Ejercicio 1 - Sistemas de archivos

Manuel Panichelli LU 072/18

Enunciado

Para un sistema de archivos ext2, se requiere que construya un algoritmo con el que se implemente la función buscar(string A, string B), la cual recibe como parámetros dos strings. El primer parámetro indica una ruta absoluta en el sistema de archivos, y el segundo corresponde a la cadena a buscar. Esta función debe mostrar como salida, todos los archivos y/o carpetas cuyo nombre coincida con B (el segundo parámetro). La búsqueda debe hacerse incluyendo todos los subdirectorios que se pueden encontrar a partir de A (el primer parámetro).

Para esto pueden utilizar las siguientes estructuras y funciones:

- Las estructuras de Ext2FS:
  - Ext2FSSuperblock (Superblock)
  - Ext2FSBlockGroupDescriptor (Block Group Descriptor)
  - Ext2FSInode (Inode)
  - Ext2FSDirEntry (Directory Entry)
- Las funciones de Ext2FS:
  - unsigned int Ext2FS::get_block_address(struct Ext2FSInode * inode, unsigned int block_number): Devuelve la dirección del bloque.
  - struct Ext2FSInode * Ext2FS::load_inode(unsigned int inode_number): Carga un inodo.
  - struct Ext2FSInode * Ext2FS::get_file_inode_from_dir_inode(struct Ext2FSInode * from, const char * filename): Si existe, obtiene el único inodo de un archivo desde el inodo de un directorio. Si no, devuelve NULL.
  - struct Ext2FSInode * Ext2FS::inode_for_path(const char * path): Obtiene el inodo desde una ruta o path dado.
  - void Ext2FS::read_block(unsigned int block_address, unsigned char * buffer): Lee un bloque de disco.
  - struct Ext2FSSuperblock * Ext2FS::superblock(): Devuelve el superbloque.
  - struct Ext2FSBlockGroupDescriptor * Ext2FS::block_group(unsigned int index): Devuelve el descriptor del bloque de grupo.
  - unsigned int Ext2FS::blockgroup_for_inode(unsigned int inode): Número de blockgroup del Inodo.
  - unsigned int Ext2FS::blockgroup_inode_index(unsigned int inode): Offset dentro de la tabla de Inodos, para el inodo.

Si requiere usar funciones adicionales, debe explicar su función, debe justificar su uso y debe implementarlas.

Resolucion

Queremos implementar una version simple de `find`

```
# A = abspath
# B = name
buscar(string abspath, string name)
```

Busca desde `abspath` todos los archivos, y printea todos los archivos y/o directorios cuyo nombre coincida con `name`.

Lo mas simple que se me ocurre es hacerlo de forma recursiva, el pseudocodigo entonces seria (y despues lo hago mas formal)

```
def buscar(abspath, name):
    # Chequeamos si el nombre coincide y en caso de que si
    # lo printeamos. Esto pasa tanto para archivos como
    # para directorios.
    if name == name_from_path(abspath):
        print(name)

    if dir es un file:
        # llegamos a nuestro caso base, estamos parados
        # sobre un file, no hay mas que buscar a partir
        # de este punto
        return

    # dir no es un file, sino un directorio, busco recursivamente en
    # todo lo que haya dentro de el
    for subdir in dir: # abuso de notacion tremendo
        buscar_desde(subdir, name)
```

Y ahora pasemos a lo complicado: implementar este algoritmo en el fs ext2.

```
#include <string.h>     // strcmp

// Defines robados del taller
#define INODE_ISDIR(inode) (inode->mode & EXT2_S_IFDIR)
#define DENT_ISDIR(dent) (dent->file_type & EXT2_FT_DIR)
#define BLOCK_SIZE(logSize) (1024 << logSize)


// Supongo que tengo una refrencia (en este caso global por simplicidad)
// al filesystem, para poder llamar a las funciones que dice el enunciado.
Ext2FS fs;

void buscar(char* abspath, char* name) {
    // Cargo el inodo del path que me dieron
    Ext2FSInode* inode = fs.inode_for_path(abspath);

    // Busco recursivamente a partir de este punto
    buscar_desde(inode);
}

// funcion auxiliar para buscar desde cierto inodo en adelante
void buscar_desde(Ext2FSInode* inode, char* name) {
    // Si no es un directorio, no hay mas cosas a partir de este punto.
    if !INODE_ISDIR(inode) { return }

    // Como estamos dentro de un directorio, tenemos que ver en sus
    // bloques todas las Ext2FSDirEntry y por cada una, si el nombre
    // coincide printearlo, seguir buscando recursivamente.
    // Recorremos todos los bloques, cada uno puede tener multiples
    // dir entries adentro, dependiendo del tamaño de ambos.
    unsigned int block_size = BLOCK_SIZE(superblock()->log_block_size);
    unsigned int total_entries = inode->size / sizeof(Ext2FSDirEntry)
    unsigned int entries_per_block = block_size / sizeof(Ext2FSDirEntry);

    for (i = 0; i < total_entries; i += entries_per_block) {
        // Interpreto el contenido del bloque i-esimo como
        // un arreglo de dir entries.
        unsigned char buff[block_size];
        read_block(get_block_address(inode, i), buff);
        Ext2FSDirEntry* entries = (Ext2FSDirEntry*) buff;

        // Recorro los dir entries, teniendo cuidado de no pasarme de la
        // cantidad total de entries. En cada loop, i tendra el primer dir
        // entry del bloque, luego sumando j + i tenemos el entry actual
        // "global" y podemos ver de no pasarnos del maximo.
        for (int j = 0; j < entries_per_block && j + i < total_entries; j++){
            // Si el nombre matchea con lo que buscamos, lo printeamos
            if (strcmp(entries[j].name, name)) {
                print(name)
            }

            // Sigo buscando recursivamente desde aqui
            buscar_desde(load_inode(dirEntries[j].inode));
        }
    }
}
```