Trabajo práctico: desarrollo de un sistema de archivos en c++

Sofia C. Arancibia
Facultad de Ingeniería y Ciencias Hídricas, *sofi.aran_05@hotmail.com*

***Resumen*—Este documento es una especificación del diseño y principales características de un sistema de archivos desarrollado en c++.**

***Palabras clave*—inodo, bloque, indices, enlaces**

I. Introducción

La aplicación presentada es un simulador de sistma de archivos. Consiste en un bloque principal que contiene el bucle que se ejecutará hasta que el ususario finalice la aplicación. Tal como lo hace la shell de linux, esta aplicación imprime un prompt ("%") en cada ciclo del bucle principal indicando que está a la espera de una instrucción. La estructura modular de la aplicación facilita la lectura del código de la misma, ya que cada módulo ejecuta una única tarea. Se utilizan las librerías wait, unistd, stdio, cstdio, cerrno, entre otras, para utilizar funciones de gestión de errores, ejecución de comandos (execvp),creación de subprocesos (fork), etc.

II. fuentes

Para facilitar la lectura, estructurado y depuración del proyecto, se utilizan funciones con objetivos específicos y únicos dentro de la aplicación.

A. main

Como se mencionó anteriormente, aquí se ejecuta el bucle principal. Este bucle sólo finaliza cuando el usuario finaliza la aplicación. En cada ciclo del bucle, se imprime el prompt y la aplicación queda a la espera de una intrucción. El usuario ingresa la instrucción que desea ejecutar como un string, para independizar al programa de la longitud de la instrucción. Éste string se convierte en cadena de caracteres para luego realizar el análisis necesario para la ejecución del comando. Utilizando la función *s_inst()*, se crea un arreglo de palabras, cuyo primer elemento se tomará como comando y los subsiguientes como argmentos del mismo. Si el comando no es "exit" el siguiente paso será ejecutar la instrucción. Adicional a los comandos propios de la manipulación de archivos, se consideran comandos especiales:

- h : Mostrar ayuda. Se invoca la función help() la cual simplemente imprime en la pantalla una lista de los comandos disponibles.
- a : Iniciar sesión como administrador.

B. Guardar()

El prototipo de esta función es:

void guardar(FILE archivo,inodo &in)*

La función no devuelve nada, por lo cual se ha declarado como *void*, y sus parámetros de entrada son un puntero al tipo FILE y un inodo pasado por referencia el cual contendrá la información resepectiva al archivo/directorio.

El objetivo de este módulo es guardar el archivo recién leido en una estructura de bloques de datos. Para ello se leen porciones del archivo del tamaño de los bloques de datos. Esta lectura se realiza tantas veces como bloques de datos requiera el archivo. Cada uno de estos bloques de 1kb de capacidad, y la cantidad de bloques necesarios se realiza diviendo el tamaño total del archivo por el tamaño de un bloque de datos. El archivo no necesariamente tendrá un tamaño múltiplo de 1kb, por lo cual, la división anterior podría generar un resto, el cual se guarda en un bloque de datos adicional. Es necesario conservar los bloques que este archivo ocupa, para luego poder consultarlo o eliminarlo según fuera el caso. Para ello, y emulando el comportamiento de sistema de archivos real, se utilizan enlaces directos, indirectos simples e indirectos dobles. Este sistema de enlaces se comporta como indica la Fig.1. Los enlaces utilizados por el archivo se guardan dentro del inodo correspondiente a dicho archivo, junto con información relevante del archivo como su nombre y propietario.

C. buscar_inodo()

El prototipo de esta función es:

int buscar_inodo(string nombre)

Esta función simplemente retorna el indice del inodo, cuyo nombre está dado por *nombre,* dentro del vector global de inodos. Si no lo encuentra (porque no se ha creado o se ha borrado el archivo), devuelve un -1 que indica esta situación.

D. buscar_sig()

El prototipo de esta función es:

int buscar_sig(string que)

Esta función retorna el índice del siguiente objeto disponibles en el vector de objetos correspondiente. Los objetos que se pueden buscar con esta rutina son inodos, bloques de datos o enlaces. Al igual que en la función anterior, se devuelve un índice en caso de encontrar disponibilidad, o -1 caso contrario.

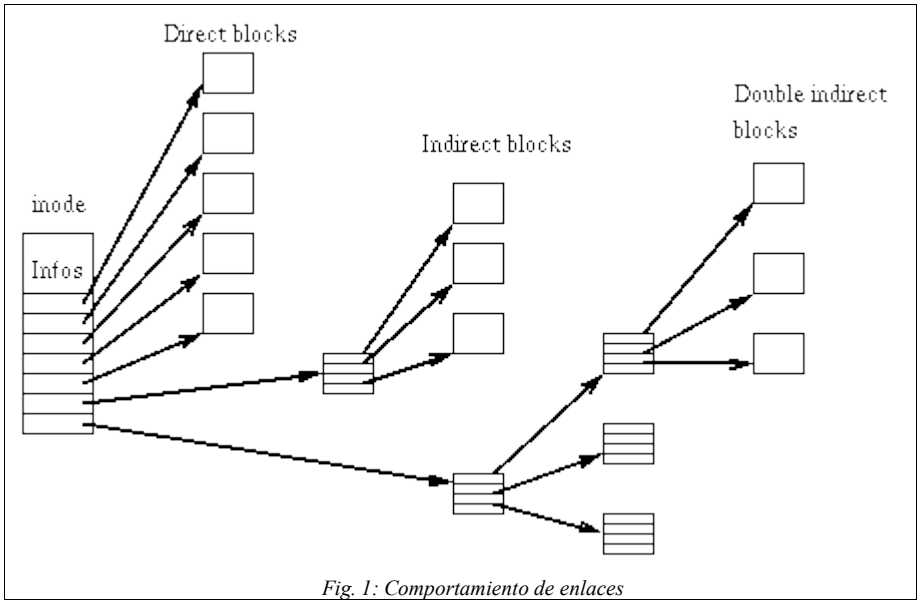

Fig. 1: Comportamiento de enlaces

E. eliminar_archivo() / eliminar_directorio()

Los prototipos de estas funciones son:

void eliminar_archivo(int indx)
void eliminar_directorio(char name)*

Ambas funciones eliminan archivos/directorios y todas las estructuras asociadas. En el caso de los archivos, se utiliza el vector de enlaces presente en su inodo (cuyo indice se indica como argumento) para vaciar los bloques de datos que el archivo ha utilizado, luego se elimina el inodo asociado. Para los directorios, se proporciona el nombre del directorio a eliminar, se busca el inodo asociado en el vector global de inodos y posteriormente se elimina su contenido. Los inodos asociados a directorios disponen de un vector de hijos, el cual contiene los números (identificadores) de los inodos correspondiente a los archivos que contiene el directorio. El borrado en este caso se realiza recorriendo este vector de hijos e invocando a *eliminar_archivo()* o *eliminar_directorio()* según el inodo represente a un archivo o a un subdirectorio.

F. crear_archivo()

El prototipo de esta función es:

void crear_archivo(string destino, char nombre, inodo &in)*

Esta función se encarga de crear fisicamente y virtualmente un archivo. Como es usual, se lee por pantalla el contenido del archivo, en este caso se lee hasta la pulsación de [enter], y se introduce dicho contenido en un archivo de texto, el cual se guardará "virtualmente" dentro del directorio indicado por *destino,* con el nombre *nombre*, y todos sus datos relevantes se devolveran en el inodo pasado como argumento por referencia.

G. owner()

El prototipo de esta función es:

string owner(char s_a)*

Esta función hace uso del comando Linux *stat* , el cual devuelve información sobre un archivo. En este caso se utilizó de modo tal que devuelva el nombre del propietario de un archivo. Se realiza un analisis sintáctico del resultado de este comando para conservar sólo el nombre del usuario, el cual luego se utilizará para determinar permisos.

Se utlizan además estructuras de datos adicionales como lo son *bloque* e *inodo*. Las cuales representan los objetos bloque de datos e inodo utilizados para guardar los archivos y los metadatos de los mismos respectivamente.