semántico y generador de código.

Ver include/ensambla.h

11.1.1 Dos pasadas

El lenguaje permite utilizar un identificador en una instrucción y declararlo en un lugar posterior. Esto es necesario para poder hacer saltos hacia delante, pero complica el compilador. La solución adoptada consiste en dar dos pasadas al código fuente.

En la primera se genera cierta información interna que se utilizará en la segunda. En la segunda pasada se genera el código. En la Figura 11.1 puedes ver un dibujo que muestra la relación entre las distintas partes funcionales y la/s pasada/s en las que intervienen.

En el dibujo, aparecen duplicadas las distintas partes del ensamblador, eso no significa que hayan dos de cada, está dibujado así para ver mejor las pasadas.

Como se puede observar, el analizador léxico genera *tokens* a medida que el analizador sintáctico se los pide. También se ve que en ambas pasadas intervienen los analizadores léxico y sintáctico, los cuales realizan exactamente la misma tarea en ambas ocasiones. Por el contrario, el analizador semántico realiza operaciones distintas en cada pasada, lo veremos con detalle más adelante.

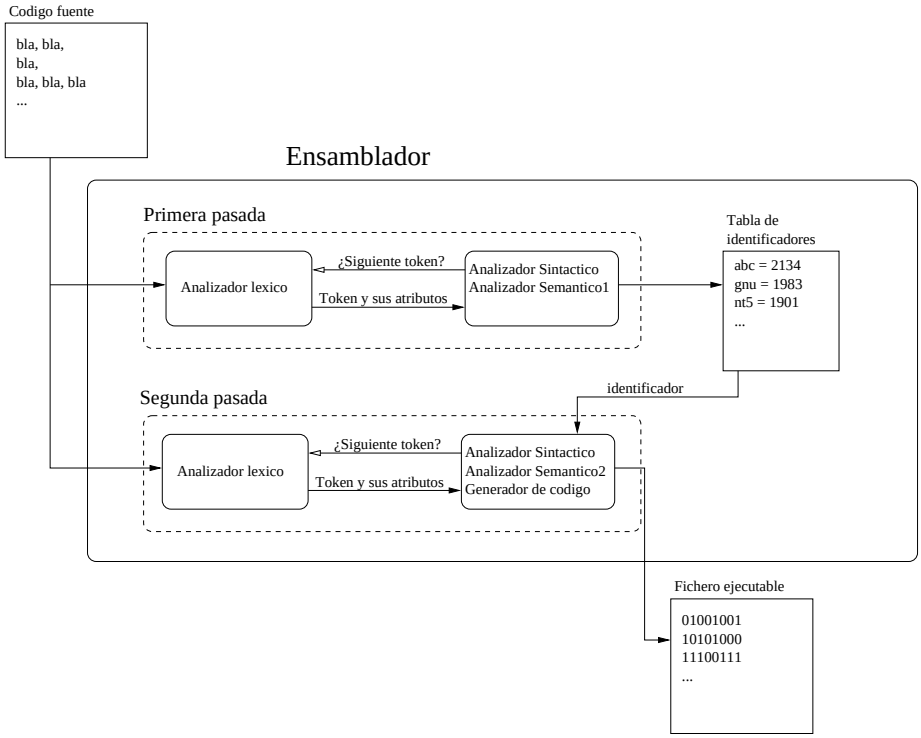

Figura 11.1: Esquema del ensamblador

El generador de código solo interviene en la última pasada, cuando ya está disponible toda la información que necesita. El atributo **pasada** indica en qué pasada se encuentra el análisis.

El análisis empieza en el método público **analizar** , que viene a ser el *main* del compilador. Es recomendable echarle un vistazo al código, ya que, al menos en este caso, está bien comentado.

Ver entorno/ensambla.cpp

11.1.2 Gestión de errores

La gestión de errores de este compilador es extremadamente sencilla. Cuando se detecta un error, da igual de qué tipo sea, se genera el error y se detiene la compilación. Es decir, no existe recuperación de errores, por lo tanto solo se puede detectar uno cada vez. Además, si el error se produjo en la segunda pasada, se borra el código objeto generado.

11.2 Analizador Léxico

11.2.1 Especificación léxica

Blancos

Consideramos blancos al espacio, los tabuladores horizontal y vertical, el retorno de carro y el "form-feed". Dichos caracteres se ignoran, no devuelven ningún token ni tienen ninguna acción asociada. Su expresión regular es:

```
\ \r\t\v\f
```

Comentarios

Al igual que los blancos, los comentarios también se ignoran. Comienzan por un punto y coma y terminan con una nueva línea, su expresión regular es:

```
;.*\ n
```

La nueva línea tan solo indica cuando termina el comentario, no está incluida en la expresión.

Nueva línea

La nueva línea se emplea como un separador en el nivel sintáctico, por ello se devuelve el token **NUEVA_LINEA** . Comienza siempre por una nueva línea y traga todos los blancos, comentarios y nuevas líneas que le sigan, hasta que encuentra un carácter distinto. Su expresión regular es:

lexema	*token*	*valor*	*lexema*	*token*	*valor*
datos	**DATOS**		codigo	**CODIGO**	
sumar	**I_RRR**	**SUMAR**	restar	**I_RRR**	**RESTAR**
and	**I_RRR**	**AND**	or	**I_RRR**	**OR**
copiar	**I_RR**	**COPIAR**	not	**I_RR**	**NOT**
cargar32	**I_RR**	**CARGAR32**	guardar32	**I_RR**	**GUARDAR32**
cargar8	**I_RR**	**CARGAR8**	guardar8	**I_RR**	**GUARDAR8**
cargar_i	**I_RI**	**CARGAR_I**	guardar_i	**I_RI**	**GUARDAR_I**
apilar	**I_R**	**APILAR**	desapilar	**I_R**	**DESAPILAR**
saltar	**I_I**	**SALTAR**	saltar0	**I_RI**	**SALTAR0**
saltarp	**I_RI**	**SALTARP**	saltarn	**I_RI**	**SALTARN**
nop	**I_**	**NOP**	ser_sis	**I_**	**SER_SIS**

Tabla 11.1: Palabras Clave.

```
\n([\ \n\r\t\v\f]|(;.*\n))*
```

Final de fichero

Como ya habrás imaginado, este token (**FINAL_FICHERO**) se emite cuando se termina de leer el fichero.

Palabras clave

La Tabla 11.1 muestra la lista de palabras clave del ensamblador.

Las palabras clave, igual que los identificadores (ver más adelante), no son sensibles a mayúsculas/minúsculas. Es lo mismo escribir `datos` que `DaToS` .

Como se puede observar en la tabla, la mayoría de las palabras claves son instrucciones del lenguaje ensamblador, las cuales se agrupan (mediante el token) según su formato, es decir, según su número y tipo de operandos. Sin operandos, con uno, dos o tres registros, con un dato inmediato o con un registro y un dato inmediato. Dentro de cada grupo el *valor* identifica de qué instrucción se trata; *valor* almacena el código de instrucción que el procesador entiende.

Identificadores

Los identificadores se utilizan para especificar variables y posiciones dentro del código. Cuando se detecta un identificador se emite el token **ID** . Su expresión regular, exceptuando las palabras clave, es:

```
[a-zA-Z][a-zA-Z0-9_]*
```

Literales numéricos

Se utilizan para inicializar las variables y para los datos inmediatos. El token que se emite es **LITERAL_NUMERICO** . Su expresión regular es:

```
#[0-9]+
```

En el atributo *valor* se almacena el valor numérico del literal.

Literales cadena

Se utilizan para inicializar las variables. El token que se emite es **CADENA** . Su expresión regular es:

```
".*"
```

En el atributo *lexema* se almacena la cadena, sin las comillas.

Registros

Se utilizan para especificar los operandos de tipo registro. El token que se emite es **REGISTRO** . Su expresión regular es:

```
@[0-1][0-5]?
```

En el atributo *valor* se almacena el número del registro.

Coma

Se utiliza para separar los operandos. El token que se emite es **COMA** . Su expresión regular es:

```
,
```

11.2.2 Autómata finito determinista

Lo encontrarás en la Figura 11.2.

11.2.3 Cuestiones de implementación

El tipo enumerado **Tokens** define los tokens que existen. El método **siguienteToken** implementa el analizador léxico. Este método devuelve un

Ver include/ensambla.h

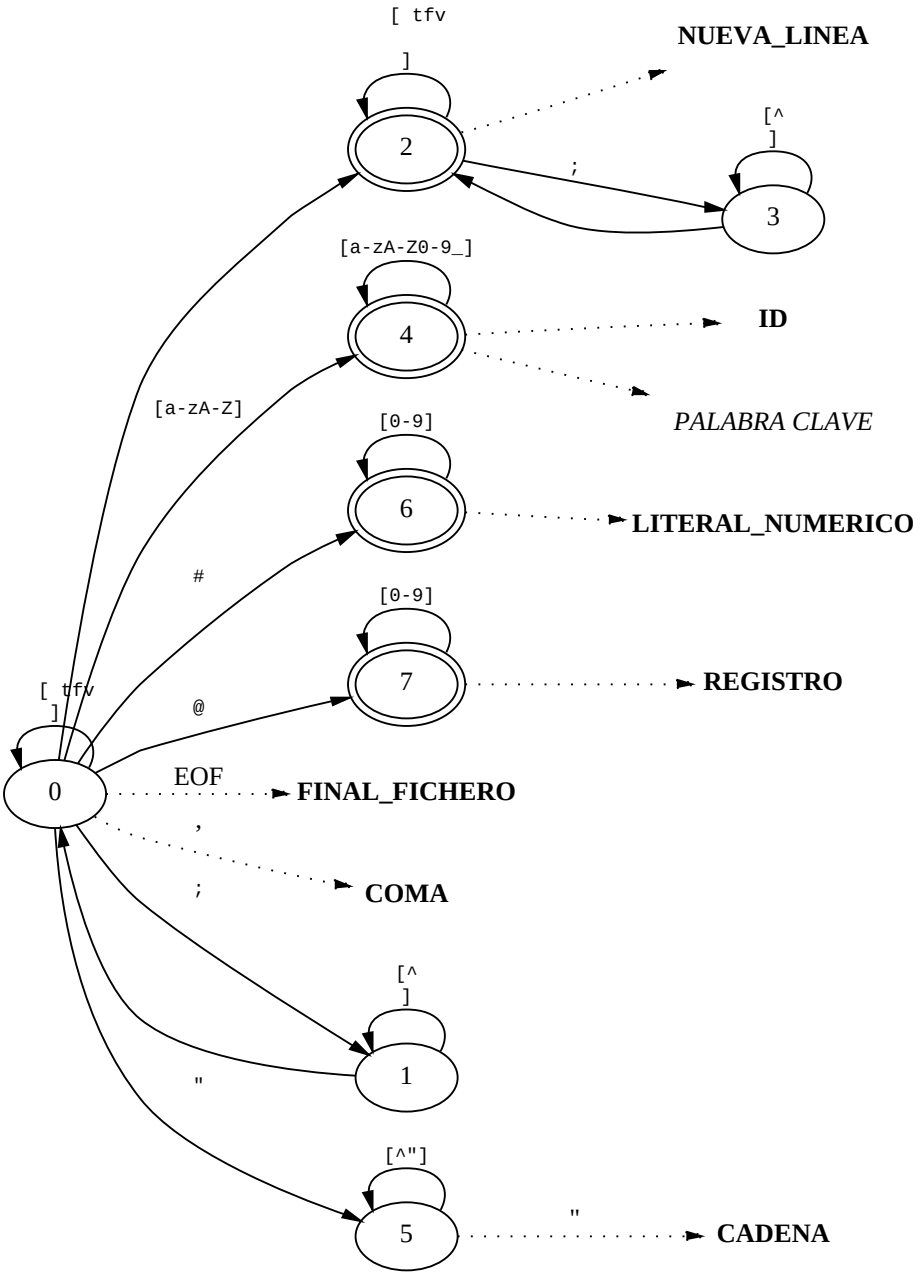

Figura 11.2: Autómata Finito Determinista

token cada vez que se le llama. Además, el último token leído se guarda en el atributo **token** , su lexema en **lexema** y, dependiendo del token, **valor** puede almacenar más información acerca del token.

Para estudiar este método lo mejor es leer el código con el dibujo del AFD (Figura 11.2) al lado, ya que lo único que hace es implementar dicho autómata. Básicamente es un bucle infinito del que se sale cuando se encuentra un token. Los arcos del autómata quedan representados por estructuras condicionales, primero se lee un carácter del código fuente y dependiendo de qué carácter se trate se pasa a analizar un subgrafo u otro. Mejor dale un vistazo al código.

Palabras reservadas

Una palabra reservada se define mediante la clase **PalabraClave** . Cada palabra reservada tiene un *token* asociado, un lexema (que debe estar siempre en mayúsculas) y opcionalmente un valor. La variable **palabrasClave** es un vector con una entrada para cada palabra reservada. La constante **N_PALABRAS_CLAVE** indica el número de palabras reservadas que existen.

Ver entorno/ensambla.cpp

Número de línea

Además, el método **siguienteToken** lleve la cuenta del número de línea que se está analizando. El número de línea actual se guarda en el atributo **linea** . Así, cuando se produzca un error, sabremos en qué línea del código fuente se produjo.

11.3 Analizador sintáctico

11.3.1 Especificación sintáctica

En la Tabla 11.2 encontrarás la especificación sintáctica. Los símbolos terminales están en negrita y empiezan por minúscula, se corresponden con los *tokens* del nivel léxico (**nl** es una abreviatura de **nueva_linea**). Los símbolos no terminales empiezan por mayúscula.

Como puedes observar, esta no es una gramática LL(1). El lenguaje es tan sencillo que resulta fácil de implementar directamente la gramática incontextual.

11.3.2 Cuestiones de implementación

Para cada símbolo no terminal existe un método, con el mismo nombre, que lo implementa. Básicamente, lo que hacen estos métodos es recorrer la parte derecha de la producción.

Si a continuación hay un símbolo terminal, se lee el siguiente token y si es el

Programa	→	**nl datos nl** Datos **codigo nl** Codigo **final_fichero** |
		datos nl Datos **codigo nl** Codigo **final_fichero** |
		nl codigo nl Codigo **final_fichero** |
		codigo nl Codigo **final_fichero**
Datos	→	LineaDatos **nl** Datos |
		λ
LineaDatos	→	**id** Constante
Constante	→	**cadena** | **literal_numérico**
Codigo	→	LineaCodigo **nl** Codigo |
		LineaCodigo **nl** |
		LineaCodigo
LineaCodigo	→	**id** Instruccion |
		id |
		Instruccion
Instrucción	→	**i_** |
		i_r registro |
		i_rr registro coma registro |
		i_rrr registro coma registro coma registro |
		i_i literal_numérico |
		i_i id |
		i_ri registro coma literal_numérico
		i_ri registro coma id

Tabla 11.2: Especificación sintáctica

esperado se sigue con el análisis, si es otro se produce un error. Si a continuación hay un símbolo no terminal se llama al método que lo implementa.

Como es una gramática incontextual puede suceder que en un momento dado haya más de una posibilidad (por ejemplo, dos símbolos no terminales distintos), entonces se lee un token, y dependiendo de qué token se trate se hace una cosa u otra.

Estos métodos, además de implementar el analizador sintáctico, implementan el analizador semántico y el generador de código.

11.4 Analizador semántico

11.4.1 Especificación semántica

Las comprobaciones semánticas que realiza el compilador son:

1. No se puede declarar un identificador más de una vez.

2. No puede referenciarse ningún identificador que no esté declarado; es indiferente que la declaración sea anterior o posterior a su referencia.

No se distingue entre identificadores que representan variables o posiciones dentro del código.

11.4.2 Cuestiones de implementación

El analizador semántico interviene en las dos pasadas que da el ensamblador al código fuente, pero hace una cosa distinta en cada una de ellas.

En la primera pasada recoge información acerca de los identificadores y comprueba la regla semántica número 1. Dicha información se guarda en el vector **ids** , cuyas entradas son instancias de la clase **Id** . En concreto, la información que se almacena es el lexema del identificador y la dirección de memoria que referencia. El atributo **direccion** indica la dirección actual de memoria, este atributo es incrementado por el analizador semántico cada vez que encuentra una instrucción o la declaración de una variable; se utiliza para saber qué dirección asociar a los identificadores.

Ver include/ensambla.h

El método **nuevoId** es el que se encarga de guardar los identificadores en el vector y de incrementar el atributo **nIds** , el cual guarda el número de identificadores almacenados; también es este método el que realiza la comprobación semántica número 1.

En la segunda pasada se comprueba la regla semántica número 2. Dicha comprobación la realiza el método **obtenerDir** , el cual devuelve la dirección asociada a un identificador, o genera un error si el identificador no existe. Este método es utilizado por el generador de código.

11.5 Generador de código

La generación de código es muy sencilla ya que se trata de un lenguaje ensamblador.

Cabecera

El código generado sigue el formato de fichero ejecutable 99, eso significa que lo primero que se debe escribir en el fichero es la cabecera. Esa información está formada por el "número mágico" que autentifica al fichero, la dirección de inicio del programa, contenida en el atributo **dirInicio** , y el tamaño de la pila que es siempre de 128.

Área de datos

Después de la cabecera comienza la imagen que se cargará en memoria. Esta empieza por el área de datos, los cuales son inicializados por el generador de código.

Código

Y después de los datos, el código. Cada instrucción en ensamblador se traduce directamente a código máquina. Al final se añaden siempre tres instrucciones que lo que hacen es terminar la ejecución del hilo, esas instrucciones en ensamblador son:

```
cargar_i @0, #3
apilar @0
ser_sis
```

Parte III

Apéndices

Apéndice A

Código fuente

A.1 Ficheros de cabecera

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/mv.h
//    Definición de la máquina virtual.
// ----------------------------------------------------------------------

#ifndef _MAQUINA_VIRTUAL
#define _MAQUINA_VIRTUAL

#include <memfis.h>
#include <mmu.h>
#include <cpu.h>
#include <teclado.h>
#include <pantalla.h>
#include <disco.h>


#define NUM_DISCOS 5


class MaqVirtual {
public:
      static MemFis memoria;
      static MMU mmu;
      static CPU cpu;
      static Teclado teclado;
      static Pantalla pantalla;
private:
      static Disco* discos[NUM_DISCOS];
```

```
public:
      // Método para crear nuevos discos
      static void crearDisco(char id, long tamBloque, long numBloques);

      // Devuelve una referencia al disco especificado
      static Disco& disco(long id);

      static void iniciar();
      static void terminar();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/memfis.h
//    Definición de la memoria física de la maquina virtual.
// ----------------------------------------------------------------------


#ifndef _MEMORIA_FISICA
#define _MEMORIA_FISICA


#define TAM_MEMORIA 8192


class MemFis {
      char memoria[TAM_MEMORIA];
public:
      void escribirByte(char dato, long direccion);
      void escribirPalabra(long dato, long direccion);
      char leerByte(long direccion);
      long leerPalabra(long direccion);
};

#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/mmu.h
//    Definición de la unidad de gestión de memoria
// ----------------------------------------------------------------------

#ifndef _MMU
#define _MMU


#include <memfis.h>

// Relacionado con la gestión de la memoria
#define NUM_SEGMENTOS 20
#define PAGS_POR_SEGMENTO 5
#define TAM_PAGINA 1024
#define NUM_PAGINAS TAM_MEMORIA/TAM_PAGINA


// Clase que define un segmento como un conjunto de páginas.
// Un valor de -1 en paginas[x] significa que la página x está libre.
class Segmento {
      long uso;
      long paginas[PAGS_POR_SEGMENTO];
public:
      Segmento();

      long libre();
      long reservar(long nPaginas);
      long reservar();
      long liberar();

      long paginaFisica(long paginaLogica);
};


// Unidad de gestión de memoria.
// Obtiene direcciones físicas a partir de direcciones lógicas.
// Reserva y libera memoria.
class MMU {
      Segmento segmentos[NUM_SEGMENTOS];

      // Para cada pagina indicamos si esta ocupada (>0) o no (=0)
      char paginas[NUM_PAGINAS];
      long nPaginasLibres;
public:
      MMU();

      // Traduce una direccion logica a una direccion fisica
      long logicaAFisica(long segmento, long dirLogica);
```

```
      // Reserva/libera una página
      long reservarPagina();
      void liberarPagina(long pagina);

      // Reserva/libera memoria
      long asignar(long memoria);
      long reservar(long segmento);
      long liberar(long segmento);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/contexto.h
//    Definición del contexto de ejecución de un hilo.
// ----------------------------------------------------------------------

#ifndef _CONTEXTO
#define _CONTEXTO


#define NUM_REGISTROS 16


// La clase Contexto define lo que se conoce como el contexto de ejecución de
// un hilo, es decir, los registros basicos de la CPU que diferencian a un
// hilo de otro.
class Contexto {
public:
      long pc;                // Contador de programa
      long sp;                // Puntero de pila

      long registros[NUM_REGISTROS];      // registros generales

      long codigo;      // Segemento de código y datos estáticos
      long pila;  // Segmento de pila
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/cpu.h
//    Definición de la unidad central de proceso (CPU).
// ----------------------------------------------------------------------

#ifndef _CPU
#define _CPU


#include <contexto.h>


// Juego de instrucciones
enum Instrucciones {SUMAR, RESTAR, AND, OR, COPIAR, NOT,
      CARGAR_I, CARGAR32, CARGAR8, GUARDAR_I, GUARDAR32, GUARDAR8,
      APILAR, DESAPILAR,
      SALTAR, SALTARO, SALTARP, SALTARN, NOP, SER_SIS};


class CPU {
      Contexto contexto;
public:
      Contexto& obtenerContexto()
            { return contexto; }

      void ponerContexto(Contexto& c)
            { contexto = c; }

      // Métodos de Lectura/Escritura en memoria
      void escribirByte(long segmento, long direccion, char dato);
      void escribirPalabra(long segmento, long direccion, long dato);
      char leerByte(long segmento, long direccion);
      long leerPalabra(long segmento, long direccion);

      // Métodos de lectura de operandos e instrucciones y de acceso a pila
      char leerOpByte();
      long leerOpPalabra();
      void apilar(long dato);
      long desapilar();

      void ejecutarPaso();    // Ejecuta una instrucción
      void ejecutar();  // Ejecuta instrucciones hasta que se acaban
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/teclado.h
//    Definición del teclado.
// ----------------------------------------------------------------------

#ifndef _TECLADO
#define _TECLADO


class Teclado {
public:
      char leerCaracter();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ------------------------------------------------------------------
//    include/pantalla.h
//    Definición de la pantalla.
// ------------------------------------------------------------------

#ifndef _PANTALLA
#define _PANTALLA


class Pantalla {
public:
      void escribirCaracter(char caracter);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/disco.h
//    Definición del controlador del disco.
// ----------------------------------------------------------------------

#ifndef _DISCO
#define _DISCO


#include <stdio.h>

// Tamaño de bloque + numero de bloques
#define CABECERA sizeof(long)+sizeof(long)


class Disco {
      FILE* disco;

      long tamBloque;
      long numBloques;

      // A partir del número del disco devuelve su nombre completo
      //   "\EAFITOS\DISCO_n" o "$HOME/EAFITOS/DISCO_n"
      char* obtenerNombre(char id);
public:
      // Constructor para inicializar un disco
      Disco(char id);
      // Constructor para formatear un disco
      Disco(char id, long tBloque, long nBloques);
        ~Disco();

      long obtenerTamBloque();
      long obtenerNumBloques();

      // Métodos de acceso al disco, leen/ecriben un bloque.
      long leer(char *buffer, long bloque);
      long escribir(char *buffer, long bloque);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/nucleo.h
//    Definición del núcleo.
// ----------------------------------------------------------------------

#ifndef _NUCLEO
#define _NUCLEO


#include <memoria.h>
#include <procesos.h>
#include <caracter.h>
#include <bloque.h>
#include <sf.h>


// Definición de llamadas al sistema
#define CREAR_HILO 1          // De gestión de hilos y procesos
#define TERMINAR_HILO 2
#define TERMINAR 3
#define CAMBIAR_UNIDAD 10     // Del sistema de ficheros
#define CAMBIAR_DIRECTORIO 11
#define CREAR_FICHERO 12
#define ABRIR_FICHERO 13
#define CERRAR_FICHERO 14
#define BORRAR_FICHERO 15
#define LEER_FICHERO 16
#define ESCRIBIR_FICHERO 17
#define SALTAR_FICHERO 18
#define EJECUTAR 20           // Para ejecutar un programa
#define LEER_ENTRADA_ESTANDAR 30    // De E/S estandar
#define ESCRIBIR_SALIDA_ESTANDAR 31


class Nucleo {
public:
      static ManejadorMemoria manejadorMemoria;
      static ManejadorProcesos manejadorProcesos;
      static DispositivosCaracter dispositivosCaracter;
      static DispositivosBloque dispositivosBloque;
      static SFV sfv;

      // Clases utilizadas como excepciones en el método reloj, sirven para
      //   informar a la CPU.
      class NoHayHilos { };         // Ya no quedan hilos, la CPU debe
                            //   dejar de ejecutar instrucciones
                            //   y devolver el control.
      class NoHayHilosListos { };   // Quedan hilos pero todos están en
                            //   SUSPENDIDO, la CPU no debe
                            //   ejecutar ninguna instrucción
```

```
                                //   este ciclo.

      static void iniciar();
      static void terminar();

      // Método equivalente a las rutinas de servicio de la interrupción
      //   de reloj que tienen los sistemas reales.
      static void reloj();
      // Método utilizado como interfaz para las llamadas al sistema.
      static void llamada();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/memoria.h
//    Definición del manejador de memoria del núcleo.
// ----------------------------------------------------------------------

#ifndef _MANEJADOR_MEMORIA
#define _MANEJADOR_MEMORIA


class ManejadorMemoria {
public:
      void iniciar() {}
      void terminar() {}

      // Métodos para reservar y liberar memoria
      long asignar(long cantidad);
      long reservar(long segmento);
      long liberar(long segmento);

      // Métodos para copiar datos de memoria de núcleo a memoria de usuario
      //   y viceversa
      void aUsuario(long segmento, long dtno, char* fte, long nBytes);
      void deUsuario(long segmento, long fte, char* dtno, long nBytes);
};

#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/hilo.h
//    Definición de un hilo.
// ----------------------------------------------------------------------

#ifndef _HILO
#define _HILO


#include <contexto.h>
#include <peticion.h>


class Hilo {
      // Posibles estados de un hilo
      enum Estado {LIBRE, LISTO, EJECUCION, SUSPENDIDO};
      Estado estado;

      // Para guardar el contexto de ejecución del hilo cuando este no se
      //   encuentre en ejecución
      Contexto contexto;

      // Proceso asociado a este hilo
      class Proceso* proceso;

      // Guarda la petición de entrada/salida del hilo
      Peticion peticion;

      // Punteros a otros hilos, para las colas de hilos
      long siguiente, anterior;
public:
      Hilo();

      // Devuelve el proceso asociado a este hilo.
      Proceso* obtenerProceso();

      // Métodos para gestionar las poner/obtener las peticiones de E/S
      Peticion& obtenerPeticion();
      void ponerPeticion(Peticion::Tipo codigo);
      void ponerPeticion(Peticion::Tipo codigo, long dato);
      void ponerPeticion(Peticion::Tipo codigo, char* datos);
      void ponerPeticion(Peticion::Tipo codigo, long dato, char* datos);

      // Métodos para gestionar la cola (atributos anterior y siguiente)
      void setAnt(long ant)   { anterior = ant; }
      void setSig(long sig)   { siguiente = sig; }
      long getAnt()           { return anterior; }
      long getSig()           { return siguiente; }
```

```
      // Métodos de información sobre el estado
      char libre();
      char listo();

      // Métodos para los cambios de estado
      void iniciar(Proceso* proc, long direccion);    // Para la llamada al
                                          //  sistema CREAR_HILO.
      void iniciar(Proceso* proc, Contexto& ctx);     // Para la ejecución de
                                          //  programas.
      void ejecutar();
      void dormir();
      void suspender();
      void reactivar(long resultado);
      void terminar();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/proceso.h
//    Definición de un proceso.
// ----------------------------------------------------------------------


#ifndef _PROCESO
#define _PROCESO


#include <flocal.h>

#define N_F_LOCALES 10


class Proceso {
      long uso;

      FicheroLocal ficherosLocales[N_F_LOCALES];
      long unidad;
      long directorio;
public:
      Proceso();

      // Indica si este proceso existe o no.
      long libre();

      // Inicia y termina este proceso.
      long iniciar(long u = 0, long d = 0);
      long terminar();

      // Métodos que implementan en parte diversas llamadas al sistema.
      //   Para cambiar la unidad y el directorio actuales
      void cambiarUnidad(long num);
      void cambiarDirectorio(char* nombre);

      //   Del sistema de ficheros
      long crear(char* nombre, long tipo);
      long abrir(char* nombre);
      void cerrar(long id);
      long leer(long id, void* destino, long nBytes);
      long escribir(long id, void* fuente, long nBytes);
      long saltar(long id, long posicion, long desde);
      void borrar(char* nombre);

      //   Para ejecutar programas
      void ejecutar(char* nombre);

      //   Operaciones sobre la entrada/salida estándar
      long leerCaracter();
```

```
      void imprimirCaracter(char c);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/cola.h
//    Definición de las colas de hilos
// ----------------------------------------------------------------------

#ifndef _COLA
#define _COLA


#include <hilo.h>

#define N_HILOS 20

class Cola {
protected:
      static Hilo hilos[N_HILOS];
      long primero;
      long tiempo;
public:
      Cola();

      long actual();
      long insertar(long hilo);
      long extraer(long hilo = -1);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/procesos.h
//    Definición del manejador de hilos y procesos.
// ----------------------------------------------------------------------

#ifndef _PROCESOS
#define _PROCESOS


#define N_PROCESOS 10
#define N_EJECUTABLES 5
#define TAJADA 100


#include <cola.h>
#include <proceso.h>
#include <ejec.h>


class ManejadorProcesos : public Cola {
      // Tabla de procesos
      Proceso procesos[N_PROCESOS];

      // Tabla de formatos de ficheros ejecutables
      Ejecutable* ejecutables[N_EJECUTABLES];
public:
      void iniciar();
      void terminar();

      // Devuelve el proceso actual
      Proceso* procesoEjecucion();

      // Metodos que implementan las correspondientes llamadas al sistema.
      long crearHilo(long direccion);
      void terminarHilo(long id=-1);

      // Método que ejecuta un programa.
      long ejecutar(long unidad, long pwd, char* nombre);

      // Metodo encargado de la politica de paso de un hilo de listo a
      // ejecucion y viceversa.
      void planificador();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/peticion.h
//    Definición de una petición de entrada/salida.
// ----------------------------------------------------------------------

#ifndef _PETICION_ES
#define _PETICION_ES


class Peticion {
public:
      // Distintos tipos de peticiones posibles
      enum Tipo {LEER_CARACTER, ESCRIBIR_CARACTER, LEER_BLOQUE,
            ESCRIBIR_BLOQUE};

      Tipo codigo;

      // Atributos que guardan información sobre la petición
      long dato;
      char* datos;
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/controla.h
//    Definición de un controlador de dispostivo.
// ----------------------------------------------------------------------

#ifndef _CONTROLADOR_DISPOSITIVO
#define _CONTROLADOR_DISPOSITIVO


#include <peticion.h>
#include <cola.h>


class Controlador : public Cola {
protected:
      // Método utilizado para registrar una petición y pasar el hilo a
      //   suspendido.
      long nuevaPeticion();

      // Una vez pasado un tiempo se llama a este método para resolver
      //   la petición.
      virtual long ejecutarPeticion() = 0;
public:
      // Método principal, encargado de gestionar la cola de hilos
      void planificador();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/ccar.h
//    Definición del controlador de tipo caracter.
// ----------------------------------------------------------------------

#ifndef _CONTROLADOR_DISPOSITIVO_CARACTER
#define _CONTROLADOR_DISPOSITIVO_CARACTER

#include <controla.h>


class ControladorCaracter : public Controlador {
      // Método encargado de resolver las peticiones de entrada/salida
      long ejecutarPeticion();
protected:
      // Identificador del controlador
      char* nombre;
public:
      ControladorCaracter(char* n);
      virtual ~ControladorCaracter();

      // Método que nos dice si este controlador es el buscado
      long es(char* n);

      // Métodos utilizados para realizar las peticiones de entrada/salida
      char leerCaracter();
      void escribirCaracter(char caracter);

      // Métodos que ejecutan las peticiones
      virtual char _leerCaracter()
            { throw "función no implementada"; }

      virtual void _escribirCaracter(char caracter)
            { throw "función no implementada"; }
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/cteclado.h
//    Definición del controlador del teclado.
// ----------------------------------------------------------------------

#ifndef _CONTROLADOR_TECLADO
#define _CONTROLADOR_TECLADO

#include <ccar.h>


class ControladorTeclado : public ControladorCaracter {
public:
      ControladorTeclado(char* n) : ControladorCaracter(n) {}

      char _leerCaracter();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/cpantall.h
//    Definición del controlador de la pantalla.
// ----------------------------------------------------------------------

#ifndef _CONTROLADOR_PANTALLA
#define _CONTROLADOR_PANTALLA

#include <ccar.h>


class ControladorPantalla : public ControladorCaracter {
public:
      ControladorPantalla(char* n) : ControladorCaracter(n) {}

      void _escribirCaracter(char caracter);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/cbloque.h
//    Definición del controlador de tipo bloque.
// ----------------------------------------------------------------------

#ifndef _CONTROLADOR_DISPOSITIVO_BLOQUE
#define _CONTROLADOR_DISPOSITIVO_BLOQUE

#include <controla.h>


class ControladorBloque : public Controlador {
      // Método encargado de resolver las peticiones de entrada/salida
      long ejecutarPeticion();

protected:
      long tamBloque;
      long numBloques;
public:
      long obtenerTamBloque()       { return tamBloque; }
      long obtenerNumBloques()      { return numBloques; }

      // Métodos para realizar las peticiones de entrada/salida
      long leer(char *datos, long bloque);
      long escribir(char *datos, long bloque);

      // Métodos que ejecutan las peticiones
      virtual long _leer(char *datos, long bloque) = 0;
      virtual long _escribir(char *datos, long bloque) = 0;
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/cdisco.h
//    Definición del controlador del disco.
// ----------------------------------------------------------------------

#ifndef _CONTROLADOR_DISCO
#define _CONTROLADOR_DISCO

#include <cbloque.h>
#include <disco.h>


class ControladorDisco : public ControladorBloque {
      Disco* disco;
public:
      ControladorDisco(Disco& d);

      long _leer(char *buffer, long bloque);
      long _escribir(char *buffer, long bloque);
};



#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/caracter.h
//    Definición de la lista de controladores de tipo caracter.
// ----------------------------------------------------------------------

#ifndef _DISPOSITIVOS_CARACTER
#define _DISPOSITIVOS_CARACTER

#include <ccar.h>


#define NUM_DISPOS_CARACTER 5


// Tabla de dispositivos de tipo caracter
class DispositivosCaracter {
      // Tabla de controladores de tipo carácter
      ControladorCaracter* dispositivos[NUM_DISPOS_CARACTER];
public:
      void iniciar();
      void terminar() {}

      // Devuelve el controlador con el identificador especificado
      ControladorCaracter* obtenerControlador(char* nombre);

      // Método encargado de gestionar las colas de hilos asociadas a
      //   los controladores de tipo carácter cada vez que se produce
      //   una interrupción de reloj
      long gestionarColas();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/bloque.h
//    Definición de la lista de controladores de tipo bloque.
// ----------------------------------------------------------------------

#ifndef _DISPOSITIVOS_BLOQUE
#define _DISPOSITIVOS_BLOQUE


#include <cbloque.h>


#define NUM_DISPOS_BLOQUE 10


// Tabla de dispositivos de tipo bloque
class DispositivosBloque {
      // Tabla de controladores de tipo bloque
      ControladorBloque* dispositivos[NUM_DISPOS_BLOQUE];
public:
      void iniciar();
      void terminar() {}

      // Inicializa el controlador con el identificador especificado
      void iniDisco(long id);

      // Devuelve el controlador con el identificador especificado
      ControladorBloque* obtenerControlador(long id);

      // Método encargado de gestionar las colas de hilos asociadas a
      //   los controladores de tipo bloque cada vez que se produce
      //   una interrupción de reloj
      long gestionarColas();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/sf.h
//    Definición del sistema de ficheros.
// ----------------------------------------------------------------------

#ifndef _SISTEMA_DE_FICHEROS
#define _SISTEMA_DE_FICHEROS


#include <cbloque.h>

// Constantes para el Sistema de Ficheros Virtual
#define MAX_FICHEROS_ABTOS 100
#define MAX_NUM_SF 10

// Tipos de ficheros
#define NORMAL 0
#define DIRECTORIO 1

// Constantes para el método saltar de la clase Fichero
#define PRINCIPIO 0
#define ACTUAL 1


// Clase utilizada como abstracción de la información de directorio
class InfoFichero {
public:
      char nombre[200];
      long tipo;
      long tamano;
};


// Clase base para los sistemas de ficheros reales
class SF {
protected:
      // Controlador de tipo bloque asociado a este sistema de ficheros
      ControladorBloque* dispositivo;

      long tamBloque;
      long numBloques;

      // Guarda el identificador del directorio raiz en el sistema de
      //   ficheros virtual.
      long dirRaiz;
public:
      SF(ControladorBloque* d) {
            dispositivo = d;
            tamBloque = dispositivo->obtenerTamBloque();
            numBloques = dispositivo->obtenerNumBloques();
```

```
            dirRaiz = -1;
      }
      virtual ~SF() {}

      long obtenerTamBloque()       { return tamBloque; }
      long obtenerNumBloques()      { return numBloques; }

      long obtenerDirRaiz()         { return dirRaiz; }
      void ponerDirRaiz(long dir)   { dirRaiz = dir; }

      // Métodos que todos los sistemas de ficheros deben implementar.
      virtual class Fichero* abrir(long idFichero) = 0;
};


// Clase base para los ficheros de los sistemas de ficheros reales
class Fichero {
protected:
      long uso;   // Número de elementos que estan usando este fichero.
      long posicion;  // Apunta al próximo caracter a leer.

      long tamBloque;   // Tamaño del bloque del sistema de ficheros
      char* buffer;     // Buffer para leer y escribir bloques.
public:
      Fichero(SF* sf) {
            uso = 1;
            posicion = 0;

            tamBloque = sf->obtenerTamBloque();
            buffer = new char[tamBloque];
      }

      virtual ~Fichero() {
            if (buffer)
                  delete buffer;
      }

      // Métodos generales que todos los ficheros deben implementar
      virtual long es(SF* sf, long idFichero) = 0;
      virtual long obtenerTamano() = 0;
      virtual long finalFichero() = 0;

      virtual long abrir() = 0;
      virtual long cerrar() = 0;
      virtual void borrar() = 0;
      virtual long leer(char* dtno, long nDatos) = 0;
      virtual long escribir(char* fte, long nDatos) = 0;
      virtual long saltar(long pos, long desde) = 0;

      // Métodos que solo los directorios deben implementar
      //   Devuelve el identificador del fichero (n? de inodo, por ejemplo)
      virtual long idFichero(char* nombre)
            { throw "función no implementada"; }
```

```
      //   Devuelve cierto o falso según el fichero exista o no
      virtual long existeFichero(char* nombre)
            { throw "función no implementada"; }

      //   Crea el fichero indicado
      virtual Fichero* crearFichero(char* nombre, char tipo)
            { throw "función no implementada"; }

      //   Borra el fichero indicado
      virtual void borrarFichero(char* nombre)
            { throw "función no implementada"; }

      //   Devuelve información de un fichero del directorio
      virtual long infoFichero(InfoFichero& info)
            { throw "función no implementada"; }
};


// Sistema de ficheros virtual
class SFV {
      // Tabla de sistemas de ficheros reales
      SF* sf[MAX_NUM_SF];

      // Tabla de ficheros abiertos en el sistema
      Fichero* ficheros[MAX_FICHEROS_ABTOS];

      // Método que busca el fichero indicado en la tabla de ficheros
      long buscar(long unidad, long nInodo);
public:
      void iniciar();
      void terminar();

      // Inicializa un sistema de ficheros
      void iniSF(ControladorBloque* cB, long n);

      // Devuelve cierto o falso según si el sistema de ficheros está o no
      //   inicializado
      long activo(long id);

      long obtenerTamano(long id);

      // Métodos para las llamadas al sistema
      long crear(long unidad, long pwd, char *nombre, long tipo);
      long abrir(long unidad, long pwd, char *nombre);
      long abrir(long id);
      long cerrar(long id);
      void borrar(long unidad, long pwd, char* nombre);
      long leer(long id, void *destino, long numBytes);
      long escribir(long id, void *fuente, long numBytes);
      long saltar(long id, long posicion, long desde);

      // Devuelve información sobre el fichero indicado
      long infoFichero(long id, InfoFichero& info);
};
```

```
#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/flocal.h
//    Definición de un fichero local. Todos los procesos tienen
//    varios.
// ----------------------------------------------------------------------

#ifndef _FICHERO_LOCAL
#define _FICHERO_LOCAL


// Clase que define los ficheros locales de los procesos, es decir, las
//   entradas de las tablas de ficheros asociadas a los procesos.
class FicheroLocal {
      // Indica si se trata de un fichero normal o de un dispositivo o si
      //   la entrada está libre
      enum Tipo {LIBRE, FICHERO, DISPOSITIVO};
      Tipo tipo;

      // Si es de tipo dispositivo, este atributo apunta a su controlador
      class ControladorCaracter* controlador;

      // Si es un ficero normal...
      long id;    // Identifica al fichero global asociado
      long posicion;    // La posición dentro del fichero
public:
      FicheroLocal();

      // Para saber si la entrada
      long libre();

      // Métodos para las llamadas al sistema
      long crear(long unidad, long pwd, char* nombre, long tipo);
      long abrir(long unidad, long pwd, char* nombre);
      void cerrar();
      long leer(void* destino, long nBytes);
      long escribir(void* fuente, long nBytes);
      long saltar(long pos, long desde);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/eafitsf.h
//    Definición del sistema de ficheros EAFIT.
// ----------------------------------------------------------------------


#ifndef _SF_EAFIT
#define _SF_EAFIT

#include <sf.h>


// Posibles estados de las entradas de los mapas de datos e inodos.
#define _LIBRE 0
#define _OCUPADO 1

// Numero de bloques directos por inodo.
#define NBDIRECTOS 7

// Maxima longitud del nombre que puede tener un directorio.
#define LONG_NOMBRE 20

// Número mágico
#define EAFITSF_ID 1500

// El superbloque se encuentra en el primer bloque del disco (bloque 0).
// Se pueden añadir más campos al final, siempre y cuando el superbloque no
// sobrepase el tamaño de un bloque.
class SuperbloqueESF {
public:
      long id;    // Número mágico

      long primerBloqueMD;    // MD: Mapa de bits de los bloques de Datos
      long numBloquesMD;
      long primerBloqueMI;    // MI: Mapa de bits de los Inodos
      long numBloquesMI;
      long primerBloqueI;     // I: Inodos
      long numBloquesI;
      long primerBloqueD;     // D: bloques de Datos
      long numBloquesD;
};



// Cada fichero viene representado por un inodo.
// Los inodos del sistema de ficheros Eafit ocupan 64 bytes.
// Los primeros contienen información varia sobre el fichero
// A continuación se reserva un espacio para uso futuro.
// Los ultimos 32 bytes almacenan las direcciones de los bloques de datos.
class InodoESF {
```

```
public:
      long tamano;
      char tipo;

      char reservado[27];

      long bloquesDirectos[NBDIRECTOS];
      long bloquesIndirectos;
};


// Clase que conoce el formato del Sistema de Ficheros Eafit.
class EafitSF : public SF {
      SuperbloqueESF superbloque;

      char* mapaDatos;
      char* mapaInodos;

      // Lee y escribe los mapas de bits del sistema de ficheros
      void leerMapaDatos();
      void escribirMapaDatos();
      void leerMapaInodos();
      void escribirMapaInodos();
public:
      // Constructores:
      //   carga un sistema de ficheros ya creado
      EafitSF(ControladorBloque* d);
      //   da formato a un disco
      EafitSF(ControladorBloque* d, long numInodos);
      ~EafitSF();

      // Abre el fichero con el identificador dado
      Fichero* abrir(long idFichero = 0);

      // Reserva y libera inodos y bloques
      long reservarInodo();
      void liberarInodo(long nInodo);
      long reservarBloque();
      void liberarBloque(long nBloque);

      // Lee y escribe inodos del disco
      InodoESF& leerInodo(long num);
      void escribirInodo(InodoESF& inodo, long n);

      // Lee y escribe bloques del disco
      long leerBloque(long nBloque, char* buffer);
      long escribirBloque(long nBloque, char* buffer);
};


// Fichero del Sistema de Ficheros Eafit.
class EFichero : public Fichero {
protected:
      EafitSF* sf;      // Sistema de Ficheros Eafit que contiene este fichero.
```

```
      long nInodo;      // Numero de Inodo.

      InodoESF inodo;          // Guarda en memoria una copia del inodo.
      long* bloquesIndirectos; // Indices a los bloques indirectos.

      // Devuelve el núm. de bloque actual, si es final de fichero genera
      //   un error
      long obtenerNBloque();
      // Devuelve el núm. de bloque actual, si es final de fichero reserva
      //   un byte.
      long reservarBloque();
public:
      EFichero(EafitSF* sf, long nInodo, InodoESF& inodo);
      ~EFichero();

      // Cierto o falso según este fichero sea o no el buscado
      long es(SF* sf, long idFichero);

      // Devuelve el tamaño del fichero
      long obtenerTamano();

      // Cierto o falso según estemos o no en el final del bloque
      long finalFichero();

      // Operaciones varias
      long abrir();
      long cerrar();
      virtual void borrar();
      long leer(char* dtno, long nDatos);
      long escribir(char* fte, long nDatos);
      long saltar(long pos, long desde);
};



// Una entrada de directorio del Sistema de Ficheros Eafit.
class EEntradaDirectorio {
public:
      long nInodo;
      char nombre[LONG_NOMBRE];
      char reservado[8];
};


// Un fichero de tipo directorio del Sistema de Ficheros Eafit.
class EDirectorio : public EFichero {
      // Crea y borra entradas de deirectorio
      void crearEntrada(char* nombre, long nInodo);
      void borrarEntrada(char* nombre);
public:
      EDirectorio(EafitSF* sf, long nInodo, InodoESF& inodo) :
            EFichero(sf, nInodo, inodo) {}

      void borrar();
```

```
      long idFichero(char* nombre);
      long existeFichero(char* nombre);
      Fichero* crearFichero(char* nombre, char tipo);
      void borrarFichero(char* nombre);
      long infoFichero(InfoFichero& info);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/ejec.h
//    Definición de la clase base Ejecutable
// ----------------------------------------------------------------------

#ifndef _EJECUTABLE
#define _EJECUTABLE

#include <contexto.h>

// Clase base para los formatos de ficheros ejecutables
class Ejecutable {
public:
      // Método para cargar un fichero ejecutable
      virtual void ejecutar(long fichero, Contexto& contexto) = 0;
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/ejec99.h
//    Definición del formato ejecutable 99
// ----------------------------------------------------------------------

#ifndef _EJECUTABLE_99
#define _EJECUTABLE_99

#include <ejec.h>

// Número mágico, identifica al fichero
#define ID_EJECUTABLE_99 2323

// Formato de fichero ejecutable de Eafitos2.
// Cabecera:
//    long id
//    long inicioPrograma
//    long tamPila
class Ejecutable99 : public Ejecutable {
public:
      // Carga un fichero ejecutable con el formato arriba indicado
      void ejecutar(long fichero, Contexto& contexto);
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    entorno.h
//    Este fichero tene la definición de Eafitos.
// ----------------------------------------------------------------------


#ifndef _ENTORNO
#define _ENTORNO


#include <ic.h>
#include <ensambla.h>


// Clase que contiene al intérprete de comandos, al compilador y que implementa
//   los menús de gestión de discos.
class Entorno {
public:
      static InterpreteComandos interpreteComandos;
      static Ensamblador ensamblador;

      static void crearDiscoNuevo();
      static void formatearDisco();
      static void iniciar();
};


#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/ic.h
//    Definición del longerprete de comandos.
// ----------------------------------------------------------------------

#ifndef _INTERPRETE_COMANDOS
#define _INTERPRETE_COMANDOS


class InterpreteComandos {
      char linea[100];  // Almacena la linea que se lee del teclado

      long unidad;            // Identifica la unidad de disco actual
      long directorio;  // Identifica el directorio actual
public:
      void iniciar();
};



#endif
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    include/ensambla.h
//    Definición del ensamlador.
// ----------------------------------------------------------------------

#ifndef _ENSAMBLADOR
#define _ENSAMBLADOR


// Posibles tokens
enum Tokens {DATOS, CODIGO, FINAL_FICHERO, NUEVA_LINEA, ID, COMA,
      REGISTRO, LITERAL_NUMERICO, CADENA,
      I_, I_R, I_RR, I_RRR, I_I, I_RI};

/* Explicación del significado de cada tipo de instrucción
 I_   : sin operandos: especiales (NOP, SER_SIS)
 I_R  : un registro: de gestión de la pila
 I_RR : dos registros: operaciones de ALU unarias y acceso a memoria
 I_RRR      : tres registros: operaciones aritmético/lógicas
 I_I  : un dato inmediato: salto incondicional
 I_RI : un registro y un dato inmediato: acceso a memoria y salto condicional
*/


// Define cada palabra clave
class PalabraClave {
public:
      char* lexema;
      long token;
      long valor;
};

// Aqui guardamos el lexema de un identificador y su dirección de memoria.
class Id {
public:
      char lexema[20];
      long direccion;
};


class Ensamblador {
      // Ficheros fuente y objeto
      long ffte;
      long fobj;

      // Número de línea en el código fuente que se está analizando
      long linea;

      // Variables que guardan información sobre el token actual
      char lexema[100];
```

```
      long token;
      long valor;

      // Función que implementa el analizador léxico. Lee el siguiente token.
      long siguienteToken();

      long pasada;      // Indica si estamos en la 1? o en la 2? pasada
      long direccion;   // Dirección actual dentro del código objeto

      // Información que se recogera en la 1? pasada y se usara en la 2?
      Id ids[100];
      long nIds;
      long dirInicio;

      // Funciones para gestionar la cola de identificadores.
      void nuevoId();
      long obtenerDir(char* nombre);

      // Funciones que implementan el analizador sintáctico, semántico y
      //   generador de código.
      // Una función para cada símbolo no terminal.
      void programa();
      void datos();
      void lineaDatos();
      void constante();
      void codigo();
      void lineaCodigo();
      void instruccion();
public:
      void analizar(char* fuente, char* objeto, long unidad, long directorio);
};


#endif
```

A.2 Maquina virtual

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/mv.cpp
//    Implementación de la máquina virtual.
// ----------------------------------------------------------------------

#include <mv.h>


MemFis MaqVirtual::memoria;
MMU MaqVirtual::mmu;
CPU MaqVirtual::cpu;
Teclado MaqVirtual::teclado;
Pantalla MaqVirtual::pantalla;
Disco* MaqVirtual::discos[NUM_DISCOS];


void MaqVirtual::crearDisco(char id, long tamBloque, long numBloques) {
      if (id<0 || id>NUM_DISCOS)
            throw "número de disco no válido";

      if (discos[id])
            throw "ese disco ya existe";

      discos[id] = new Disco(id+'0', tamBloque, numBloques);
}

Disco& MaqVirtual::disco(long id) {
      if (id<0 || id>=NUM_DISCOS)
            throw "ese disco no existe";

      if (discos[id]==NULL)
            throw "ese disco no existe";

      return *(discos[id]);
}


void MaqVirtual::iniciar() {
      // Inicializar los discos
      for (long i=0; i<NUM_DISCOS; i++) {
            discos[i] = NULL;
            try {
                  discos[i] = new Disco(i+'0');
            }
            catch(...) {}
      }
}
```

```
void MaqVirtual::terminar() {
      // Limpiar la tabla de discos
      for (long i=0; i<5; i++)
            if (discos[i])
                  delete discos[i];
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/memfis.cpp
//    Implementación de la memoria física.
// ----------------------------------------------------------------------

#include <memfis.h>


void MemFis::escribirByte(char dato, long direccion) {
      if (direccion<0 || direccion>=TAM_MEMORIA)
            throw "dirección fuera de rango";

      memoria[direccion] = dato;
}

void MemFis::escribirPalabra(long dato, long direccion) {
      if (direccion<0 || direccion>=TAM_MEMORIA-3)
            throw "dirección fuera de rango";

      *((long*)(memoria + direccion)) = dato;
}

char MemFis::leerByte(long direccion) {
      if (direccion<0 || direccion>=TAM_MEMORIA)
            throw "dirección fuera de rango";

      return memoria[direccion];
}

long MemFis::leerPalabra(long direccion) {
      if (direccion<0 || direccion>=TAM_MEMORIA-3)
            throw "dirección fuera de rango";

      return *((long*)(memoria + direccion));
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/mmu.cpp
//    Implementación de la unidad de gestión de memoria.
// ----------------------------------------------------------------------

#include <mmu.h>
#include <mv.h>


// Constructor, pone el segmento y todas las páginas en libre.
Segmento::Segmento() {
      uso = 0;

      // Inicializar las páginas a libre
      for (long i=0; i<PAGS_POR_SEGMENTO; i++)
            paginas[i] = -1;
}


long Segmento::libre() {
      return !uso;
}

// Reserva memoria
long Segmento::reservar(long nPaginas) {
      if (uso)
            throw "ese segmento ya está ocupado";

      // Reserva las páginas necesarias
      long paginaLogica=0;
      for (; nPaginas>0; nPaginas--) {
            long paginaFisica = MaqVirtual::mmu.reservarPagina();
            paginas[paginaLogica] = paginaFisica;
            paginaLogica++;
      }
      uso = 1;

      return uso;
}

// Incrementa el uso del segmento
long Segmento::reservar() {
      if (!uso)
            throw "ese segmento está libre";
      uso++;
      return uso;
}

// Liberar el segmento
long Segmento::liberar() {
```

```
      if (!uso)
            throw "ese segmento ya está libre";

      // Si ya no utiliza nadie este segmento, liberar las páginas que tenia
      //   ocupadas
      uso--;
      if (uso==0)
            for (long i=0; i<PAGS_POR_SEGMENTO; i++)
                  if (paginas[i]>=0) {
                        MaqVirtual::mmu.liberarPagina(paginas[i]);
                        paginas[i] = -1;
                  }

      return uso;
}

// Obtiene la página física asociada a la pagina lógica dada
long Segmento::paginaFisica(long paginaLogica) {
      if (paginaLogica<0 || paginaLogica>=PAGS_POR_SEGMENTO)
            throw "número de página fuera de rango";

      return paginas[paginaLogica];
}



MMU::MMU() {
      // Poner todas las páginas físicas como libres
      for (long i=0; i<NUM_PAGINAS; i++)
            paginas[i] = 0;

      nPaginasLibres = NUM_PAGINAS;
}

// Traduce una dirección física a una dirección lógica
long MMU::logicaAFisica (long segmento, long dirLogica) {
      long paginaLogica, paginaFisica, desplazamiento;

      // Calcula la página física
      paginaLogica = dirLogica/TAM_PAGINA;
      paginaFisica = segmentos[segmento].paginaFisica(paginaLogica);

      // Fallo de página
      if (paginaFisica<0)
            throw "fallo de página";

      // Calcula el desplazamiento
      desplazamiento = dirLogica%TAM_PAGINA;

      // Calcula y devuelve la dirección física
      return (paginaFisica*TAM_PAGINA) + desplazamiento;
}

// Reserva una página física
```

```
long MMU::reservarPagina() {
      // Busca y reserva una página física, devuelve su identificador
      for (long i=0; i<NUM_PAGINAS; i++)
            if (paginas[i]==0) {
                  paginas[i] = 1;
                  nPaginasLibres--;
                  return i;
            }

      throw "todas las páginas están ocupadas";
}

// Libera la página física indicada
void MMU::liberarPagina(long pagina) {
      if (paginas[pagina]==0)
            throw "esa página ya está libre";

      paginas[pagina] = 0;
      nPaginasLibres++;
}

// Reserva un segmento y la cantidad de memoria solicitada, devuelve el
//   selector del segmento
long MMU::asignar(long cantidad) {
      // Calcula el número de páginas necesarias
      long nPaginas = cantidad/TAM_PAGINA;
      if (cantidad%TAM_PAGINA)
            nPaginas++;

      // Comprueba que haya suficiente memoria
      if (nPaginas>nPaginasLibres)
            throw "no hay suficente memoria";

      // Busca un segmento libre
      for (long i=0; i<NUM_SEGMENTOS; i++)
            if (segmentos[i].libre()) {
                  segmentos[i].reservar(nPaginas);
                  return i;
            }

      throw "todos los segmentos están ocupados";
}

// Incrementa el uso de un segmento ya reservado
long MMU::reservar(long segmento) {
      return segmentos[segmento].reservar();
}

// Libera un segmento
long MMU::liberar(long segmento) {
      return segmentos[segmento].liberar();
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/cpu.cpp
//    Implementación de la unidad central de proceso.
// ----------------------------------------------------------------------

#include <cpu.h>
#include <mv.h>
#include <nucleo.h>


// Escribir un byte en memoria
void CPU::escribirByte(long segmento, long direccion, char dato) {
      long dirFisica = MaqVirtual::mmu.logicaAFisica(segmento, direccion);
      MaqVirtual::memoria.escribirByte(dato, dirFisica);
}

// Escribir una palabra en memoria
void CPU::escribirPalabra(long segmento, long direccion, long dato) {
      long dirFisica = MaqVirtual::mmu.logicaAFisica(segmento, direccion);
      MaqVirtual::memoria.escribirPalabra(dato, dirFisica);
}

// Leer un byte de memoria
char CPU::leerByte(long segmento, long direccion) {
      long dirFisica = MaqVirtual::mmu.logicaAFisica(segmento, direccion);
      return MaqVirtual::memoria.leerByte(dirFisica);
}

// Leer una palabra de memoria
long CPU::leerPalabra(long segmento, long direccion) {
      long dirFisica = MaqVirtual::mmu.logicaAFisica(segmento, direccion);
      return MaqVirtual::memoria.leerPalabra(dirFisica);
}

// Leer un operando de tamaño de un byte
char CPU::leerOpByte() {
      return leerByte(contexto.codigo, contexto.pc++);
}

// Leer un operando de tamaño de una palabra
long CPU::leerOpPalabra() {
      long dato = leerByte(contexto.codigo, contexto.pc);
      contexto.pc += 4;
      return dato;
}

// Apilar un dato en la pila
void CPU::apilar(long dato) {
      escribirPalabra(contexto.pila, contexto.sp, dato);
      contexto.sp+=4;
```

```
}

// Desapilar un dato de la pila
long CPU::desapilar() {
      contexto.sp-=4;
      return leerPalabra(contexto.pila, contexto.sp);
}

// Ejecutar una instrucción
void CPU::ejecutarPaso() {
      // Simula una interrupción de reloj. Le cede el control al núcleo.
      Nucleo::reloj();

      // Adquisición de la instrucción
      char instruccion = leerOpByte();

      // Decodificación, adquisición de los operandos y ejecución
      switch (instruccion) {
            long fte1, fte2, destino;
            case SUMAR:
                  fte1 = leerOpByte();
                  fte2 = leerOpByte();
                  destino = leerOpByte();
                  contexto.registros[destino] = contexto.registros[fte1]
                                    + contexto.registros[fte2];
                  break;
            case RESTAR:
                  fte1 = leerOpByte();
                  fte2 = leerOpByte();
                  destino = leerOpByte();
                  contexto.registros[destino] = contexto.registros[fte1]
                                    - contexto.registros[fte2];
                  break;
            case AND:
                  fte1 = leerOpByte();
                  fte2 = leerOpByte();
                  destino = leerOpByte();
                  contexto.registros[destino] = contexto.registros[fte1]
                                    & contexto.registros[fte2];
                  break;
            case OR:
                  fte1 = leerOpByte();
                  fte2 = leerOpByte();
                  destino = leerOpByte();
                  contexto.registros[destino] = contexto.registros[fte1]
                                    | contexto.registros[fte2];
                  break;
            case COPIAR:
                  fte1 = leerOpByte();
                  destino = leerOpByte();
                  contexto.registros[destino] = contexto.registros[fte1];
                  break;
            case NOT:
                  fte1 = leerOpByte();
```

```
      destino = leerOpByte();
      contexto.registros[destino] = ~contexto.registros[fte1];
      break;
case CARGAR32:
      destino = leerOpByte();
      fte1 = leerOpByte();
      fte1 = leerPalabra(contexto.codigo,
                         contexto.registros[fte1]);
      contexto.registros[destino] = fte1;
      break;
case GUARDAR32:
      fte1 = leerOpByte();
      destino = leerOpByte();
      escribirPalabra(contexto.codigo,
                  contexto.registros[destino],
                  contexto.registros[fte1]);
      break;
case CARGAR8:
      destino = leerOpByte();
      fte1 = leerOpByte();
      fte1 = leerByte(contexto.codigo,
                  contexto.registros[fte1]);
      contexto.registros[destino] = fte1;
      break;
case GUARDAR8:
      fte1 = leerOpByte();
      destino = leerOpByte();
      escribirByte(contexto.codigo,
                  contexto.registros[destino],
                  contexto.registros[fte1]);
      break;
case CARGAR_I:
      destino = leerOpByte();
      fte1 = leerOpPalabra();
      contexto.registros[destino] = fte1;
      break;
case GUARDAR_I:
      fte1 = leerOpByte();
      destino = leerOpPalabra();
      escribirPalabra(contexto.codigo, destino,
                  contexto.registros[fte1]);
      break;
case APILAR:
      fte1 = leerOpByte();
      apilar(contexto.registros[fte1]);
      break;
case DESAPILAR:
      destino = leerOpByte();
      contexto.registros[destino] = desapilar();
      break;
case SALTAR:
      destino = leerOpPalabra();
      contexto.pc = destino;
      break;
```

```
            case SALTAR0:
                  fte1 = leerOpByte();
                  destino = leerOpPalabra();
                  if (contexto.registros[fte1]==0)
                        contexto.pc = destino;
                  break;
            case SALTARP:
                  fte1 = leerOpByte();
                  destino = leerOpPalabra();
                  if (contexto.registros[fte1]>0)
                        contexto.pc = destino;
                  break;
            case SALTARN:
                  fte1 = leerOpByte();
                  destino = leerOpPalabra();
                  if (contexto.registros[fte1]<0)
                        contexto.pc = destino;
                  break;
            case NOP:
                  break;
            case SER_SIS:
                  Nucleo::llamada();
                  break;
            default:
                  throw "instrucción desconocida";
      }
}

// Ejecutar instrucciones mientras queden hilos en ejecución
void CPU::ejecutar() {
      while (1) {
            try {
                  ejecutarPaso();
            }
            catch(Nucleo::NoHayHilosListos) {}
            catch(Nucleo::NoHayHilos) { return; }
            catch(char* c) {
                  Nucleo::manejadorProcesos.terminarHilo();
            }
      }
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/teclado.cpp
//    Implementación del teclado virtual.
// ----------------------------------------------------------------------

#include <curses.h>
#include <teclado.h>

char Teclado::leerCaracter() {
      return getch();
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/pantalla.cpp
//    Implementación de la pantalla virtual.
// ----------------------------------------------------------------------

#include <curses.h>
#include <pantalla.h>

void Pantalla::escribirCaracter(char caracter) {
      addch(caracter);
      refresh();
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    mv/disco.cpp
//    Implementación de un disco virtual.
// ----------------------------------------------------------------------

#include <disco.h>

#include <stdio.h>
#include <string.h>
#ifdef _LINUX_
      #include <stdlib.h>
#endif

// A partir del número del disco devuelve su nombre completo
char* Disco::obtenerNombre(char id) {
      static char nombre[50];

      // Si el SO anfitrión es GNU/Linux, "//EAFITOS/DISCO_n"
      #ifdef _LINUX_
            strcpy(nombre, getenv("HOME"));
            strcat(nombre, "/EAFITOS/DISCO_");
      #endif

      // Si el SO anfitrión es DOS, "\EAFITOS\DISCO_n"
      #ifdef _DOS_
            strcpy(nombre, "\\EAFITOS\\DISCO_");
      #endif

      nombre[strlen(nombre)] = id;
      nombre[strlen(nombre)+1] = '\0';

      return nombre;
}

// Constructor utilizado para inicializar los discos
Disco::Disco(char id) {
      char* nombre;
      nombre = obtenerNombre(id);

      disco = fopen(nombre, "r+b");
      if (disco == NULL)
            throw "ese disco no existe";

      // Lee el tamaño y el número de bloques del disco
      fread(&tamBloque,sizeof(tamBloque),1,disco);
      fread(&numBloques,sizeof(numBloques),1,disco);
}

// Constructor utilizado para crear los discos
Disco::Disco(char id, long tBloque, long nBloques) {
```

```
	tamBloque = tBloque;
	numBloques = nBloques;

	char* nombre;
	nombre = obtenerNombre(id);

	// Crea el fichero
	disco = fopen(nombre, "wb");
	if (disco==NULL)
		throw "no se pudo crear el disco";

	// Escribe el tamaño y el número de bloques del disco
	  fwrite(&tamBloque,sizeof(tamBloque),1,disco);
	fwrite(&numBloques,sizeof(numBloques),1,disco);

	// Escribe espacios en el fichero hasta alcanzar el tamaño deseado
	for (long i=0; i<tamBloque*numBloques; i++)
		fprintf(disco, " ");

	freopen(nombre, "r+b", disco);
}

Disco::~Disco() {
	if (disco)
		fclose(disco);
}


long Disco::obtenerTamBloque() {
	return tamBloque;
}

long Disco::obtenerNumBloques() {
	return numBloques;
}

// Leer un bloque
long Disco::leer(char *datos, long bloque) {
	if (bloque<0 || bloque >= numBloques)
		throw "número de bloque no válido";

	fseek(disco,CABECERA+bloque*tamBloque,SEEK_SET);
	return fread(datos,tamBloque,1,disco);
}

// Escribir un bloque
long Disco::escribir(char *datos, long bloque) {
	if (bloque<0 || bloque >= numBloques)
		throw "número de bloque no válido";

	fseek(disco,CABECERA+bloque*tamBloque,SEEK_SET);
	return fwrite(datos,tamBloque,1,disco);
}
```

A.3 Nucleo

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/nucleo.cpp
//    Implementación del núcleo del SO.
// ----------------------------------------------------------------------

#include <nucleo.h>
#include <mv.h>


ManejadorMemoria Nucleo::manejadorMemoria;
ManejadorProcesos Nucleo::manejadorProcesos;
DispositivosCaracter Nucleo::dispositivosCaracter;
DispositivosBloque Nucleo::dispositivosBloque;
SFV Nucleo::sfv;

// "Arranca" el núcleo
void Nucleo::iniciar() {
      manejadorMemoria.iniciar();
      manejadorProcesos.iniciar();
      dispositivosCaracter.iniciar();
      dispositivosBloque.iniciar();
      sfv.iniciar();
}

// Termina el núcleo
void Nucleo::terminar() {
      sfv.terminar();
      dispositivosBloque.terminar();
      dispositivosCaracter.terminar();
      manejadorProcesos.terminar();
      manejadorMemoria.terminar();
}

// Rutina de servicio de la interrupción de reloj que genera la CPU
void Nucleo::reloj() {
      // Gestionar las colas de controladores
        long r1 = dispositivosCaracter.gestionarColas();
        long r2 = dispositivosBloque.gestionarColas();

      // Llama al planificador del manejador de hilos y procesos
        try {
                manejadorProcesos.planificador();
        }
        catch (...) {
            if (r1 || r2)
                  throw NoHayHilosListos();
            else
```

```
                throw NoHayHilos();
        }
}

// Interfaz al núcleo para las llamadas al sistema
void Nucleo::llamada() {
        CPU& cpu = MaqVirtual::cpu;
        Contexto& contexto = cpu.obtenerContexto();
        ManejadorMemoria& manejadorMemoria = Nucleo::manejadorMemoria;

        // Obtención del hilo que se está ejecutando ahora, para saber
        //   después si hubo cambio de contexto.
        long hilo = manejadorProcesos.actual();

        // Obtener el proceso en ejecución.
        Proceso* proceso = manejadorProcesos.procesoEjecucion();

        // Declaración de variables que luego usaremos como parametros en
        //   las llamadas al sistema.
        long param1, param2, param3, res;
        char cadena[100];
        cadena[99] = '\0';

        // Lee el primer valor de los parametros: el servicio que se solicita
        long servicio = cpu.desapilar();

        // En funcion del servicio extraigo el resto de parametros y realizo
        //   la llamada que corresponda.
        try {
              switch (servicio) {
                    case CREAR_HILO:
                          param1 = cpu.desapilar();
                          res = manejadorProcesos.crearHilo(param1);
                          break;
                    case TERMINAR_HILO:
                          param1 = cpu.desapilar();
                          manejadorProcesos.terminarHilo(param1);
                          break;
                    case TERMINAR:
                          manejadorProcesos.terminarHilo();
                          break;
                    case CAMBIAR_UNIDAD:
                          param1 = cpu.desapilar();
                          proceso->cambiarUnidad(param1);
                          break;
                    case CAMBIAR_DIRECTORIO:
                          param1 = cpu.desapilar();
                          manejadorMemoria.deUsuario(contexto.codigo,
                                              param1, cadena, 99);
                          proceso->cambiarDirectorio(cadena);
                          break;
                    case CREAR_FICHERO:
                          param1 = cpu.desapilar();
                          manejadorMemoria.deUsuario(contexto.codigo,
```

```
                        param1, cadena, 99);
        param2 = cpu.desapilar();
        res = proceso->crear(cadena, param2);
        break;
case ABRIR_FICHERO:
        param1 = cpu.desapilar();
        manejadorMemoria.deUsuario(contexto.codigo,
                        param1, cadena, 99);
        res = proceso->abrir(cadena);
        break;
case CERRAR_FICHERO:
        param1 = cpu.desapilar();
        proceso->cerrar(param1);
        break;
case BORRAR_FICHERO:
        param1 = cpu.desapilar();
        manejadorMemoria.deUsuario(contexto.codigo,
                        param1, cadena, 99);
        proceso->borrar(cadena);
        break;
case LEER_FICHERO:
        param1 = cpu.desapilar();
        param2 = cpu.desapilar();
        manejadorMemoria.deUsuario(contexto.codigo,
                        param2, cadena, 99);
        param3 = cpu.desapilar();
        res = proceso->leer(param1, cadena, param3);
        break;
case ESCRIBIR_FICHERO:
        param1 = cpu.desapilar();
        param2 = cpu.desapilar();
        manejadorMemoria.deUsuario(contexto.codigo,
                        param2, cadena, 99);
        param3 = cpu.desapilar();
        res = proceso->escribir(param1, cadena, param3);
        break;
case SALTAR_FICHERO:
        param1 = cpu.desapilar();
        param2 = cpu.desapilar();
        param3 = cpu.desapilar();
        res = proceso->saltar(param1, param2, param3);
        break;
case EJECUTAR:
        param1 = cpu.desapilar();
        manejadorMemoria.deUsuario(contexto.codigo,
                        param1, cadena, 99);
        proceso->ejecutar(cadena);
        break;
case LEER_ENTRADA_ESTANDAR:
        res = proceso->leerCaracter();
        break;
case ESCRIBIR_SALIDA_ESTANDAR:
        param1 = cpu.desapilar();
        proceso->imprimirCaracter((char)param1);
```

```
                        break;
                default:
                        res = -1;
        }
    }
    catch(...) {
        res = -1;
    }

    // Si no ha habido cambio de contexto, devuelve el resultado al hilo
    //   en el registro general 0
    try {
        if (hilo == manejadorProcesos.actual())
            contexto.registros[0] = res;
    }
    catch(...) {}
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/memoria.cpp
//    Implementación del manejador de memoria.
// ----------------------------------------------------------------------

#include <memoria.h>
#include <mv.h>


long ManejadorMemoria::asignar(long cantidad) {
      return MaqVirtual::mmu.asignar(cantidad);
}

long ManejadorMemoria::reservar(long segmento) {
      return MaqVirtual::mmu.reservar(segmento);
}

long ManejadorMemoria::liberar(long segmento) {
      return MaqVirtual::mmu.liberar(segmento);
}


void ManejadorMemoria::aUsuario(long segmento, long dtno, char* fte, long nBytes) {
      for (long i=0; i<nBytes; i++)
            MaqVirtual::cpu.escribirByte(segmento, dtno+i, fte[i]);
}

void ManejadorMemoria::deUsuario(long segmento, long fte, char* dtno, long nBytes) {
      for (long i=0; i<nBytes; i++)
            dtno[i] = MaqVirtual::cpu.leerByte(segmento, fte+i);
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/hilo.cpp
//    Implementación de un hilo.
// ----------------------------------------------------------------------

#include <hilo.h>
#include <mv.h>
#include <nucleo.h>


Hilo::Hilo() {
      estado = LIBRE;
}

// Devuelve el proceos asociado
Proceso* Hilo::obtenerProceso() {
      return proceso;
}

// Devuelve la petición de E/S
Peticion& Hilo::obtenerPeticion() {
      return peticion;
}

// Métodos para poner la petición
void Hilo::ponerPeticion(Peticion::Tipo codigo) {
      peticion.codigo = codigo;
}

void Hilo::ponerPeticion(Peticion::Tipo codigo, long dato) {
      peticion.codigo = codigo;
      peticion.dato = dato;
}

void Hilo::ponerPeticion(Peticion::Tipo codigo, char* datos) {
      peticion.codigo = codigo;
      peticion.datos = datos;
}

void Hilo::ponerPeticion(Peticion::Tipo codigo, long dato, char* datos) {
      peticion.codigo = codigo;
      peticion.dato = dato;
      peticion.datos = datos;
}

// ¿Está libre esta entrada en la tabla de hilos?
char Hilo::libre() {
      if (estado==LIBRE)
            return 1;
      return 0;
```

```
}

// ¿Está este hilo en estado LISTO?
char Hilo::listo() {
      if (estado==LISTO)
            return 1;
      return 0;
}


// Métodos para las transiciones entre estados.
//   Crea un hilo a partir de otro (para la llamada al sistema CREAR_HILO)
void Hilo::iniciar(Proceso* proc, long direccion) {
      // Inicializa el contexto del nuevo hilo
      contexto = MaqVirtual::cpu.obtenerContexto();
      contexto.pc = direccion;

      // Reserva la memoria necesaria
      Nucleo::manejadorMemoria.reservar(contexto.codigo);
      contexto.pila = Nucleo::manejadorMemoria.asignar(128);

      // Asocia e inicia el proceso
      proceso = proc;
      proceso->iniciar();

      // Pasa el hilo a LISTO
      estado = LISTO;
}

//   Crea un hilo nuevo para un proceso nuevo (ejecución de programas)
void Hilo::iniciar(Proceso* proc, Contexto& ctx) {
      // Inicializa el contexto del nuevo hilo
      contexto = ctx;

      // Asocia el proceso
      proceso = proc;

      // Pasa el hilo a LISTO
      estado = LISTO;
}

//   Pasa de LISTO a EJECUCION
void Hilo::ejecutar() {
      MaqVirtual::cpu.ponerContexto(contexto);
      estado = EJECUCION;
}

//   Pasa de EJECUCION a LISTO
void Hilo::dormir() {
      contexto = MaqVirtual::cpu.obtenerContexto();
      estado = LISTO;
}

//   Pasa de EJECUCION a SUSPENDIDO
```

```
void Hilo::suspender() {
      contexto = MaqVirtual::cpu.obtenerContexto();
      estado = SUSPENDIDO;
}

//   Pasa de SUSPENDIDO a LISTO
void Hilo::reactivar(long resultado) {
      contexto.registros[0] = resultado;
      estado = LISTO;
}

//   Termina un hilo, da igual en que estado se enecuentre
void Hilo::terminar() {
      // Libera la memoria asociada
      Nucleo::manejadorMemoria.liberar(contexto.codigo);
      Nucleo::manejadorMemoria.liberar(contexto.pila);

      // Termina el proceso
      proceso->terminar();

      // Pone el hilo en LIBRE
      estado = LIBRE;
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/proceso.cpp
//    Implementación de un proceso.
// ----------------------------------------------------------------------

#include <string.h>

#include <proceso.h>
#include <nucleo.h>


Proceso::Proceso() {
      uso = 0;
}

// ¿Está libre el proceso?
long Proceso::libre() {
      return !uso;
}


// Inicia el proceso o incrementa su uso si el proceso ya existe
long Proceso::iniciar(long u, long d) {
      if (uso==0) {
            // Inicializa la entrada/salida estandar
            abrir("--teclado--");
            abrir("--pantalla--");

            // Inicializa la unidad y el directorio actuales
            unidad = u;
            directorio = d;
            Nucleo::sfv.abrir(directorio);
      }
      uso++;

      return uso;
}

// Termina el proceso
long Proceso::terminar() {
      // Reduce el uso del proceso
      uso--;

      // Si el uso es cero lo termina
      if (uso==0) {
            // Cierra todos los ficheros abiertos de este proceso
            for (long i=0; i<N_F_LOCALES; i++) {
                  if (!ficherosLocales[i].libre())
                        ficherosLocales[i].cerrar();
            }
```

```
            // Cierra el directorio actual del proceso
            Nucleo::sfv.cerrar(directorio);
      }

      return uso;
}

// Cambia la unidad actual
void Proceso::cambiarUnidad(long num) {
      // Si la nueva unidad existe cambia;
      if (Nucleo::sfv.activo(num)) {
            // Cierra el directorio actual
            Nucleo::sfv.cerrar(directorio);

            // Cambia a la nueva unidad
            unidad = num;

            // Abre el directorio raíz de la nueva unidad como el nuevo
            //   directorio actual
            directorio = Nucleo::sfv.abrir(unidad, 0, "/");
      }

      // si no, genera un error
      throw "esa unidad no existe";
}

// Cambia el directorio actual
void Proceso::cambiarDirectorio(char* nombre) {
      // Abre el nuevo directorio
      long dir = Nucleo::sfv.abrir(unidad, directorio, nombre);

      // Cierra el directorio actual
      Nucleo::sfv.cerrar(directorio);

      // Actualiza el directorio actual
      directorio = dir;
}

// Crea un nuevo fichero
long Proceso::crear(char* nombre, long tipo) {
      // Busca una entrada libre en la tabla de ficheros locales,
      for (long fL=0; fL<N_F_LOCALES; fL++)
            if (ficherosLocales[fL].libre()) {
                  // Crea el fichero
                  ficherosLocales[fL].crear(unidad, directorio,
                                            nombre, tipo);

                  // Devuelve el identificador local del fichero
                  return fL;
            }

      //  si no la encuentra genera un error
      throw "demasiados ficheros abiertos";
```

```
}

// Abre un fichero
long Proceso::abrir(char* nombre) {
      // Busca una entrada libre en la tabla de ficheros locales,
      for (long fL=0; fL<N_F_LOCALES; fL++)
            if (ficherosLocales[fL].libre()) {
                  // Abre el fichero
                  ficherosLocales[fL].abrir(unidad, directorio, nombre);

                  // Devuelve el identificador local del fichero
                  return fL;
            }

      //  si no la encuentra genera un error
      throw "demasiados ficheros abiertos";
}

// Cierra un fichero
void Proceso::cerrar(long id) {
      // Si el identificador está fuera de rango genera un error
      if (id<0 || id>=N_F_LOCALES)
            throw "número de fichero fuera de rango";

      // Cierra el fichero
      ficherosLocales[id].cerrar();
}

// Lee de un fichero
long Proceso::leer(long id, void* destino, long nBytes) {
      if (id<0 || id>=N_F_LOCALES)
            throw "número de fichero fuera de rango";

      return ficherosLocales[id].leer(destino, nBytes);
}

// Escribe en un fichero
long Proceso::escribir(long id, void* fuente, long nBytes) {
      if (id<0 || id>=N_F_LOCALES)
            throw "número de fichero fuera de rango";

      return ficherosLocales[id].escribir(fuente, nBytes);
}

// Cambia la porsición dentro de un fichero
long Proceso::saltar(long id, long posicion, long desde) {
      if (id<0 || id>=N_F_LOCALES)
            throw "número de fichero fuera de rango";

      return ficherosLocales[id].saltar(posicion, desde);
}

// Lee un carácter de la entrada estándar
long Proceso::leerCaracter() {
```

```
      char c;
      ficherosLocales[0].leer(&c,1);
      return (long)c;
}

// Imprime un carácter en la salida estándar
void Proceso::imprimirCaracter(char c) {
      ficherosLocales[1].escribir(&c, 1);
}

// Borra un fichero
void Proceso::borrar(char* nombre) {
      Nucleo::sfv.borrar(unidad, directorio, nombre);
}

// Ejecuta un programa
void Proceso::ejecutar(char* nombre) {
      Nucleo::manejadorProcesos.ejecutar(unidad, directorio, nombre);
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/cola.cpp
//    Implementación de la cola de hilos que esperan por un recurso.
// ----------------------------------------------------------------------

#include <cola.h>


Hilo Cola::hilos[N_HILOS];


Cola::Cola() {
      primero = -1;
}


// Devuelve el identificador del hilo que es cabeza de la cola
long Cola::actual() {
      if (primero<0)
            throw "no hay ningún hilo en la cola";

      return primero;
}


// Inserta un hilo en el final de la cola
long Cola::insertar(long hilo) {
      if (primero == -1) {
            primero = hilo;
            hilos[hilo].setAnt(hilo);
            hilos[hilo].setSig(hilo);
      } else {
            hilos[hilos[primero].getAnt()].setSig(hilo);
            hilos[hilo].setAnt(hilos[primero].getAnt());
            hilos[primero].setAnt(hilo);
            hilos[hilo].setSig(primero);
      }

        return hilo;
}

// Extrae un hilo de la cola
long Cola::extraer(long hilo) {
      if (primero == -1)
            throw "no hay ningún hilo en la cola";

      // Por defecto (si no se específica el número de hilo) se elimina el
      //   hilo de la cabeza
      if (hilo==-1)
            hilo = primero;
```

```
      if (hilos[hilo].getSig() == primero)
            primero = -1;
      else {
            hilos[hilos[hilo].getAnt()].setSig(hilos[hilo].getSig());
            hilos[hilos[hilo].getSig()].setAnt(hilos[hilo].getAnt());
            if (hilo == primero)
                  primero = hilos[hilo].getSig();
      }

      return hilo;
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/procesos.cpp
//    Implementación del manejador de hilos y de procesos.
// ----------------------------------------------------------------------

#include <procesos.h>
#include <nucleo.h>
#include <ejec99.h>


void ManejadorProcesos::iniciar() {
      // Registrar formatos de ficheros ejecutables
      ejecutables[0] = new Ejecutable99();
      for (long i=1; i<N_EJECUTABLES; i++)
            ejecutables[i] = 0;
}

void ManejadorProcesos::terminar() {
      // Limpia la tabla de formatos de ficheros ejecutables
      for (long i=0; i<N_EJECUTABLES; i++)
            if (ejecutables[i])
                  delete ejecutables[i];
}


Proceso* ManejadorProcesos::procesoEjecucion() {
      return hilos[actual()].obtenerProceso();
}


long ManejadorProcesos::crearHilo(long direccion) {
      // Busca una entrada libre
      long hilo;
      for (hilo=0; hilo<N_HILOS && !hilos[hilo].libre(); hilo++);
      if (hilo==N_HILOS)
            throw "ya no quedan entradas de hilos libres";

      // Inicia el hilo
      Proceso* proceso = hilos[actual()].obtenerProceso();
      hilos[hilo].iniciar(proceso, direccion);

      // Inserta el hilo en la cola
      insertar(hilo);

      return hilo;
}

void ManejadorProcesos::terminarHilo(long id) {
      // Si no se especifica el hilo se asume que es el actual
      if (id<0)
```

```
            id = actual();

      // Comprueba que el id es válido
      if (id>=N_HILOS)
            throw "número de hilo fuera de rango";

      // Comprueba que el hilo existe
      if (hilos[id].libre())
            throw "ese hilo no existe";

      extraer(id);
      hilos[id].terminar();
}


long ManejadorProcesos::ejecutar(long unidad, long pwd, char* nombre) {
      // Busco una entrada libre de hilo
      long hilo;
      for (hilo=0; hilo<N_HILOS && !hilos[hilo].libre(); hilo++);
      if (hilo==N_HILOS)
            throw "ya no quedan entradas de hilos libres";

      // Busco una entrada libre de proceso
      long proc;
      for (proc=0; proc<N_PROCESOS && !procesos[proc].libre(); proc++);
      if (proc==N_PROCESOS)
            throw "ya no quedan entradas de procesos libres";

      // Abre el fichero.
      long id = Nucleo::sfv.abrir(unidad, pwd, nombre);

      // Prueba a ejecutar el fichero
      Contexto contexto;
      for (long i=0; i<N_EJECUTABLES && ejecutables[i]!=0; i++)
            try {
                  ejecutables[i]->ejecutar(id, contexto);

                  // Cerrar el fichero
                  Nucleo::sfv.cerrar(id);

                  // Iniciar el proceso y el hilo
                  procesos[proc].iniciar(unidad, pwd);
                  hilos[hilo].iniciar(&procesos[proc], contexto);

                  // Insertar el hilo en la cola de hilos listos
                  insertar(hilo);

                  // Regresar con exito
                  return hilo;
            }
            catch(...) {}

      // Cerrar el fichero
      Nucleo::sfv.cerrar(id);
```

```
      // No se reconoció el fichero.
      throw "fichero no ejecutable";
}


void ManejadorProcesos::planificador() {
      long hilo = actual();

      if (hilos[hilo].listo()) {
            // Si no hay ningún hilo en ejecución pasa el hilo de la
            //   cabeza de la cola de hilos listos a EJECUCIÓN
            hilos[hilo].ejecutar();
            tiempo = TAJADA;
      } else {
            // Si ya hay algún hilo en EJECUCION...
            // Decrementa el contador de tiempo
            tiempo--;

            // Si ya se ha terminado el tiempo del hilo actual
            //   cambia el hilo en ejecución
            if (tiempo==0) {
                  // Pasa el hilo en ejecución a LISTO
                  hilos[hilo].dormir();
                  extraer(hilo);
                  insertar(hilo);

                  // Pasa el nuevo hilo que es cabeza de la cola de
                  //   hilos listos a EJECUCION
                  hilos[actual()].ejecutar();

                  // Inicializa el contador de tiempo
                  tiempo = TAJADA;
            }
      }
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/controla.cpp
//    Implementación de un controlador de dispositivo.
// ----------------------------------------------------------------------

#include <controla.h>
#include <nucleo.h>


long Controlador::nuevaPeticion() {
      // Transición EJECUCION -> SUSPENDIDO
      long hilo = Nucleo::manejadorProcesos.extraer();
      hilos[hilo].suspender();
      insertar(hilo);

      // Si no habían hilos en SUSPENDIDO inicializa el contador de tiempo
      if (actual()==hilo)
            tiempo = 10;

      return hilo;
}


void Controlador::planificador() {
      long hilo = actual();
      if (--tiempo==0) {
            // Ejecutamos la petición pendiente
            long resultado = ejecutarPeticion();

            // Transición SUSPENDIDO -> LISTO
            extraer();
            hilos[hilo].reactivar(resultado);
            Nucleo::manejadorProcesos.insertar(hilo);

            // Inicializo el contador.
            // Esto debería depender del dispositivo concreto.
            tiempo = 10;
      }
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/ccar.cpp
//    Implementación de un controlador de tipo caracter.
// ----------------------------------------------------------------------

#include <ccar.h>
#include <string.h>

// Ejecuta la petición del hilo que es cabeza de la cola
long ControladorCaracter::ejecutarPeticion() {
      // Obtiene la petición
      Peticion& peticion = hilos[actual()].obtenerPeticion();

      // Ejecuta la petición
      if (peticion.codigo==Peticion::LEER_CARACTER)
            return _leerCaracter();
      else if (peticion.codigo==Peticion::ESCRIBIR_CARACTER)
            _escribirCaracter(peticion.dato);

      return 0;
}

// Inicializa el controlador de carácter asignandole un identificador (nombre)
ControladorCaracter::ControladorCaracter(char* n) {
      nombre = new char[strlen(n)+1];
      strcpy(nombre, n);
}

ControladorCaracter::~ControladorCaracter() {
      if (nombre)
            delete nombre;
}


long ControladorCaracter::es(char* n) {
      if (strcmp(nombre, n) == 0)
            return 1;
      return 0;
}

// Para leer un carácter
char ControladorCaracter::leerCaracter() {
      try {
            // Registra la petición
            long h = nuevaPeticion();
            hilos[h].ponerPeticion(Peticion::LEER_CARACTER);
            return 0;
      }
      catch(...) {
            return _leerCaracter();
```

```
	}
}

// Para imiprimir un carácter
void ControladorCaracter::escribirCaracter(char caracter) {
	try {
		// Registra la petición
		long h = nuevaPeticion();
		hilos[h].ponerPeticion(Peticion::ESCRIBIR_CARACTER, caracter);
	}
	catch(...) {
		_escribirCaracter(caracter);
	}
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/cteclado.cpp
//    Implementación del controlador del teclado.
// ----------------------------------------------------------------------

#include <cteclado.h>
#include <mv.h>

char ControladorTeclado::_leerCaracter() {
      return MaqVirtual::teclado.leerCaracter();
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/cpantall.cpp
//    Implementación del controlador de la pantalla.
// ----------------------------------------------------------------------

#include <cpantall.h>
#include <mv.h>


void ControladorPantalla::_escribirCaracter(char caracter) {
      MaqVirtual::pantalla.escribirCaracter(caracter);
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/cbloque.cpp
//    Implmentación de un controlador de tipo bloque.
// ----------------------------------------------------------------------

#include <cbloque.h>


long ControladorBloque::ejecutarPeticion() {
      // Obtiene la petición
      Peticion& peticion = hilos[actual()].obtenerPeticion();

      // Ejecuta la petición
      if (peticion.codigo==Peticion::LEER_BLOQUE)
            _leer(peticion.datos, peticion.dato);
      else if (peticion.codigo==Peticion::ESCRIBIR_BLOQUE)
            _escribir(peticion.datos, peticion.dato);

      return 0;
}


// Para lee un bloque
long ControladorBloque::leer(char* datos, long bloque) {
      try {
            // Registra la petición
            long h = nuevaPeticion();
            hilos[h].ponerPeticion(Peticion::LEER_BLOQUE, bloque, datos);
            return 0;
      }
      catch(...) {
            return _leer(datos, bloque);
      }
}

// Para escribir un bloque
long ControladorBloque::escribir(char* datos, long bloque) {
      try {
            // Registra la petición
            long h = nuevaPeticion();
            hilos[h].ponerPeticion(Peticion::ESCRIBIR_BLOQUE, bloque,datos);
            return 0;
      }
      catch(...) {
            return _escribir(datos, bloque);
      }
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/cdisco.cpp
//    Implementación del controlador del disco virtual.
// ----------------------------------------------------------------------

#include <cdisco.h>


ControladorDisco::ControladorDisco(Disco& d) {
      disco = &d;

      tamBloque = disco->obtenerTamBloque();
      numBloques = disco->obtenerNumBloques();
}

long ControladorDisco::_leer(char *buffer, long bloque) {
      return disco->leer(buffer, bloque);
}

long ControladorDisco::_escribir(char *buffer, long bloque) {
      return disco->escribir(buffer, bloque);
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/caracter.cpp
//    Implementación de la lista de controladores de tipo caracter.
// ----------------------------------------------------------------------

#include <caracter.h>
#include <cteclado.h>
#include <cpantall.h>


void DispositivosCaracter::iniciar() {
      // Teclado (0) y pantalla (1)
      dispositivos[0] = new ControladorTeclado("--teclado--");
      dispositivos[1] = new ControladorPantalla("--pantalla--");

      // Ya no quedan más dispositivos.
      for (long i=2; i<NUM_DISPOS_CARACTER; i++)
            dispositivos[i] = 0;
}


ControladorCaracter* DispositivosCaracter::obtenerControlador(char* nombre) {
      for (long i=0; i<NUM_DISPOS_CARACTER; i++)
            if (dispositivos[i])
                  if (dispositivos[i]->es(nombre))
                        return dispositivos[i];

      return 0;
}


long DispositivosCaracter::gestionarColas() {
      long res = 0;

      // Recorre la tabla de controladores de tipo carácter
      for (long i=0; i<NUM_DISPOS_CARACTER; i++)
            // Si el controlador existe...
            if (dispositivos[i]) {
                  try {
                        // ...llama a su planificador
                        dispositivos[i]->planificador();
                        res = 1;
                  }
                  catch(...) {}
            }
      return res;
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/es/bloque.cpp
//    Implementación de la lista de controladores de tipo bloque.
// ----------------------------------------------------------------------

#include <bloque.h>
#include <mv.h>
#include <cdisco.h>


void DispositivosBloque::iniciar() {
      for (long i=0; i<NUM_DISPOS_BLOQUE; i++) {
            dispositivos[i] = NULL;
            try {
                  iniDisco(i);
            }
            catch(...) {}
      }
}


void DispositivosBloque::iniDisco(long id) {
      Disco& disco = MaqVirtual::disco(id);
      dispositivos[id] = new ControladorDisco(disco);
}

ControladorBloque* DispositivosBloque::obtenerControlador(long id) {
      if (id<0 || id>=NUM_DISPOS_BLOQUE)
            throw "número de dispositivo fuera de rango";
      return dispositivos[id];
}


long DispositivosBloque::gestionarColas() {
      long res = 0;

      // Recorre la tabla de controladores de tipo bloque
      for (long i=0; i<NUM_DISPOS_BLOQUE; i++)
            // Si el controlador existe...
            if (dispositivos[i]) {
                  try {
                        // ...llama a su planificador
                        dispositivos[i]->planificador();
                        res = 1;
                  }
                  catch(...) {}
            }
      return res;
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/sf/sfv.cpp
//    Implementación del sistema de ficheros virtual.
// ----------------------------------------------------------------------

#include <string.h>

#include <sf.h>
#include <nucleo.h>
#include <eafitsf.h>


long SFV::buscar(long unidad, long nInodo) {
      // Recorre la tabla de ficheros
      for (long i=0; i<MAX_FICHEROS_ABTOS; i++)
            // Si el fichero existe...
            if (ficheros[i])
                  // ...si el fichero es el buscado devuelve su id.
                  if (ficheros[i]->es(sf[unidad], nInodo))
                        return i;

      return -1;
}

// Inicializa el sistema de ficheros
void SFV::iniciar() {
      long i;

      // Inicializa la tabla de ficheros
      for (i=0; i<MAX_FICHEROS_ABTOS; i++)
            ficheros[i] = NULL;

      // Inicializa la tabla de sitemas de ficheros
      for (i=0; i<MAX_NUM_SF; i++) {
            sf[i] = NULL;
            try {
                  ControladorBloque* cB;
                  cB = Nucleo::dispositivosBloque.obtenerControlador(i);
                  iniSF(cB, i);
            }
            catch(...) {}
      }
}

void SFV::terminar() {
      long i;

      // Limpia la tabla de ficheros
      for (i=0; i<MAX_FICHEROS_ABTOS; i++)
            if (ficheros[i])
```

```
                while (cerrar(i));

        // Limpia la tabla de sistemas de ficheros
        for (i=0; i<MAX_NUM_SF; i++)
                if (sf[i])
                        delete sf[i];
}


// Inicializa un sistema de ficheros
void SFV::iniSF(ControladorBloque* cB, long n) {
        if (cB==NULL)
                throw "ese dispositivo no existe";

        try {
                sf[n] = new EafitSF(cB);
        }
        catch(...) {
                // Aquí añadir soporte para otros sistemas de ficheros
                throw "sistema de ficheros desconocido";
        }

        // Abrir el directorio raiz
        for (long i=0; i<MAX_FICHEROS_ABTOS; i++)
                if (ficheros[i]==NULL) {
                        ficheros[i] = sf[n]->abrir(0);
                        sf[n]->ponerDirRaiz(i);
                        return;
                }

        delete sf[n];
        throw "demasiados ficheros abiertos";
}

long SFV::activo(long id) {
        if (id<0 || id>=MAX_NUM_SF)
                throw "número de unidad fuera de rango";

        if (sf[id]==NULL)
                return 0;

        return 1;
}


long SFV::obtenerTamano(long id) {
        if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
                throw "ese fichero no está abierto";

        return ficheros[id]->obtenerTamano();
}


long SFV::crear(long unidad, long pwd, char *nombre, long tipo) {
```

```
	Fichero* fich;
	char *token1, *token2;
	long id, nInodo, temporal=0;

	// Determina si es una ruta absoluta o relativa, obtiene el directorio
	//   inicial donde empezará la busqueda (fich)
	if (nombre[0]=='/') {
		id = sf[unidad]->obtenerDirRaiz();
		fich = ficheros[id];
	} else
		fich = ficheros[pwd];

	token1 = strtok(nombre, "/");
	token2 = strtok(NULL, "/");
	while (token2!=NULL) {
		nInodo = fich->idFichero(token1);
		if (temporal)
			delete fich;

		    // Comprueba si el fichero ya está abierto
		id = buscar(unidad, nInodo);
		if (id!=-1) {
			temporal = 0;
			fich = ficheros[id];
		} else {
			// Lo abre temporalmente
			temporal = 1;
			fich = sf[unidad]->abrir(nInodo);
		}

		// Siguiente subdirectorio
		token1 = token2;
		token2 = strtok(NULL, "/");
	}

	// Busca una entrada libre, si no hay genera un error
	for (id=0; id<MAX_FICHEROS_ABTOS && ficheros[id]!=NULL; id++);
	if (id>=MAX_FICHEROS_ABTOS)
		throw "demasiados ficheros abiertos";

	// Crea el fichero
	ficheros[id] = fich->crearFichero(token1, tipo);

	if (temporal)
		delete fich;

	return id;
}

long SFV::abrir(long unidad, long pwd, char *nombre) {
	Fichero* fich;
	char* cad;
	long id, nInodo, temporal=0;
```

```
	// Determina si es una ruta absoluta o relativa, obtiene el directorio
	//   inicial donde empezará la busqueda (fich)
	if (nombre[0]=='/') {
		id = sf[unidad]->obtenerDirRaiz();
		fich = ficheros[id];
	} else
		fich = ficheros[pwd];

	cad = strtok(nombre, "/");
	while (cad!=NULL) {
		nInodo = fich->idFichero(cad);
		if (temporal)
			delete fich;

		// Comprueba si el fichero ya está abierto
		id = buscar(unidad, nInodo);
		if (id!=-1) {
			temporal = 0;
			fich = ficheros[id];
		} else {
			// Lo abre temporalmente
			temporal = 1;
			fich = sf[unidad]->abrir(nInodo);
		}

		// Siguiente subdirectorio
		cad = strtok(NULL, "/");
	  }

	// Abre el fichero
	if (id==-1) {
		// Busca una entrada libre, si no hay genera error
		for (id=0; id<MAX_FICHEROS_ABTOS && ficheros[id]!=NULL; id++);
		if (id>=MAX_FICHEROS_ABTOS)
			throw "demasiados ficheros abiertos";

		ficheros[id] = fich;
	} else
		// Si ya estaba abierto solo incrementa su uso
		ficheros[id]->abrir();

	return id;
}

// Incrementa el uso de un fichero que ya está abierto
long SFV::abrir(long id) {
	// Comprueba que el identificador se válido
	if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
		throw "ese fichero no está abierto";

	return ficheros[id]->abrir();
}

long SFV::cerrar(long id) {
```

```
	// Comprueba que el identificador se válido
	if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
		throw "ese fichero no está abierto";

	// Cierra el fichero
	long uso = ficheros[id]->cerrar();
	if (uso==0) {
		delete ficheros[id];
		ficheros[id] = NULL;
	}
	return uso;
}

void SFV::borrar(long unidad, long pwd, char* nombre) {
	Fichero* fich;
	char *token1, *token2;
	long id, nInodo, temporal=0;

	// Determina si es una ruta absoluta o relativa, obtiene el directorio
	//   inicial donde empezará la busqueda (fich)
	if (nombre[0]=='/') {
		id = sf[unidad]->obtenerDirRaiz();
		fich = ficheros[id];
	} else
		fich = ficheros[pwd];

	token1 = strtok(nombre, "/");
	token2 = strtok(NULL, "/");
	while (token2!=NULL) {
		nInodo = fich->idFichero(token1);
		if (temporal)
			delete fich;

		// Comprueba si el fichero ya está abierto
		id = buscar(unidad, nInodo);
		if (id!=-1) {
			temporal = 0;
			fich = ficheros[id];
		} else {
			// Lo abre temporalmente
			temporal = 1;
			fich = sf[unidad]->abrir(nInodo);
		}

		// Siguiente subdirectorio
		token1 = token2;
		token2 = strtok(NULL, "/");
	}

	// Borra el fichero
	fich->borrarFichero(token1);

	if (temporal)
		delete fich;
```

```
}

long SFV::leer(long id, void *destino, long numBytes) {
      if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
            throw "ese fichero no está abierto";

      return ficheros[id]->leer((char*)destino, numBytes);
}

long SFV::escribir(long id, void *fuente, long numBytes) {
      if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
            throw "ese fichero no está abierto";

      return ficheros[id]->escribir((char*)fuente, numBytes);
}

long SFV::saltar(long id, long posicion, long desde) {
      if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
            throw "ese fichero no está abierto";

      return ficheros[id]->saltar(posicion, desde);
}


long SFV::infoFichero(long id, InfoFichero& info) {
      if (id<0 || id>=MAX_FICHEROS_ABTOS || ficheros[id]==NULL)
            throw "ese fichero no está abierto";

      return ficheros[id]->infoFichero(info);
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/sf/flocal.cpp
//    Implementación de los ficheros locales de cada proceso.
// ----------------------------------------------------------------------

#include <flocal.h>
#include <nucleo.h>


FicheroLocal::FicheroLocal() {
      tipo = LIBRE;
}


long FicheroLocal::libre() {
      if (tipo==LIBRE)
            return 1;
      return 0;
}


long FicheroLocal::crear(long unidad, long pwd, char* nombre, long tipo) {
      // Comprueba que la entrada esté libre
      if (tipo!=LIBRE)
            throw "no puedes abrir un fichero que ya está abierto";

      // Crea el fichero
      id = Nucleo::sfv.crear(unidad, pwd, nombre, tipo);
      FicheroLocal::tipo = FICHERO;

      // Inicialmente la posición del fichero es cero
      posicion = 0;

      // Devuelve el identificador global del fichero
      return id;
}

long FicheroLocal::abrir(long unidad, long pwd, char* nombre) {
      // Comprueba que la entrada esté libre
      if (tipo!=LIBRE)
            throw "no puedes abrir un fichero que ya está abierto";

      // Busca un controlador de dispositivo identificado con "nombre"
      controlador = Nucleo::dispositivosCaracter.obtenerControlador(nombre);
      if (controlador) {
            // El fichero es en realidad un controlador
            tipo = DISPOSITIVO;
            return -1;
      } else {
            // Es un fichero normal, lo abre
```

```
            tipo = FICHERO;
            posicion = 0;
            return Nucleo::sfv.abrir(unidad, pwd, nombre);
      }
}

void FicheroLocal::cerrar() {
      if (tipo==LIBRE)
            throw "no puedes cerrar un fichero que no está abierto";

      // Si es un fichero normal lo cierra
      if (tipo==FICHERO)
            Nucleo::sfv.cerrar(id);

      tipo = LIBRE;
}

long FicheroLocal::leer(void* destino, long nBytes) {
      if (tipo==LIBRE)
            throw "no puedes leer de un fichero que no está abierto";

      // Realiza una operación u otra dependiendo del tipo de fichero
      if (tipo==FICHERO) {
            // Si es un fichero normal leemos de él llamando a SFV
            Nucleo::sfv.saltar(id, posicion, PRINCIPIO);
            nBytes = Nucleo::sfv.leer(id, destino, nBytes);
            posicion += nBytes;
      }
      else if (tipo==DISPOSITIVO)
            // Si es un fichero de tipo dispositivo leemos con el
            //   método ControladorCaracter::leerCaracter
            for (long i=0; i<nBytes; i++)
                  ((char*)destino)[i] = controlador->leerCaracter();

      // Devolvemos el número de bytes leidos
      return nBytes;
}

long FicheroLocal::escribir(void* fuente, long nBytes) {
      if (tipo==LIBRE)
            throw "no puedes escribir en un fichero que no está abierto";

      // Realiza una operación u otra dependiendo del tipo de fichero
      if (tipo==FICHERO) {
            // Si es un fichero normal escribimos en él llamando a SFV
            Nucleo::sfv.saltar(id, posicion, PRINCIPIO);
            nBytes = Nucleo::sfv.escribir(id, fuente, nBytes);
            posicion += nBytes;
      }
      else if (tipo==DISPOSITIVO)
            // Si es un fichero de tipo dispositivo escribimos con el
            //   método ControladorCaracter::leerCaracter
            for (long i=0; i<nBytes; i++)
                  controlador->escribirCaracter(((char*)fuente)[i]);
```

```
      // Devolvemos el número de bytes escritos
      return nBytes;
}

long FicheroLocal::saltar(long pos, long desde) {
      if (tipo==LIBRE)
            throw "no puedes saltar en un fichero que no está abierto";

      // Si es un fichero normal cambiamos la posición (si es de tipo
      //   dispositivo no tiene sentido cambiarla)
      if (tipo==FICHERO)
            posicion = Nucleo::sfv.saltar(id, pos, desde);

      return posicion;
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/sf/eafitsf.cpp
//    Implementación del sistema de ficheros EAFIT. La parte que
//    entiende el formato del disco.
// ----------------------------------------------------------------------

#include <string.h>

#include <eafitsf.h>
#include <bloque.h>
#include <sf.h>


void EafitSF::leerMapaDatos() {
      for (long i=0; i<superbloque.numBloquesMD; i++)
            dispositivo->leer(mapaDatos+tamBloque*i, superbloque.primerBloqueMD+i);
}

void EafitSF::escribirMapaDatos() {
      if (mapaDatos)
            for (long i=0; i<superbloque.numBloquesMD; i++)
                  dispositivo->escribir(mapaDatos+tamBloque*i, superbloque.primerBloqueMD+i);
}

void EafitSF::leerMapaInodos() {
      for (long i=0; i<superbloque.numBloquesMI; i++)
            dispositivo->leer(mapaInodos+tamBloque*i, superbloque.primerBloqueMI+i);
}

void EafitSF::escribirMapaInodos() {
      if (mapaDatos)
            for (long i=0; i<superbloque.numBloquesMI; i++)
                  dispositivo->escribir(mapaInodos+tamBloque*i, superbloque.primerBloqueMI+i);
}



EafitSF::EafitSF(ControladorBloque* d) : SF(d) {
      mapaDatos = mapaInodos = NULL;

      // Lee el superbloque
      char* buffer = new char[tamBloque];
      dispositivo->leer(buffer, 0);
      memcpy(&superbloque, buffer, sizeof(superbloque));
        delete buffer;

      // Comprueba que sea válido
      if (superbloque.id != EAFITSF_ID)
            throw "este disco no tiene uns sitema de ficheros Eafit";
```

```
      // Carga los mapas de bits
      mapaDatos = new char[superbloque.numBloquesMD * tamBloque];
        leerMapaDatos();

      mapaInodos = new char[superbloque.numBloquesMI * tamBloque];
      leerMapaInodos();
}

EafitSF::EafitSF(ControladorBloque* d, long numInodos) : SF(d) {
      mapaDatos = mapaInodos = NULL;

      char* buffer = new char[tamBloque];

      // Debe garantizarse un minimo de inodos, para que el mapa de bits
      // ocupe al menos un bloque.
      if (numInodos < tamBloque)
            numInodos = tamBloque;

        // numero de inodos que tiene cada bloque
      long inodosXBloque = tamBloque/sizeof(InodoESF);

      // Construcción y escritura del superbloque
      SuperbloqueESF sb;
      sb.id = EAFITSF_ID;

      sb.numBloquesMI = numInodos / tamBloque;
      sb.numBloquesI = numInodos / inodosXBloque;
      long resto = numBloques - 1 - sb.numBloquesMI - sb.numBloquesI;
      sb.numBloquesMD = resto / (tamBloque + 1);
      if (resto % (tamBloque + 1))
            sb.numBloquesMD++;
      sb.numBloquesD = resto - sb.numBloquesMD;

      sb.primerBloqueMD = 1;
      sb.primerBloqueMI = sb.primerBloqueMD + sb.numBloquesMD;
      sb.primerBloqueI = sb.primerBloqueMI + sb.numBloquesMI;
      sb.primerBloqueD = sb.primerBloqueI + sb.numBloquesI;

      memcpy(buffer, &sb, sizeof(sb));
      dispositivo->escribir(buffer, 0);


      // Escritura de los mapas de bits, de bloques de datos y de inodos
      // inicializados a LIBRE.
      char* mapaBits = new char[tamBloque];
      memset(mapaBits,_LIBRE,tamBloque);
      long i;
      for (i=sb.primerBloqueMD; i<sb.primerBloqueMI; i++)
            dispositivo->escribir(mapaBits,i);
      for (i=sb.primerBloqueMI; i<sb.primerBloqueI; i++)
            dispositivo->escribir(mapaBits,i);

      // Crea el directorio raiz
      InodoESF raiz;
```

```
      raiz.tipo = DIRECTORIO;
      raiz.tamano = 0;
      memset(&raiz.bloquesDirectos, _LIBRE, NBDIRECTOS*sizeof(long));
      raiz.bloquesIndirectos = _LIBRE;

        memcpy(buffer, &raiz, sizeof(raiz));
      dispositivo->escribir(buffer, sb.primerBloqueI);

      mapaBits[0] = _OCUPADO;
      dispositivo->escribir(mapaBits,sb.primerBloqueMI);

      delete mapaBits;
        delete buffer;
}

EafitSF::~EafitSF() {
      escribirMapaInodos();
      escribirMapaDatos();
}


Fichero* EafitSF::abrir(long idFichero) {
      Fichero* fich = NULL;
      InodoESF inodo = leerInodo(idFichero);
      switch (inodo.tipo) {
            case NORMAL:
                  fich = new EFichero(this, idFichero, inodo);
                  break;
            case DIRECTORIO:
                  fich = new EDirectorio(this, idFichero, inodo);
                  break;
      }
      return fich;
}


long EafitSF::reservarInodo() {
      unsigned long i;
      for (i=0; i<superbloque.numBloquesI*(tamBloque/sizeof(InodoESF)); i++)
            if (mapaInodos[i] == _LIBRE) {
                  mapaInodos[i] = _OCUPADO;
                  return i;
            }
      throw "no quedan inodos libres";
}

void EafitSF::liberarInodo(long nInodo) {
      if (mapaInodos[nInodo] == _LIBRE)
            throw "ese inodo ya está libre";
      mapaInodos[nInodo] = _LIBRE;
}

long EafitSF::reservarBloque() {
      for (long i=0; i<superbloque.numBloquesD; i++)
```

```
            if (mapaDatos[i] == _LIBRE) {
                  mapaDatos[i] = _OCUPADO;
                  return i;
            }
      throw "no quedan bloques libres";
}

void EafitSF::liberarBloque(long nBloque) {
      if (mapaDatos[nBloque] == _LIBRE)
            throw "ese inodo ya está libre";
      mapaDatos[nBloque] = _LIBRE;
}


InodoESF& EafitSF::leerInodo(long num) {
      static InodoESF inodo;

      // Calcula el número de inodos por bloque
      long inodosXBloque = tamBloque/sizeof(InodoESF);

      // Lee el bloque que contiene al inodo
      char* buffer = new char[tamBloque];
      dispositivo->leer(buffer,superbloque.primerBloqueI+num/inodosXBloque);

      // Extrae el inodo del bloque
      memcpy(&inodo, buffer+(num%inodosXBloque)*sizeof(InodoESF),
            sizeof(InodoESF));

      // Libera la memoria utilizada
      delete buffer;

      // Devuelve el inodo
      return inodo;
}

void EafitSF::escribirInodo(InodoESF& inodo, long num) {
      // Calcula el número de inodos por bloque
      long inodosXBloque = tamBloque/sizeof(InodoESF);

      // Lee el bloque que contiene al inodo
      char* buffer = new char[tamBloque];
      dispositivo->leer(buffer, superbloque.primerBloqueI +
                              num/inodosXBloque);

      // Inserta el inodo en el bloque
      memcpy(buffer + (num%inodosXBloque)*sizeof(InodoESF), &inodo,
            sizeof(InodoESF));

      // Escribe el bloque en el disco
      dispositivo->escribir(buffer, superbloque.primerBloqueI +
                              num/inodosXBloque);

      // Libera la memoria utilizada
      delete buffer;
```

```
}

long EafitSF::leerBloque(long nBloque, char* buffer) {
      return dispositivo->leer(buffer, superbloque.primerBloqueD + nBloque);
}

long EafitSF::escribirBloque(long nBloque, char* buffer) {
      return dispositivo->escribir(buffer, superbloque.primerBloqueD + nBloque);
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/sf/efichero.cpp
//    Implementación del sistema de ficheros EAFIT. La parte que
//    entiende de los ficheros.
// ----------------------------------------------------------------------

#include <string.h>
#include <eafitsf.h>

// Devuelve el número de bloque que corresponde a la posición actual dentro
//   del fichero, si es final de fichero genera un error
long EFichero::obtenerNBloque() {
      // Si es final de fichero genera error
      if (finalFichero())
            throw "final de fichero";

      // Calcula el número de bloque relativo del fichero
      long n = posicion/tamBloque;

      // Devuelve el número de bloque en el sistema de ficheros
      if (n<NBDIRECTOS)
            return inodo.bloquesDirectos[n];

        return bloquesIndirectos[n-NBDIRECTOS];
}

// Devuelve el número de bloque que corresponde a la posición actual dentro
//   del fichero, si es final de fichero reserva un byte más
long EFichero::reservarBloque() {
      // Si es final de fichero reserva memoria
      if (finalFichero()) {
            // Incrementa el tamaño del fichero
            inodo.tamano++;

            // Si ya no queda espacio en este bloque reservamos otro
            if (posicion % tamBloque == 0) {
                  // Reservamos un y obtenemos su número
                  long nBloque = sf->reservarBloque();

                  // Calcula el número de bloque relativo del fichero
                  long n = posicion/tamBloque;

                  // Asocia al bloque reservado al fichero
                  if (n<NBDIRECTOS)
                        inodo.bloquesDirectos[n] = nBloque;
                  else {
                        bloquesIndirectos[n-NBDIRECTOS] = nBloque;
                        // Si es necesario reservamos un bloque para
                        //   los bloques indirectos
                        if (n==NBDIRECTOS)
```

```
                              inodo.bloquesIndirectos =
                                    sf->reservarBloque();
                        }
                  }
            }

            // Llama a obtenerNBloque para saber el número de bloque actual
            return obtenerNBloque();
}


EFichero::EFichero(EafitSF* sf, long nInodo, InodoESF& inodo) : Fichero(sf) {
            EFichero::sf = sf;
            EFichero::nInodo = nInodo;
            EFichero::inodo = inodo;

            // Reserva memoria para los bloques indirectos, aunque no sea necesario
            bloquesIndirectos = (long*) new char[tamBloque];

            // Si es necesario lee el bloque de bloques indirectos
            if (inodo.tamano/tamBloque>=NBDIRECTOS)
                  sf->leerBloque(inodo.bloquesIndirectos,(char*)bloquesIndirectos);
}

EFichero::~EFichero() {
            if (bloquesIndirectos)
                  delete bloquesIndirectos;
}


long EFichero::es(SF* s, long nI) {
            if (sf==s && nInodo==nI)
                  return 1;
            return 0;
}

long EFichero::obtenerTamano() {
            return inodo.tamano;
}

long EFichero::finalFichero() {
            if (posicion>=inodo.tamano)
                  return 1;
            return 0;
}


long EFichero::abrir() {
            uso++;
            return uso;
}

long EFichero::cerrar() {
            // Reduce el uso del fichero
```

```
	uso--;

	// Si ya no lo usa nadies
	if (uso==0) {
		// Escribe el inodo
		sf->escribirInodo(inodo, nInodo);

		// Si es necesario escribe el bloque de bloques indirectos
		if (inodo.tamano/tamBloque>NBDIRECTOS)
			sf->escribirBloque(inodo.bloquesIndirectos,
					(char*)bloquesIndirectos);

		// Libera la memoria reservada en el constructor
		delete buffer;
		buffer = NULL;
		delete bloquesIndirectos;
		bloquesIndirectos = NULL;
	  }

	return uso;
}

void EFichero::borrar() {
	// Libera los bloques del fichero
	long i;
	for (i=0; i<inodo.tamano/tamBloque; i++) {
		if (i<NBDIRECTOS)
			sf->liberarBloque(inodo.bloquesDirectos[i]);
		else
			sf->liberarBloque(bloquesIndirectos[i-NBDIRECTOS]);
	}

	// Libera el bloque de bloques indirectos, si existe
	if (i>=NBDIRECTOS)
		sf->liberarBloque(inodo.bloquesIndirectos);
}

long EFichero::leer(char* dtno, long nDatos) {
	long i = 0;
	try {
		// Lee el bloque actual
		long nBloque = obtenerNBloque();
		sf->leerBloque(nBloque, buffer);

		// Lee todos los bytes que se puedan
		for (i=0; i<nDatos; i++) {
			// Si llegamos al final de fichero regresamos y
			//   devolvemos los bytes leidos
			if (posicion > inodo.tamano)
				return i;

			// Lee un byte y actualiza la posición
			dtno[i] = buffer[posicion%tamBloque];
			posicion++;
```

```
                // Si es necesario lee otro bloque
                if (posicion%tamBloque == 0) {
                      nBloque = obtenerNBloque();
                      sf->leerBloque(nBloque, buffer);
                }
          }
      }
      catch(...) {}

      // Devolvemos los bytes leidos
      return i;
}

long EFichero::escribir(char* fte, long nDatos) {
      long nBloque;

      for (long i=0; i<nDatos; i++) {
            // Lee el bloque actual (y si hace falta reserva uno)
            nBloque = reservarBloque();
            if (i==0 || posicion%tamBloque==0)
                  sf->leerBloque(nBloque, buffer);

            // Escribe un byte en el buffer
            buffer[posicion%tamBloque] = fte[i];
            posicion++;

            // Cuando el buffer se llena, escribe el bloque
            if (posicion%tamBloque == 0)
                  sf->escribirBloque(nBloque, buffer);
      }

      // Escribe el bloque
      sf->escribirBloque(nBloque, buffer);

      return nDatos;
}

long EFichero::saltar(long pos, long desde) {
      // Calcula la posicion nueva posicion
      if (desde==PRINCIPIO) {
            posicion = pos;
      } else if (desde==ACTUAL) {
            posicion += pos;
      } else {
            throw "posición inicial desconocida";
      }

      // Ajusta la posicion dentro de los limites del fichero
      if (posicion<0) {
            posicion = 0;
      } else if (posicion>inodo.tamano) {
            posicion = inodo.tamano;
      }
```

```
        return posicion;
}



void EDirectorio::crearEntrada(char* nombre, long inodo) {
      posicion = 0;
      EEntradaDirectorio entrada;

      // Busca una entrada libre
      while(leer((char*)&entrada, sizeof(entrada))==sizeof(entrada))
            if (entrada.nInodo < 0) {
                  posicion -= sizeof(entrada);
                  break;
            }

      // Crea la entrada
      strcpy(entrada.nombre, nombre);
      entrada.nInodo = inodo;

      // Escribe la entrada
      escribir((char*)&entrada, sizeof(entrada));
}

void EDirectorio::borrarEntrada(char* nombre) {
      posicion = 0;
      EEntradaDirectorio entrada;

      // Busca la entrada de fichero a borrar
      while(leer((char*)&entrada, sizeof(entrada))==sizeof(entrada))
            if (entrada.nInodo>=0 && strcmp(entrada.nombre, nombre)==0) {
                  // Libera la entrada
                  entrada.nInodo = -entrada.nInodo;
                  posicion -= sizeof(entrada);
                  escribir((char*)&entrada, sizeof(entrada));

                  return;
            }

      throw "ese fichero no existe";
}


void EDirectorio::borrar() {
      EEntradaDirectorio entrada;
      posicion = sizeof(entrada);

      // Si el directorio no está vacio genera un error
      while(leer((char*)&entrada, sizeof(entrada))==sizeof(entrada))
            if (entrada.nInodo>=0)
                  throw "el directorio no está vacio";

      EFichero::borrar();
```

```
}


long EDirectorio::idFichero(char* nombre) {
      posicion = 0;
      EEntradaDirectorio entrada;

      // Busca el fichero solicitado
      while(leer((char*)&entrada, sizeof(entrada))==sizeof(entrada))
            if (entrada.nInodo>=0 && strcmp(entrada.nombre, nombre)==0)
                  // Devuelve el número de inodo del fichero
                  return entrada.nInodo;

      throw "ese fichero no existe";
}

long EDirectorio::existeFichero(char* nombre) {
      posicion = 0;
      EEntradaDirectorio entrada;

      // Busca el fichero
      while(leer((char*)&entrada, sizeof(entrada))==sizeof(entrada))
            if (entrada.nInodo>=0 && strcmp(entrada.nombre, nombre)==0)
                  // Encontrado, devuelve cierto
                  return 1;

      // No se encontró, devuelve falso
      return 0;
}


Fichero* EDirectorio::crearFichero(char* nombre, char tipo) {
      // Comprueba que el fichero no exista
      if (existeFichero(nombre))
            throw "ese fichero ya existe";

      // Reserva un inodo para el fichero
      long nInodo = sf->reservarInodo();

      // Crea el inodo
      InodoESF inodo;
      inodo.tipo = tipo;
        inodo.tamano = 0;

      // Abre el fichero
      Fichero* fich;
      switch (tipo) {
            case NORMAL:
                  fich = new EFichero(sf, nInodo, inodo);
                  break;
            case DIRECTORIO:
                  fich = new EDirectorio(sf, nInodo, inodo);
                  // Crea la entrada al directorio padre
                  ((EDirectorio*)fich)->crearEntrada("..",
```

```
                                        EFichero::nInodo);
                break;
          default:
                throw "tipo de fichero desconocido";
     }

     // Crea la entrada en el directorio
     crearEntrada(nombre, nInodo);

     // Devuelve un puntero al fichero
     return fich;
}

void EDirectorio::borrarFichero(char* nombre) {
     // Obtiene el número de inodo del fichero
     long nInodo = idFichero(nombre);

     // Lee el inodo
     InodoESF inodo;
     inodo = sf->leerInodo(nInodo);

     // Abre el fichero
     Fichero* fich;
     switch (inodo.tipo) {
          case NORMAL:
                fich = new EFichero(sf, nInodo, inodo);
                break;
          case DIRECTORIO:
                fich = new EDirectorio(sf, nInodo, inodo);
                break;
          default:
                throw "tipo de fichero desconocido";
     }

     // Borra el contenido del fichero
     fich->borrar();

     delete fich;

     // Libera el inodo
     sf->liberarInodo(nInodo);

     // Libera la entrada del directorio
     borrarEntrada(nombre);
}

long EDirectorio::infoFichero(InfoFichero& info) {
     EEntradaDirectorio entrada;
     InodoESF inodo;

     // Lee una entrada ocupada del directorio
     while(leer((char*)&entrada, sizeof(entrada))==sizeof(entrada))
          if (entrada.nInodo>=0) {
                // Obtiene la información sobre la entrada
```

```
                strcpy(info.nombre, entrada.nombre);
                inodo = sf->leerInodo(entrada.nInodo);
                info.tipo = inodo.tipo;
                info.tamano = inodo.tamano;

                // Indica que encontró una entrada ocuapada
                return 1;
            }

        // Indica que ya no quedan más ficheros en este directorio
        return 0;
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    nucleo/ejec/ejec99.cpp
//    Implementación del formato ejecutable 99
// ----------------------------------------------------------------------

#include <ejec99.h>
#include <nucleo.h>


void Ejecutable99::ejecutar(long fichero, Contexto& contexto) {
      SFV& sfv = Nucleo::sfv;
      ManejadorMemoria& manejadorMemoria = Nucleo::manejadorMemoria;

      // Posiciona el puntero dentro del fichero para leer su cabecera
      sfv.saltar(fichero, 0, PRINCIPIO);

      // Comprueba que el fichero corresponde a este formato.
      long id;
      sfv.leer(fichero, &id, sizeof(id));
      if (id!=ID_EJECUTABLE_99)
            throw "formato de ejecutable desconocido";

      // Lee el resto de la cabecera:
      //   el inicio del programa
      sfv.leer(fichero, &(contexto.pc), sizeof(long));
      //   el tamano de la pila
      long tamPila;
      sfv.leer(fichero, &tamPila, sizeof(tamPila));

      // Calcula el tamano de la imagen
      long tamImagen = sfv.obtenerTamano(fichero) - 3*sizeof(long);

      // Inicializa la pila
      contexto.pila = manejadorMemoria.asignar(tamPila);
      contexto.sp = 0;

      // Lee la imagen del ejecutable
      char* buffer = new char[tamImagen];
      sfv.leer(fichero, buffer, tamImagen);

      // Inicializa el segmento de código y datos estáticos
      contexto.codigo = manejadorMemoria.asignar(tamImagen);
      manejadorMemoria.aUsuario(contexto.codigo, 0, buffer, tamImagen);
}
```

A.4 Entorno

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    entorno.cc
//    Este fichero tan solo contiene la función main.
// ----------------------------------------------------------------------


#include <curses.h>
#include <stdio.h>
#include <stdlib.h>

#include <entorno.h>
#include <cdisco.h>
#include <eafitsf.h>
#include <mv.h>
#include <nucleo.h>


InterpreteComandos Entorno::interpreteComandos;
Ensamblador Entorno::ensamblador;


void Entorno::crearDiscoNuevo() {
      clear();
      addstr("\nCREAR UN DISCO NUEVO\n\n");
      addstr("Elige el disco a crear (0-4)\n");
      addstr("Pulsa x para volver atras\n");
      refresh();

      char car;
      do {
            car = getch();
      } while ((car<'0' || car>'4') && car!='x' && car!='X');

      if (car=='x' || car=='X')
            return;

      try {
            MaqVirtual::crearDisco(car-'0', 1024, 1024);
            Nucleo::dispositivosBloque.iniDisco(car-'0');
      }
      catch (char* cad) {
            printw("Error, %s\n", cad);
            refresh();
      }
}

void Entorno::formatearDisco() {
```

```
	clear();
	addstr("\nFORMATEAR UN DISCO\n\n");
	addstr("Elige el disco a formatear (0-4)\n");
	addstr("Pulsa x para volver atras\n");
	refresh();

	char car;
	do {
		car = getch();
	} while ((car<'0' || car>'4') && car!='x' && car!='X');

	if (car=='x' || car=='X')
		return;

	try {
		ControladorBloque *cB = Nucleo::dispositivosBloque.obtenerControlador(car-'0');
		EafitSF sf(cB, 0);
		Nucleo::sfv.iniSF(cB, car-'0');
	}
	catch (char* cad) {
		printw("Error, %s\n", cad);
		refresh();
	}
}


void Entorno::iniciar() {
	// Inicializamos Curses
	initscr();
	noecho();
	cbreak();
	idlok(stdscr, TRUE);
	scrollok(stdscr, TRUE);


	char car;
	do {
		clear();
		addstr("\n****  SISTEMA OPERATIVO EAFITOS  ****\n\n");
		addstr("1- Interprete de Comandos\n");
		addstr("2- Crear un nuevo disco\n");
		addstr("3- Dar formato a un disco ya creado");
		addstr(" (Sistema de ficheros Eafit)\n");
		addstr("0- Salir\n");
		refresh();

		do {
			car = getch();
		} while (car<'0' || car>'3');

		switch (car) {
			case '0': break;
			case '1': interpreteComandos.iniciar();
				  break;
```

```
                case '2': crearDiscoNuevo();
                        break;
                case '3': formatearDisco();
            }
        } while (car!='0');

        addstr("Gracias por usar EafitOS\n");
        refresh();

        // Finalizamos Curses
        endwin();
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    entorno/ic.cpp
//    Implementa el interprete de comandos.
// ----------------------------------------------------------------------

#include <curses.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <ic.h>
#include <mv.h>
#include <nucleo.h>
#include <entorno.h>


#ifdef _LINUX_
#define SEPARADOR "/"
#endif
#ifdef _DOS_
#define SEPARADOR "\\"
#endif


void InterpreteComandos::iniciar() {
      SFV& sfv = Nucleo::sfv;

      // Busca el primer disco formateado que exista
      long i;
      for (i=0; i<MAX_NUM_SF; i++)
            if (sfv.activo(i)) {
                  // Inicializa la unidad y directorio actuales
                  unidad = i;
                  directorio = sfv.abrir(i, 0, "/");
                  break;
            }

      // Si no existe ningún disco formateado regresa
      if (i>=MAX_NUM_SF)
            return;

      // Limpia la pantalla
      clear();
      long y, x, yMaxima, xMaxima;
      getmaxyx(stdscr, yMaxima, xMaxima);

      // Bucle principal
      while (1) {
            printw("Eafitos (%d): ", unidad);
            refresh();
```

```
echo();
getnstr(linea, sizeof(linea));
noecho();
// Arreglo para el scroll
getyx(stdscr,y,x);
if (y==yMaxima-1)
      scroll(stdscr);

if (strlen(linea)==0)
      continue;

    char* cad = strtok(linea," ");
try {
      if (strcmp(cad,"salir")==0)
      // Sale del interprete de comandos
            return;
      else if (strcmp(cad,"disco")==0) {
      // Cambia de unidad
            // Obtiene el número de unidad
            cad = strtok(NULL," ");
            if (cad == NULL)
                  throw "falta el número de disco";

            // Comprueba que la unidad sea válida
            long n = atoi(cad);
            if (!(sfv.activo(n)))
                  throw "esa unidad no existe";

            // Actualiza la unidad y directorio actuales
            unidad = n;
            long dir = sfv.abrir(unidad, directorio, "/");
            sfv.cerrar(directorio);
            directorio = dir;
      } else if (strcmp(cad,"creadir")==0) {
      // Crea un directorio
            // Obtiene el nombre del directorio
            cad = strtok(NULL," ");
            if (cad == NULL)
                  throw "falta el nombre del directorio";

            // Crea el directorio
            long id = sfv.crear(unidad, directorio, cad,
                              DIRECTORIO);

            // Cierra el directorio
            sfv.cerrar(id);
      } else if (strcmp(cad,"cd")==0) {
      // Cambia de directorio
            // Obtiene el nombre del directorio
            cad = strtok(NULL," ");
            if (cad == NULL)
                  throw "falta el nombre del directorio";

            // Actualiza el directorio actual
```

```
      long id = sfv.abrir(unidad,directorio,cad);
      sfv.cerrar(directorio);
      directorio = id;
} else if (strcmp(cad,"poner")==0) {
// Copia un fichero del SO anfitrión a Eaftios
      // Obtiene el nombre del fichero
      cad = strtok(NULL," ");
      if (cad == NULL)
            throw "falta el nombre del fichero";

      // Abre el fichero en el SO anfitrión
      FILE* fte = fopen(cad,"rb");
      if (fte == NULL)
            throw "ese fichero no existe";

      // Obtiene el nombre del fichero sin la ruta
      char *cad2, *cad3;
      cad2 = strtok(cad, SEPARADOR);
      while (cad2) {
            cad3 = cad2;
            cad2 = strtok(NULL, SEPARADOR);
      }
      cad = cad3;

      // Crea el fichero en Eafitos
      long id = sfv.crear(unidad, directorio, cad,
                        NORMAL);

      // Copia el fichero
      char dato;
      while (!feof(fte)) {
            fread(&dato,sizeof(char),1,fte);
            sfv.escribir(id, &dato, 1);
      }

      // Cierra los ficheros
      sfv.cerrar(id);
      fclose(fte);
} else if (strcmp(cad,"obtener")==0) {
// Copia un fichero Eafitos al SO anfitrión
      // Obtiene el nombre del fichero
      cad = strtok(NULL," ");
      if (cad == NULL)
            throw "falta el nombre del fichero";

      // Abre el fichero en Eafitos
      long fte = sfv.abrir(unidad, directorio, cad);

      // Obtiene el nombre del fichero sin la ruta
      char cad2[100];
      strcpy(cad2, cad);
      char *cad3, *cad4;
      cad3 = strtok(cad2, "/");
      while (cad3) {
```

```
                cad4 = cad3;
                cad3 = strtok(NULL, "/");
        }
        cad = cad4;

        // Crea el fichero en el SO anfitrión
        FILE* dtno = fopen(cad,"wb");

        // Copia el fichero
        char dato;
        long nDatos;
        while (1) {
                nDatos = sfv.leer(fte, &dato, 1);
                if (!nDatos)
                        break;
                fwrite(&dato,sizeof(char),1,dtno);
        }

        // Cierra los ficheros
        sfv.cerrar(fte);
        fclose(dtno);
} else if (strcmp(cad,"borrar")==0) {
// Borra un fichero
        // Obtiene el nombre del fichero
        cad = strtok(NULL," ");
        if (cad == NULL)
                throw "falta el nombre del fichero";

        // Borra el fichero
        sfv.borrar(unidad,directorio,cad);
} else if (strcmp(cad,"dir")==0) {
// Lista el contenido de un directorio
        InfoFichero info;
        sfv.saltar(directorio, 0, PRINCIPIO);

        printw("Nombre              \tTipo\tTamaño\n");
        printw("------              \t----\t------\n");
        while (sfv.infoFichero(directorio, info)) {
                printw("%-20s\t", info.nombre);
                if (info.tipo==NORMAL)
                        printw("\t");
                else if (info.tipo==DIRECTORIO)
                        printw("<DIR>\t");
                else
                        printw("<???>\t");
                printw("%d\n", info.tamano);
        }
        refresh();
} else if (strcmp(cad,"ver")==0) {
// Muestra el contenido de un fichero
        // Obtiene el nombre del fichero
        cad = strtok(NULL," ");
        if (cad == NULL)
                throw "falta el nombre del fichero";
```

```
                        // Abre el fichero
                        long id = sfv.abrir(unidad,directorio,cad);

                        // Escribe el fichero en pantalla
                        char car;
                        while (sfv.leer(id, &car, 1)==1)
                              addch(car);
                        refresh();

                        // Cierra el fichero
                        sfv.cerrar(id);
                  } else if (strcmp(cad,"compilar")==0) {
                  // Compila un fichero
                        // Obtiene el nombre del fichero
                        cad = strtok(NULL," ");
                        if (cad == NULL)
                              throw "falta el nombre del fichero";

                        // Genera el nombre del fichero ejecutable
                        char obj[100];
                        strcpy(obj, cad);
                        strcat(obj, ".exe");

                        // Compila el fichero
                        Entorno::ensamblador.analizar(cad, obj, unidad,
                                                directorio);
                  } else {
                  // Ejecuta el programa
                        // Crea el hilo y el proceso
                        Nucleo::manejadorProcesos.ejecutar(unidad,
                                          directorio, cad);

                        // Lo ejecuta
                        MaqVirtual::cpu.ejecutar();
                  }
            }
            catch(char* cad) {
                  printw("Error, %s\n", cad);
                  refresh();
            }
      }
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    entorno/ensambla.cpp
//    Implementación el Ensamblador.
// ----------------------------------------------------------------------

#include <curses.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ensambla.h>
#include <nucleo.h>
#include <cpu.h>


PalabraClave palabrasClave[] = {
      {"DATOS", DATOS, 0}, {"CODIGO", CODIGO, 0},
      {"SUMAR", I_RRR, SUMAR}, {"RESTAR", I_RRR, RESTAR},
      {"AND", I_RRR, SUMAR}, {"OR", I_RRR, RESTAR},
      {"COPIAR", I_RR, COPIAR}, {"NOT", I_RR, RESTAR},
      {"CARGAR32", I_RR, CARGAR32}, {"GUARDAR32", I_RR, GUARDAR32},
      {"CARGAR8", I_RR, CARGAR8}, {"GUARDAR8", I_RR, GUARDAR8},
      {"CARGAR_I", I_RI, CARGAR_I}, {"GUARDAR_I", I_RI, GUARDAR_I},
      {"APILAR", I_R, APILAR}, {"DESAPILAR", I_R, DESAPILAR},
      {"SALTAR", I_I, SALTAR}, {"SALTARO", I_RI, SALTARO},
      {"SALTARP", I_RI, SALTARP}, {"SALTARN", I_RI, SALTARN},
      {"NOP", I_, NOP}, {"SER_SIS", I_, SER_SIS}
};

#define N_PALABRAS_CLAVE 22


long Ensamblador::siguienteToken() {
      SFV& sfv = Nucleo::sfv;

      while (1) {
            long i;
            char c;
            long fin = sfv.leer(ffte, &c, 1);

            if (!fin) {
                  token = FINAL_FICHERO;
                  return token;
            }
            if (c==';') {
                  while (1) {
                        fin = sfv.leer(ffte, &c, 1);
                        if (c=='\n' || !fin)
                              break;
```

```
      }
      if (fin)
            sfv.saltar(ffte, -1, ACTUAL);
      continue;
}
if (c=='\n') {
      linea++;
      while (1) {
            fin = sfv.leer(ffte, &c, 1);
            if (!fin)
                  break;
            if (isspace(c)) {
                  if (c=='\n')
                        linea++;
                  continue;
            }
            if (c==';') {
                  while (1) {
                        fin = sfv.leer(ffte, &c, 1);
                        if (c=='\n' || !fin)
                              break;
                  }
                  if (fin)
                        sfv.saltar(ffte, -1, ACTUAL);
                  continue;
            }
            sfv.saltar(ffte, -1, ACTUAL);
            break;
      }
      token = NUEVA_LINEA;
      return token;
}
if (isspace(c))
      continue;
if (c==',') {
      token = COMA;
      return token;
}
if (c=='@') {
      for (i=0; i<2; i++) {
            fin = sfv.leer(ffte, &c, 1);
            if (!fin)
                  break;
            if (isdigit(c))
                  lexema[i]=c;
            else {
                  sfv.saltar(ffte, -1, ACTUAL);
                  break;
            }
      }
      if (i==0)
            throw "falta número de registro";
      else {
            lexema[i]='\0';
```

```
                valor = atol(lexema);
                if (valor>=16)
                        throw "número de registro demasiado grande";
                token = REGISTRO;
                return token;
        }
}
if (c=='#') {
        for (i=0; i<12; i++) {
                fin = sfv.leer(ffte, &c, 1);
                if (!fin)
                        break;
                if (isdigit(c))
                        lexema[i]=c;
                else {
                        sfv.saltar(ffte, -1, ACTUAL);
                        break;
                }
        }
        if (i==0)
                throw "falta dato inmediato";
        else {
                lexema[i]='\0';
                valor = atol(lexema);
                token = LITERAL_NUMERICO;
                return token;
        }
}
if (c=='"') {
        for (i=0; i<100; i++) {
                fin = sfv.leer(ffte, &c, 1);
                if (!fin)
                        throw "fin de fichero inesperado";
                if (c=='"')
                        break;
                if (c=='\\') {
                        fin = sfv.leer(ffte, &c, 1);
                        if (!fin)
                                throw "fin de fichero inesperado";
                        if (c=='n')
                                c = '\n';
                        if (c=='t')
                                c = '\t';
                }
                lexema[i] = c;
        }
        if (i>=100)
                throw "cadena demasiado larga (máximo 100 caracteres)";
        lexema[i] = '\0';
        token = CADENA;
        return token;
}
if (isalpha(c)) {
        lexema[0] = toupper(c);
```

```
                for (i=1; i<20; i++) {
                      fin = sfv.leer(ffte, &c, 1);
                      if (!fin || (!isalnum(c) && c!='_')) {
                            lexema[i]='\0';
                            if (fin)
                                  sfv.saltar(ffte, -1, ACTUAL);
                            for (i=0; i<N_PALABRAS_CLAVE; i++)
                                  if (strcmp(lexema,palabrasClave[i].lexema)==0) {
                                        token = palabrasClave[i].token;
                                        valor = palabrasClave[i].valor;
                                        return token;
                                  }
                            token = ID;
                            return token;
                      }
                      lexema[i] = toupper(c);
                }
          }

          throw "caracter desconocido";
    }
}



void Ensamblador::nuevoId() {
      if (nIds>=100)
            throw "demasiados identificadores (máximo 100)";

      for (long i=0; i<nIds; i++)
            if (!strcmp(lexema,ids[i].lexema))
                  throw "declaración múltiple de un identificador";

      strcpy(ids[nIds].lexema, lexema);
      ids[nIds].direccion = direccion;
      nIds++;
}

long Ensamblador::obtenerDir(char* nombre) {
      for (long i=0; i<nIds; i++)
            if (strcmp(nombre, ids[i].lexema)==0)
                  return ids[i].direccion;

      throw "identificador no declarado";
}


void Ensamblador::programa() {
      SFV& sfv = Nucleo::sfv;

      // Escribe la cabecera del fichero ejecutable, formato 99 (2? pasada)
      if (pasada==1) {
            long cod;
            cod = 2323;
```

```
      sfv.escribir(fobj, &cod, 4);
      sfv.escribir(fobj, &dirInicio, 4);
      cod = 128;
      sfv.escribir(fobj, &cod, 4);
}

// Lee el primer token del programa
siguienteToken();

// Si son comentarios o líneas vacias, lee el siguiente token
if (token==NUEVA_LINEA)
      siguienteToken();

// Si hay sección de datos la procesa
if (token==DATOS) {
      siguienteToken();
      if (token!=NUEVA_LINEA)
            throw "nueva linea esperada";
      datos();
}

// Comprueba que la sección de código empice por "CODIGO NUEVA_LINEA"
if (token!=CODIGO)
      throw "CODIGO esperado";

siguienteToken();
if (token!=NUEVA_LINEA)
      throw "nueva linea esperada";

// Guarda la dirección de inicio del código para la cabecera
if (pasada==0)
      dirInicio = direccion;

// Procesa el código
siguienteToken();
codigo();

// Añade la instrucción implicita para finalizar el hilo (2? pasada)
if (pasada==0)
      direccion+=9;
else if (pasada==1) {
      char byte = CARGAR_I;
      sfv.escribir(fobj, &byte, 1);
      byte = 0;
      sfv.escribir(fobj, &byte, 1);
      long dato = 3;
      sfv.escribir(fobj, &dato, 4);
      byte = APILAR;
      sfv.escribir(fobj, &byte, 1);
      byte = 0;
      sfv.escribir(fobj, &byte, 1);
      byte = SER_SIS;
      sfv.escribir(fobj, &byte, 1);
}
```

```
}

void Ensamblador::datos() {
      siguienteToken();

      if (token!=CODIGO) {
            lineaDatos();

            siguienteToken();
            if (token!=NUEVA_LINEA)
                  throw "nueva linea esperada";

            datos();
      }
}

void Ensamblador::lineaDatos() {
      if (token!=ID)
            throw "identificador esperado";

      // Añadir la variable a la lista
      if (pasada==0)
            nuevoId();

      constante();
}

void Ensamblador::constante() {
      SFV& sfv = Nucleo::sfv;

      siguienteToken();

      if (token==LITERAL_NUMERICO) {
            if (pasada==0)
                  direccion+=4;
            else if (pasada==1)
                  sfv.escribir(fobj, &valor, 4);
      } else if (token==CADENA) {
            if (pasada==0)
                  direccion += strlen(lexema)+1;
            else if (pasada==1)
                  for (unsigned long i=0; i<=strlen(lexema); i++)
                        sfv.escribir(fobj, &lexema[i], 1);
      } else
            throw "constante esperada";
}

void Ensamblador::codigo() {
      lineaCodigo();

      if (token!=FINAL_FICHERO) {
            if (token!=NUEVA_LINEA)
                  throw "nueva linea esperada";
```

```
            siguienteToken();

            if (token!=FINAL_FICHERO)
                  codigo();
      }
}

void Ensamblador::lineaCodigo() {
      if (token==ID) {
            // Añadir la etiqueta a la lista
            if (pasada==0)
                  nuevoId();

            siguienteToken();
            if (token==NUEVA_LINEA || token==FINAL_FICHERO)
                  return;
      }

      instruccion();
      siguienteToken();
}

void Ensamblador::instruccion() {
      SFV& sfv = Nucleo::sfv;

      switch(token) {
            case I_:
                  // Escribimos el código de instrucción
                  if (pasada==1)
                        sfv.escribir(fobj, &valor, 1);
                  // Incrementamos la dirección de memoria
                  if (pasada==0)
                        direccion++;
                  break;
            case I_R:
                  // Escribimos el código de instrucción
                  if (pasada==1)
                        sfv.escribir(fobj, &valor, 1);
                  // Leemos el registro y lo escribimos
                  siguienteToken();
                  if (token!=REGISTRO)
                        throw "registro esperado";
                  if (pasada==1)
                        sfv.escribir(fobj, &valor, 1);
                  // Incrementamos la dirección de memoria
                  if (pasada==0)
                        direccion+=2;
                  break;
            case I_RR:
                  // Escribimos el código de instrucción
                  if (pasada==1)
                        sfv.escribir(fobj, &valor, 1);
                  // Leemos el primer registro y lo escribimos
                  siguienteToken();
```

```
            if (token!=REGISTRO)
                  throw "registro esperado";
            if (pasada==1)
                  sfv.escribir(fobj, &valor, 1);
            // Leemos la coma
            siguienteToken();
            if (token!=COMA)
                  throw "coma esperada";
            // Leemos el segundo registro y lo escribimos
            siguienteToken();
            if (token!=REGISTRO)
                  throw "registro esperado";
            if (pasada==1)
                  sfv.escribir(fobj, &valor, 1);
            // Incrementamos la dirección de memoria
            if (pasada==0)
                  direccion+=3;
            break;
      case I_RRR:
            // Escribimos el código de instrucción
            if (pasada==1)
                  sfv.escribir(fobj, &valor, 1);
            // Leemos el primer registro y lo escribimos
            siguienteToken();
            if (token!=REGISTRO)
                  throw "registro esperado";
            if (pasada==1)
                  sfv.escribir(fobj, &valor, 1);
            // Leemos la coma
            siguienteToken();
            if (token!=COMA)
                  throw "coma esperada";
            // Leemos el segundo registro y lo escribimos
            siguienteToken();
            if (token!=REGISTRO)
                  throw "registro esperado";
            if (pasada==1)
                  sfv.escribir(fobj, &valor, 1);
            // Leemos la coma
            siguienteToken();
            if (token!=COMA)
                  throw "coma esperada";
            // Leemos el tercer registro y lo escribimos
            siguienteToken();
            if (token!=REGISTRO)
                  throw "registro esperado";
            if (pasada==1)
                  sfv.escribir(fobj, &valor, 1);
            // Incrementamos la dirección de memoria
            if (pasada==0)
                  direccion+=4;
            break;
      case I_I:
            // Escribimos el código de instrucción
```

```
                if (pasada==1)
                      sfv.escribir(fobj, &valor, 1);
                // Leemos el dato inmediato y lo escribimos
                siguienteToken();
                if (pasada==1) {
                      if (token==ID)
                            valor = obtenerDir(lexema);
                      else if (token!=LITERAL_NUMERICO)
                            throw "identificador o número esperado";
                      sfv.escribir(fobj, &valor, 4);
                }
                // Incrementamos la dirección de memoria
                if (pasada==0)
                      direccion+=5;
                break;
          case I_RI:
                // Escribimos el código de instrucción
                if (pasada==1)
                      sfv.escribir(fobj, &valor, 1);
                // Leemos el registro y lo escribimos
                siguienteToken();
                if (token!=REGISTRO)
                      throw "registro esperado";
                if (pasada==1)
                      sfv.escribir(fobj, &valor, 1);
                // Leemos la coma
                siguienteToken();
                if (token!=COMA)
                      throw "coma esperada";
                // Leemos el dato inmediato y lo escribimos
                siguienteToken();
                if (pasada==1) {
                      if (token==ID)
                            valor = obtenerDir(lexema);
                      else if (token!=LITERAL_NUMERICO)
                            throw "identificador o número esperado";
                      sfv.escribir(fobj, &valor, 4);
                }
                // Incrementamos la dirección de memoria
                if (pasada==0)
                      direccion+=6;
                break;
          case FINAL_FICHERO:
                break;
          default:
                throw "instrucción desconocida";
     }
}


void Ensamblador::analizar(char* fte, char* obj, long unidad, long dir) {
     SFV& sfv = Nucleo::sfv;

     // Inicialización de algunas variables
```

```
	ffte = fobj = -1;
	linea = 1;
	pasada = 0;
	direccion = 0;
	nIds = 0;

	try {
		// Abrimos el fichero fuente
		ffte = sfv.abrir(unidad, dir, fte);
		if (ffte==-1)
			throw "no se pudo abrir el fichero fuente";

		// Primera pasada. Análisis léxico y sintáctico, obtención de
		// información para la segunda pasada
		programa();

		// Creamos el fichero objeto
		fobj = sfv.crear(unidad, dir, obj, 0);
		if (fobj==-1)
			throw "no se pudo crear el fichero objeto";

		// Segunda pasada. Análisis semántico y generación de código
		sfv.saltar(ffte, 0, PRINCIPIO);
		linea = 1;
		pasada = 1;
		programa();

		// Cerramos los ficheros
		sfv.cerrar(ffte);
		sfv.cerrar(fobj);
	}
	catch(char* msg) {
		// Cerramos los ficheros abiertos y borramos el código generado
		if (ffte>=0) {
			sfv.cerrar(ffte);
			if (fobj>=0) {
				sfv.cerrar(fobj);
				sfv.borrar(unidad, dir, obj);
			}
		}

		printw("Error en la linea %d, %s\n", linea, msg);
		refresh();
		return;
	}

	addstr("Código generado con éxito\n");
	refresh();
}
```

```
//    Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//    Este código es libre. Está protegido por la licencia GNU GPL.
//    Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//    eafitos2.cc
//    Este fichero tan solo contiene la función main.
// ----------------------------------------------------------------------


#include <mv.h>
#include <nucleo.h>
#include <entorno.h>


int main(void)
{
      MaqVirtual::iniciar();
      Nucleo::iniciar();
      Entorno::iniciar();
      Nucleo::terminar();
      MaqVirtual::terminar();

      return 0;
}
```

Apéndice B

Práctica: semáforos

B.1 Planteamiento

Eafitos es un sistema operativo multitarea, pero no dispone de ningún mecanismo de sincronización entre hilos. Esto convierte la multitarea en algo bastante inútil. El objetivo de esta práctica es implementar un mecanismo de sincronización con el cual sea fácil controlar el acceso a secciones críticas de código.

B.2 Solución

El mecanismo elegido para implementar son los semáforos.

Vamos a implementar dos nuevas llamadas al sistema que recibirán un solo parámetro. Ese parámetro será una dirección de memoria y actuará a modo de cerradura. Una llamada servirá para reservar una cerradura y la otra para liberarla. Cuando un hilo solicite una cerradura, si no está reservada se le concederá y seguirá ejecutándose, si está reservada el hilo pasará a suspendido hasta que la cerradura que solicitó sea liberada.

Para implementar esto primero crearemos dos nuevas clases llamadas **`Semaforo`** y **`Semaforos`** (encontraras su código en la Sección B.3.

Además, habrá que hacer otras tres pequeñas modificaciones:

- Añadir un atributo más a la clase **`Nucleo`** :

```
static Semaforos semaforos;
```

- Definir dos nuevas constantes en *include/nucleo.h*:

```
#define RESERVAR_CERRADURA 4
#define LIBERAR_CERRADURA 5
```

- Añadir dos entradas en el bloque **`switch`** del método **`Nucleo::llamada:`**

```
        case RESERVAR_CERRADURA:
                param1 = cpu.desapilar();
                Nucleo::semaforos.reservar(param1);
                break;
        case LIBERAR_CERRADURA:
                param1 = cpu.desapilar();
                Nucleo::semaforos.liberar(param1);
                break;
```

B.3 Código nuevo

```
//      Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//      Este código es libre. Está protegido por la licencia GNU GPL.
//      Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//      include/semaforo.h
//      Definición de los semáforos.
// ----------------------------------------------------------------------

#ifndef _SEMAFOROS
#define _SEMAFOROS

#include <cola.h>


#define N_SEMAFOROS 20


class Semaforo : public Cola {
        long cerradura;
public:
        Semaforo() { cerradura = -1; }

        long libre();
        long es(long c);
        void reservar(long c);
        void liberar();
};



class Semaforos : public Cola {
        Semaforo semaforos[N_SEMAFOROS];
public:
        void reservar(long cerradura);
        void liberar(long cerradura);
};
```

```
#endif
```

```
//      Copyright (C) 1998-1999 Juan David Ibáñez Palomar
//      Este código es libre. Está protegido por la licencia GNU GPL.
//      Mira el fichero Copyright para más detalles.
//
// ----------------------------------------------------------------------
//      include/semaforo.cpp
//      Implementación de los semáforos.
// ----------------------------------------------------------------------

#include <semaforo.h>
#include <nucleo.h>


long Semaforo::libre() {
        if (cerradura<0)
                return 1;
        return 0;
}

long Semaforo::es(long c) {
        if (cerradura==c)
                return 1;
        return 0;
}

void Semaforo::reservar(long c) {
        if (cerradura<0) {
                cerradura = c;
                return;
        }

        if (cerradura==c) {
                // Transición de EJECUCIÓN a SUSPENDIDO
                long hilo = Nucleo::manejadorProcesos.extraer();
                hilos[hilo].suspender();
                insertar(hilo);
                return;
        }

        throw "la cerradura no se corresponde";
}

void Semaforo::liberar() {
        if (libre())
                throw "este semáfaro no está reservado";

        try {
                // Transición de SUSPENDIDO a LISTO
                long hilo = extraer();
                hilos[hilo].reactivar(0);
                Nucleo::manejadorProcesos.insertar(hilo);
        }
        catch(...) {
                cerradura = -1;
```

```
        }
}


void Semaforos::reservar(long cerradura) {
        for (long i=0; i<N_SEMAFOROS; i++) {
                if (semaforos[i].es(cerradura)) {
                        semaforos[i].reservar(cerradura);
                        return;
                }
        }

        for (long i=0; i<N_SEMAFOROS; i++) {
                if (semaforos[i].libre()) {
                        semaforos[i].reservar(cerradura);
                        return;
                }
        }

        throw "no quedan cerraduras libres";
}

void Semaforos::liberar(long cerradura) {
        for (long i=0; i<N_SEMAFOROS; i++) {
                if (semaforos[i].es(cerradura)) {
                        semaforos[i].liberar();
                        return;
                }
        }

        throw "no quedan cerraduras libres";
}
```

B.4 Ejemplo

El código que viene a continuación es un ejemplo de programa de Eafitos que hace uso de los semáforos.

```
; Programa que crea dos hilos, cada uno de ellos imprime un mensaje y espera
;   a que el usuario pulse una tecla
DATOS
        cerradura #0
        padre   "Soy el padre: "
        hijo    "Soy el hijo: "
CODIGO
        ; crea un hilo
        cargar_i @1, hilo
        apilar @1
        cargar_i @2, #1
        apilar @2
        ser_sis

        cargar_i @2, padre
        saltar main

hilo    cargar_i @2, hijo


main    ; reserva la cerradura
        cargar_i @1, cerradura
        apilar @1
        cargar_i @1, #4
        apilar @1
        ser_sis


        ; imprime la cadena
        cargar_i @1, #31        ; código de llamada al sistema
        cargar_i @3, #1         ; desplazamiento para recorrer la cadena

bucle   cargar8 @4, @2
        saltar0 @4, fin

        apilar @4       ; llamada al sistema
        apilar @1
        ser_sis

        sumar @2, @3, @2
        saltar bucle


fin     ; libera la cerradura
```

```
cargar_i @1, cerradura
apilar @1
cargar_i @1, #5
apilar @1
ser_sis
```

Apéndice C

The GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

C.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

C.2 Terms and Conditions for Copying, Distribution, and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are

imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make

exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

C.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *<one line to give the program's name and a brief idea of what it does.>* Copyright ©19yy *<name of author>*

> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision
comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  This
is free software, and you are welcome to redistribute it under certain
conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <*signature of Ty Coon*>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Apéndice D

GNU GPL en castellano

LICENCIA PÚBLICA GENERAL DE GNU
Versión 2, Junio de 1991 [1]

D.1 Preámbulo

Las licencias que cubren la mayor parte del software están diseñadas para quitarle a usted la libertad de compartirlo y modificarlo. Por el contrario, la Licencia Pública General de GNU pretende garantizarle la libertad de compartir y modificar software libre, para asegurar que el software es libre para todos sus usuarios. Esta Licencia Pública General se aplica a la mayor parte del software del la Free Software Foundation y a cualquier otro programa si sus autores se comprometen a utilizarla. (Existe otro software de la Free Software Foundation que está cubierto por la Licencia Pública General de GNU para Bibliotecas). Si quiere, también puede aplicarla a sus propios programas.

Cuando hablamos de software libre, estamos refiriéndonos a libertad, no a precio. Nuestras Licencias Públicas Generales están diseñadas para asegurarnos de que tenga la libertad de distribuir copias de software libre (y cobrar por ese servicio si quiere), de que reciba el código fuente o que pueda conseguirlo si lo quiere, de que pueda modificar el software o usar fragmentos de él en nuevos programas libres, y de que sepa que puede hacer todas estas cosas.

Para proteger sus derechos necesitamos algunas restricciones que prohiban a cualquiera negarle a usted estos derechos o pedirle que renuncie a ellos. Estas restricciones se traducen en ciertas obligaciones que le afectan si distribuye copias del software, o si lo modifica.

[1] Traduccida al catellano por Jesús M. González, jgb@gsyc.inf.uc3m.es.

Por ejemplo, si distribuye copias de uno de estos programas, sea gratuitamente, o a cambio de una contraprestación, debe dar a los receptores todos los derechos que tiene. Debe asegurarse de que ellos también reciben, o pueden conseguir, el código fuente. Y debe mostrarles estas condiciones de forma que conozcan sus derechos.

Protegemos sus derechos con la combinación de dos medidas:

1. Ponemos el software bajo copyright y
2. le ofrecemos esta licencia, que le da permiso legal para copiar, distribuir y/o modificar el software.

También, para la protección de cada autor y la nuestra propia, queremos asegurarnos de que todo el mundo comprende que no se proporciona ninguna garantía para este software libre. Si el software se modifica por cualquiera y éste a su vez lo distribuye, queremos que sus receptores sepan que lo que tienen no es el original, de forma que cualquier problema introducido por otros no afecte a la reputación de los autores originales.

Por último, cualquier programa libre está constantemente amenazado por patentes sobre el software. Queremos evitar el peligro de que los redistribuidores de un programa libre obtengan patentes por su cuenta, convirtiendo de facto el programa en propietario. Para evitar esto, hemos dejado claro que cualquier patente debe ser pedida para el uso libre de cualquiera, o no ser pedida.

Los términos exactos y las condiciones para la copia, distribución y modificación se exponen a continuación.

D.2 Términos y condiciones para la copia, distribución y modificación

0. Esta Licencia se aplica a cualquier programa u otro tipo de trabajo que contenga una nota colocada por el tenedor del copyright diciendo que puede ser distribuido bajo los términos de esta Licencia Pública General. En adelante, "Programa" se referirá a cualquier programa o trabajo que cumpla esa condición y "trabajo basado en el Programa" se referirá bien al Programa o a cualquier trabajo derivado de él según la ley de copyright. Esto es, un trabajo que contenga el programa o una proción de él, bien en forma literal o con modificaciones y/o traducido en otro lenguaje. Por lo tanto, la traducción está incluida sin limitaciones en el término "modificación". Cada concesionario (licenciatario) será denominado "usted".

   Cualquier otra actividad que no sea la copia, distribución o modificación no está cubierta por esta Licencia, está fuera de su ámbito. El acto de ejecutar el Programa no está restringido, y los resultados del Programa están cubiertos únicamente si sus contenidos constituyen un trabajo basado en el Programa, independientemente de haberlo producido mediante la ejecución del programa. El que esto se cumpla, depende de lo que haga el programa.

1. Usted puede copiar y distribuir copias literales del código fuente del Programa, según lo has recibido, en cualquier medio, supuesto que de forma adecuada y bien visible publique en cada copia un anuncio de copyright adecuado y un repudio de garantía, mantenga intactos todos los anuncios que se refieran a esta Licencia y a la ausencia de garantía, y proporcione a cualquier otro receptor del programa una copia de esta Licencia junto con el Programa.

   Puede cobrar un precio por el acto físico de transferir una copia, y puede, según su libre albedrío, ofrecer garantía a cambio de unos honorarios.

2. Puede modificar su copia o copias del Programa o de cualquier porción de él, formando de esta manera un trabajo basado en el Programa, y copiar y distribuir esa modificación o trabajo bajo los términos del apartado 1, antedicho, supuesto que además cumpla las siguientes condiciones:

   a. Debe hacer que los ficheros modificados lleven anuncios prominentes indicando que los ha cambiado y la fecha de cualquier cambio.
   b. Debe hacer que cualquier trabajo que distribuya o publique y que en todo o en parte contenga o sea derivado del Programa o de cualquier parte de él sea licenciada como un todo, sin carga alguna, a todas las terceras partes y bajo los términos de esta Licencia.
   c. Si el programa modificado lee normalmente órdenes interactivamente cuando es ejecutado, debe hacer que, cuando comience su ejecución para ese uso interactivo de la forma más habitual, muestre o escriba un mensaje que incluya un anuncio de copyright y un anuncio de que no se ofrece ninguna garantía (o por el contrario que sí se ofrece garantía) y que los usuarios pueden redistribuir el programa bajo estas condiciones, e indicando al usuario cómo ver una copia de esta licencia. (Excepción: si el propio programa es interactivo pero normalmente no muestra ese anuncio, no se requiere que su trabajo basado en el Programa muestre ningún anuncio).

   Estos requisitos se aplican al trabajo modificado como un todo. Si partes identificables de ese trabajo no son derivadas del Programa, y pueden, razonablemente, ser consideradas trabajos independientes y separados por ellos mismos, entonces esta Licencia y sus términos no se aplican a esas partes cuando sean distribuidas como trabajos separados. Pero cuando distribuya esas mismas secciones como partes de un todo que es un trabajo basado en el Programa, la distribución del todo debe ser según los términos de esta licencia, cuyos permisos para otros licenciatarios se extienden al todo completo, y por lo tanto a todas y cada una de sus partes, con independencia de quién la escribió.

   Por lo tanto, no es la intención de este apartado reclamar derechos o desafiar sus derechos sobre trabajos escritos totalmente por usted mismo. El intento es ejercer el derecho a controlar la distribución de trabajos derivados o colectivos basados en el Programa.

   Además, el simple hecho de reunir un trabajo no basado en el Programa con el Programa (o con un trabajo basado en el Programa) en un volumen de almacenamiento o en un medio de distribución no hace que dicho trabajo entre dentro del ámbito cubierto por esta Licencia.

3. Puede copiar y distribuir el Programa (o un trabajo basado en él, según se especifica en el apartado 2, como código objeto o en formato ejecutable según los términos de los apartados 1 y 2, supuesto que además cumpla una de las siguientes condiciones:

   a. Acompañarlo con el código fuente completo correspondiente, en formato electrónico, que debe ser distribuido según se especifica en los apartados 1 y 2 de esta Licencia en un medio habitualmente utilizado para el intercambio de programas, o

   b. Acompañarlo con una oferta por escrito, válida durante al menos tres años, de proporcionar a cualquier tercera parte una copia completa en formato electrónico del código fuente correspondiente, a un coste no mayor que el de realizar físicamente la distribución del fuente, que será distribuido bajo las condiciones descritas en los apartados 1 y 2 anteriores, en un medio habitualmente utilizado para el intercambio de programas, o

   c. Acompañarlo con la información que recibiste ofreciendo distribuir el código fuente correspondiente. (Esta opción se permite sólo para distribución no comercial y sólo si usted recibió el programa como código objeto o en formato ejecutable con tal oferta, de acuerdo con el apartado b anterior).

   Por código fuente de un trabajo se entiende la forma preferida del trabajo cuando se le hacen modificaciones. Para un trabajo ejecutable, se entiende por código fuente completo todo el código fuente para todos los módulos que contiene, más cualquier fichero asociado de definición de interfaces, más los guiones utilizados para controlar la compilación e instalación del ejecutable. Como excepción especial el código fuente distribuido no necesita incluir nada que sea distribuido normalmente (bien como fuente, bien en forma binaria) con los componentes principales (compilador, kernel y similares) del sistema operativo en el cual funciona el ejecutable, a no ser que el propio componente acompañe al ejecutable.

   Si la distribución del ejecutable o del código objeto se hace mediante la oferta acceso para copiarlo de un cierto lugar, entonces se considera la oferta de acceso para copiar el código fuente del mismo lugar como distribución del código fuente, incluso aunque terceras partes no estén forzadas a copiar el fuente junto con el código objeto.

4. No puede copiar, modificar, sublicenciar o distribuir el Programa excepto como prevé expresamente esta Licencia. Cualquier intento de copiar, modificar sublicenciar o distribuir el Programa de otra forma es inválida, y hará que cesen automáticamente los derechos que te proporciona esta Licencia. En cualquier caso, las partes que hayan recibido copias o derechos de usted bajo esta Licencia no cesarán en sus derechos mientras esas partes continúen cumpliéndola.

5. No está obligado a aceptar esta licencia, ya que no la ha firmado. Sin embargo, no hay hada más que le proporcione permiso para modificar o distribuir el Programa o sus trabajos derivados. Estas acciones están prohibidas por la ley si no acepta esta Licencia. Por lo tanto, si modifica

o distribuye el Programa (o cualquier trabajo basado en el Programa), está indicando que acepta esta Licencia para poder hacerlo, y todos sus términos y condiciones para copiar, distribuir o modificar el Programa o trabajos basados en él.

6. Cada vez que redistribuya el Programa (o cualquier trabajo basado en el Programa), el receptor recibe automáticamente una licencia del licenciatario original para copiar, distribuir o modificar el Programa, de forma sujeta a estos términos y condiciones. No puede imponer al receptor ninguna restricción más sobre el ejercicio de los derechos aquí garantizados. No es usted responsable de hacer cumplir esta licencia por terceras partes.

7. Si como consecuencia de una resolución judicial o de una alegación de infracción de patente o por cualquier otra razón (no limitada a asuntos relacionados con patentes) se le imponen condiciones (ya sea por mandato judicial, por acuerdo o por cualquier otra causa) que contradigan las condiciones de esta Licencia, ello no le exime de cumplir las condiciones de esta Licencia. Si no puede realizar distribuciones de forma que se satisfagan simultáneamente sus obligaciones bajo esta licencia y cualquier otra obligación pertinente entonces, como consecuencia, no puede distribuir el Programa de ninguna forma. Por ejemplo, si una patente no permite la redistribución libre de derechos de autor del Programa por parte de todos aquellos que reciban copias directa o indirectamente a través de usted, entonces la única forma en que podría satisfacer tanto esa condición como esta Licencia sería evitar completamente la distribución del Programa.

   Si cualquier porción de este apartado se considera inválida o imposible de cumplir bajo cualquier circunstancia particular ha de cumplirse el resto y la sección por entero ha de cumplirse en cualquier otra circunstancia.

   No es el propósito de este apartado inducirle a infringir ninguna reivindicación de patente ni de ningún otro derecho de propiedad o impugnar la validez de ninguna de dichas reivindicaciones. Este apartado tiene el único propósito de proteger la integridad del sistema de distribución de software libre, que se realiza mediante prácticas de licencia pública. Mucha gente ha hecho contribuciones generosas a la gran variedad de software distribuido mediante ese sistema con la confianza de que el sistema se aplicará consistentemente. Será el autor/donante quien decida si quiere distribuir software mediante cualquier otro sistema y una licencia no puede imponer esa elección.

   Este apartado pretende dejar completamente claro lo que se cree que es una consecuencia del resto de esta Licencia.

8. Si la distribución y/o uso de el Programa está restringida en ciertos países, bien por patentes o por interfaces bajo copyright, el tenedor del copyright que coloca este Programa bajo esta Licencia puede añadir una limitación explícita de distribución geográfica excluyendo esos países, de forma que la distribución se permita sólo en o entre los países no excluidos de esta manera. En ese caso, esta Licencia incorporará la limitación como si estuviese escrita en el cuerpo de esta Licencia.

9. La Free Software Foundation puede publicar versiones revisadas y/o nuevas de la Licencia Pública General de tiempo en tiempo. Dichas nuevas

versiones serán similares en espíritu a la presente versión, pero pueden ser diferentes en detalles para considerar nuevos problemas o situaciones.

Cada versión recibe un número de versión que la distingue de otras. Si el Programa especifica un número de versión de esta Licencia que se refiere a ella y a "cualquier versión posterior", tienes la opción de seguir los términos y condiciones, bien de esa versión, bien de cualquier versión posterior publicada por la Free Software Foundation. Si el Programa no especifica un número de versión de esta Licencia, puedes escoger cualquier versión publicada por la Free Software Foundation.

10. Si quiere incorporar partes del Programa en otros programas libres cuyas condiciones de distribución son diferentes, escribe al autor para pedirle permiso. Si el software tiene copyright de la Free Software Foundation, escribe a la Free Software Foundation: algunas veces hacemos excepciones en estos casos. Nuestra decisión estará guiada por el doble objetivo de de preservar la libertad de todos los derivados de nuestro software libre y promover el que se comparta y reutilice el software en general.

AUSENCIA DE GARANTÍA

11. COMO EL PROGRAMA SE LICENCIA LIBRE DE CARGAS, NO SE OFRECE NINGUNA GARANTÍA SOBRE EL PROGRAMA, EN TODA LA EXTENSIÓN PERMITIDA POR LA LEGISLACIÓN APLICABLE. EXCEPTO CUANDO SE INDIQUE DE OTRA FORMA POR ESCRITO, LOS TENEDORES DEL COPYRIGHT Y/U OTRAS PARTES PROPORCIONAN EL PROGRAMA "TAL CUAL", SIN GARANTÍA DE NINGUNA CLASE, BIEN EXPRESA O IMPLÍCITA, CON INCLUSIÓN, PERO SIN LIMITACIÓN A LAS GARANTÍAS MERCANTILES IMPLÍCITAS O A LA CONVENIENCIA PARA UN PROPÓSITO PARTICULAR. CUALQUIER RIESGO REFERENTE A LA CALIDAD Y PRESTACIONES DEL PROGRAMA ES ASUMIDO POR USTED. SI SE PROBASE QUE EL PROGRAMA ES DEFECTUOSO, ASUME EL COSTE DE CUALQUIER SERVICIO, REPARACIÓN O CORRECCIÓN.

12. EN NINGÚN CASO, SALVO QUE LO REQUIERA LA LEGISLACIÓN APLICABLE O HAYA SIDO ACORDADO POR ESCRITO, NINGÚN TENEDOR DEL COPYRIGHT NI NINGUNA OTRA PARTE QUE MODIFIQUE Y/O REDISTRIBUYA El PROGRAMA SEGÚN SE PERMITE EN ESTA LICENCIA SERÁ RESPONSABLE ANTE USTED POR DAÑOS, INCLUYENDO CUALQUIER DAÑO GENERAl, ESPECIAL, INCIDENTAL O RESULTANTE PRODUCIDO POR EL USO O LA IMPOSIBILIDAD DE USO DEL PROGRAMA (CON INCLUSIÓN, PERO SIN LIMITACIÓN A LA PÉRDIDA DE DATOS O A LA GENERACIÓN INCORRECTA DE DATOS O A PÉRDIDAS SUFRIDAS POR USTED O POR TERCERAS PARTES O A UN FALLO DEL PROGRAMA AL FUNCIONAR EN COMBINACIÓN CON CUALQUIER OTRO PROGRAMA), INCLUSO SI DICHO TENEDOR U OTRA PARTE HA SIDO ADVERTIDO DE LA POSIBILIDAD DE DICHOS DAÑOS.

FIN DE TÉRMINOS Y CONDICIONES

D.3 Cómo aplicar estos términos a sus nuevos programas

Si usted desarrolla un nuevo Programa, y quiere que sea del mayor uso posible para el público en general, la mejor forma de conseguirlo es convirtiéndolo en software libre que cualquiera pueda redistribuir y cambiar bajo estos términos.

Para hacerlo, añada los siguientes anuncios al programa. Lo más seguro es añadirlos al principio de cada fichero fuente para transmitir lo más efectivamente posible la ausencia de garantía. Además cada fichero debería tener al menos la línea de "copyright" y un indicador a dónde puede encontrarse el anuncio completo.

> *<una línea para indicar el nombre del programa y una rápida idea de qué hace.>* Copyright (C) 19aa *<nombre del autor>*
>
> Este programa es software libre. Puede redistribuirlo y/o modificarlo bajo los términos de la Licencia Pública General de GNU según es publicada por la Free Software Foundation, bien de la versión 2 de dicha Licencia o bien (según su elección) de cualquier versión posterior.
>
> Este programa se distribuye con la esperanza de que sea útil, pero SIN NINGUNA GARANTÍA, incluso sin la garantía MERCANTIL implícita o sin garantizar la CONVENIENCIA PARA UN PROPÓSITO PARTICULAR. Véase la Licencia Pública General de GNU para más detalles.
>
> Debería haber recibido una copia de la Licencia Pública General junto con este programa. Si no ha sido así, escriba a la Free Software Foundation, Inc., en 675 Mass Ave, Cambridge, MA 02139, EEUU.

Añada también información sobre cómo contactar con usted mediante correo electrónico y postal.

Si el programa es interactivo, haga que muestre un pequeño anuncio como el siguiente, cuando comienza a funcionar en modo interactivo:

```
Gnomovision versión 69, Copyright (C) 19aa nombre del autor

Gnomovision no ofrece ABSOLUTAMENTE NINGUNA GARANTÍA. Para más detalles
escriba ‘‘show w’’.

Esto es software libre, y se le invita a redistribuirlo bajo ciertas
condiciones.  Escriba ‘‘show c’’ para más detalles.
```

Los comandos hipotéticos "show w" y "show c" deberían mostrar las partes adecuadas de la Licencia Pública General. Por supuesto, los comandos que use pueden llamarse de cualquier otra manera. Podrían incluso ser pulsaciones del ratón o elementos de un menú (lo que sea apropiado para su programa).

También deberías conseguir que su empleador (si trabaja como programador) o tu Universidad (si es el caso) firme un "renuncia de copyright" para el programa, si es necesario. A continuación se ofrece un ejemplo, altere los nombres según sea conveniente:

> Yoyodyne, Inc. mediante este documento renuncia a cualquier interés de derechos de copyright con respecto al programa Gnomovision (que hace pasadas a compiladores) escrito por Pepe Programador.
>
> <*firma de Pepito Grillo*>, 20 de diciembre de 1996 Pepito Grillo, Presidente de Asuntillos Varios.

Esta Licencia Pública General no permite que incluya sus programas en programas propietarios. Si su programa es una biblioteca de subrutinas, puede considerar más útil el permitir el enlazado de aplicaciones propietarias con la biblioteca. Si este es el caso, use la Licencia Pública General de GNU para Bibliotecas en lugar de esta Licencia.