ITBA

- TPE 3 -

Filesystem versionado

Arolfo Franco
Mozzino Jorge

Introducción

Para el presente trabajo nos proponemos a implementar un sistema de archivos en nuestro sistema operativo.
Se pide la implementación de los comandos standard de UNIX (rm, cp, mv, ln, …), además de un sistema de “snapshots” para poder guardar las modificaciones de un archivo y poder luego volver a la versión anterior.
Dicho sistema de archivos debe poder persistir en el disco duro del equipo.

Implementación

Diseño del Sistema de Archivos

Se decidió ir por un formato similar al de UNIX en cuanto al diseño del filesystem.
Contamos con un primer código de verificación, del cual veremos su uso más adelante, la estructura de entrada del directorio root, una zona de mapa de bits, donde decimos qué bloques están libres, una zona de inodos y, finalmente, los bloques para el uso de los archivos.

Utilizamos un código de verificación inicial por si es que el disco donde se está cargando el filesystem en la etapa de booteo no posee el formato anteriormente dicho. Así, si durante la inicialización del sistema, se lee dicho código de verificación, no se formatea el disco. En caso contrario, sí lo hace.

Para saber que bloques del disco están libres y cuáles no, utilizamos un mapa de bits, en el que indexamos por el número de bloque y verificamos si éste está o no libre.

Para el guardado de archivos y directorios decidimos utilizar estructuras de inodos. Estos se componen por:

- Atributos: contienen el tamaño del archivo, nombre, fecha de última modificación, la versión previa del archivo (esto se utilizará luego para snapshots), el número de inodo de su padre (si es que es un directorio, sino es -1) y si está libre u ocupado.
- Un arreglo de índices, el cual me lista los números de bloques utilizados por el archivo en orden, terminando en -1.

Como no contamos con un “mapa de bits de inodos libres”, así como en el caso de los bloques, debemos valernos de la variable en los atributos para saber si un inodo está ocupado o no. Para una mejor performance durante la ejecución del sistema, se optó por cargar en el inicio del sistema, un mapa de bits en memoria, el cual me indica que inodos están libres.

En cuanto a la diseño de los directorios, optamos por una lista de estructuras del tipo Entry, las cuales poseen el nombre del archivo, el número del inodo al que refiere el archivo y si es que se trata de un link o no. Cuando se crea un directorio, se le cargan por default las entradas del directorio actual (“.”) con su mismo número de inodo, y la del padre(“..”) con el número de inodo correspondiente.

Implementación de comandos

Se implementaron los comandos básicos de un filesystem:

- **cat [nombre] [secuencia de caracteres]**: sirve para concatenar al final de un archivo una secuencia de caracteres. Si el archivo no existe, se lo crea. De esta forma, para crear archivos vacíos se puede llamar a cat sin una secuencia de caracteres para concatenar.
- **mkdir [nombre del directorio]:** sirve para crear un nuevo directorio
- **ls [nombre del directorio]:** sirve para listar el contenido de un directorio. Si no se dice un nombre, se toma el directorio corriente. Debido a que el sistema permite que el usuario recupere archivos borrados, se listan todos los archivos, junto con su tamaño y estado (donde estado es borrado o vivo). El formato con el que se lista es el siguiente:

  Nombre | Tamaño | Estado
- **cp [fuente] [destino]:** sirve para copiar el contenido de fuente en destino. No es recursivo, es decir que copiar un directorio hace un shallow copy, los archivos de adentro quedan linkeados.
- **mv [fuente] [destino]:** sirve para mover el contenido de fuente a destino. También sirve para renombrar archivos.
- **rm [nombre de archivo]:** sirve para remover un archivo o directorio. Sin embargo, se guarda la información de dicho archivo para que el usuario lo pueda recuperar. Si se desea eliminar el archivo para siempre se debe usar forcerm. No se puede hacer rm de un archivo, si ya existe uno muerto con ese nombre.
- **forcerm [nombre de archivo]:** sirve para remover un archivo definitivamente. No se puede deshacer y no queda guardado nada del archivo.
- **ln [destino] [fuente]:** sirve para linkear un archivo a otro. De esa forma, si se modifica el link, se modifica el archivo fuente y viceversa. Si se borra un archivo que tiene un link y se trata de acceder al link el comportamiento no está definido.
- **revert [archivo] [número de revisión]:** Sirve para volver a una versión anterior de un archivo. Se creará una nueva versión del archivo con el contenido de la versión vieja.
- **history [archivo]:** sirve para listar hasta 200 revisiones anteriores del archivo.

Snapshots

Para los snapshots, se mantiene en cada inodo una referencia al inodo anterior. De esta forma, cada vez que se escribe en un inodo, se copian los bloques que se modificarán en un inodo nuevo (la referencia a los bloques que no se modifican queda igual). Entonces, queda formada un lista simplemente enlazada de inodos.
En un principio, se había pensado que los snapshots iban a ser parte de los atributos de un inodo, pero esto limitaba en gran cantidad la "memoria" de snapshots. Debido a esta limitación, se decidió que cada snapshot fuera un inodo separado.
También se crea un snapshot cada vez que se hace un move: se guarda un snapshot para mantener el nombre anterior.

Uso en disco

El disco se fragmentó de la siguiente forma:

La primera parte es un bloque de testeo, en el que se guarda la palabra mágica y la entrada del root.
Al primer bloque le siguen 4 bloques de un mapa de bits que indican cuáles bloques están disponibles. Así, se mapean los últimos 8MB del disco (se usa un disco de 10MB).
A continuación, se usan casi 2MB (ya que se usaron 5 bloques para otros temas) de inodos.
Preparamos los inodos de forma que ocupen un espacio de 1 bloque de disco cada uno.
De esta forma, se permite que un archivo esté compuesto por hasta 109 bloques (cada bloque es de 3,5KB).

Driver de disco

Se usó una única función para acceder al disco. La misma recibe como parámetros la cantidad de bloques a leer, el buffer donde tomar o dejar la información, la acción a realizar (lectura o escritura) y qué sector (numerado según LPA).
Al principio, surgieron problemas con el disco ya que se le estaban pasando caracteres con signo, y cuando alguno empezaba en 1, el disco rellenaba con unos, de forma que al querer levantar lo que se guardó se obtenían inconsistencias. Esto se solucionó al hacer que se guarden sin signo los caracteres.