Laboratorio 4: Censurando un sistemas de archivos

FaMAFyC - Sistemas operativos 2020

Consigna

Revisiones

- 2019 y 2020 Milagro Teruel
- 2018 Cristian Cardellino, Milagro Teruel
- Tomamos mucho de:
    - 2012, Nicolás Wolovick
    - Original 2009-2010, Rafael Carrascosa, Nicolás Wolovick

Sistemas de archivos

Antes de avanzar con las tareas, vamos a dar un poco de contexto:

El *sistema de archivos* es una de las tantas abstracciones (exitosas) que presenta el sistema operativo para manejar recursos. Nacidas originalmente como una interfaz para los medios de almacenamiento permanente, evolucionaron rápidamente hasta convertirse en una parte fundamental de los sistemas operativos.

En los 70's, UNIX [KR74] fue diseñado alrededor de la idea *todo es un archivo*. Esto puso al filesystem en un lugar mucho más importante que una interfaz cómoda e independiente del hardware, que servía solo para manejar dispositivos de almacenamiento permanente.

En este diseño, aparecen *archivos especiales*, como `/dev/mt`, `/dev/mem`, `/dev/tty0` que brindaban la misma interfaz para objetos a priori disímiles: cintas, memoria y líneas seriales. Esto posibilitó que los programas de usuario tuvieran acceso a dispositivos que de otra manera resultaban complicados de programar. Una vez más las Ciencias de la Computación encontraban una abstracción correcta a fin de solucionar un problema recurrente y ampliar el alcance de los mecanismos de cómputo.

Otro aporte de diseño importante fue el *sistemas de archivos removibles*, donde en un solo árbol de directorios se podían *montar* varios dispositivos con sistemas de archivos independientes, brindando una interfaz abstracta, uniforme y cómoda de usar.

La idea de los *mountable FS* permitió que diferentes formatos en disco de sistemas de archivos pudieran ser montados en la misma jerarquía de directorios. Bastaba que los desarrolladores del kernel programaran un driver que leyera el formato en disco deseado, para que todos los programas de usuario se beneficiaran con la posibilidad de leer, por ejemplo, una vieja partición de un disco Winchester en formato previo al BSD Unix Fast File System [MJLF84]. También son últiles para montar discos remotos a través de una red y con formatos completamente nuevos, como por ejemplo montar un filesystem que permita leer archivos de Google Drive.

A medida que los formatos en disco proliferaban y nacían los *network file systems* (NFS), se hizo necesario estructurar mejor el código del sistema de archivos para que solo una pequeña parte deba ser agregada a fin de leer y escribir en los formatos de disco que median entre el sistema de archivos y el dispositivo. Así nació el *virtual file system* o *virtual filesystem switch* (VFS) en SunOS 2.0 de 1985, a fin de acceder de manera transparente tanto a volúmenes UFS como a sistemas de archivos remotos NFS.

Cuando nació Linux la idea del VFS estaba establecida. Esto significa soporte para aproximadamente *70 sistemas de archivos distintos*, lo que coloca a Linux en un lugar privilegiado en cuanto a compatibilidad.

El concepto de VFS fue llevado al extremo, copiando el principio de Plan9: "*todo es un filesystem*".

FUSE

Con esta forma de modularizar la estructura interna del Filesystem resulta sencillo agregar un nuevo FS al kernel, basta con implementar el módulo correspondiente siguiendo los lineamiento de cualquiera de ellos y cargarlo. Sin embargo, el código que desarrollamos corre dentro del espacio kernel y cualquier falla involucra posiblemente un reinicio del mismo. Esto sin contar con la escasez de herramientas de debugging y el hecho de tener que involucrarse con las estructuras internas del kernel, por lo que todo el proceso de desarrollo se vuelve dificultoso.

FUSE o Filesystem in USErspace, es un módulo más del VFS que permite implementar filesystems a nivel de userspace. Las ventajas son varias [Singh06]:

- Interfaz sencilla: aísla de ciertas complejidades inherentes al desarrollo dentro del kernel.
- Estabilidad: al estar el FS en userspace, cualquier error se subsana desmontando el sistema de archivos.
- Posibilidad de debuggear: tenemos todas las herramientas userland para atacar los bugs.

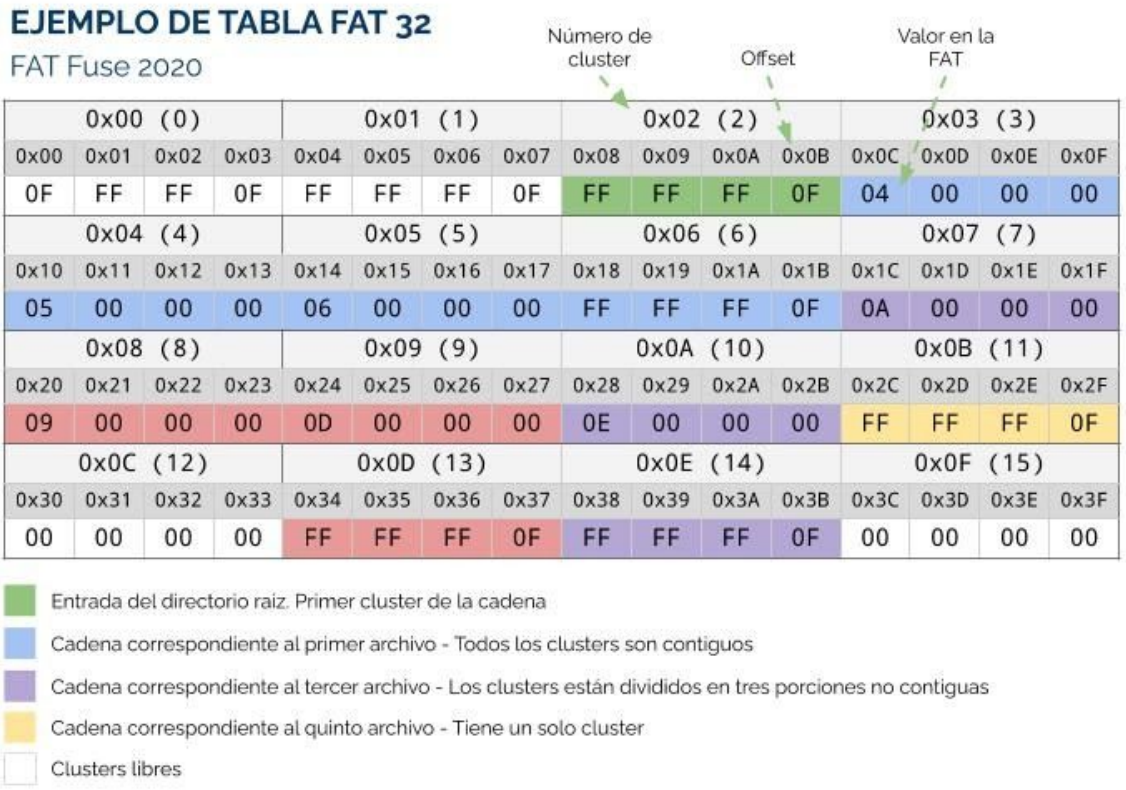
- Variedad de language bindings: se puede hacer un FUSE con Python, Perl, C, Haskell, sh, etc. Sí, lo repetimos, se puede hacer un sistema de archivos en Python, Haskell o con el shell.

Sistema de archivos FAT

El sistema de archivos FAT está resumido en la correspondiente página de Wikipedia que les recomendamos que lean. Aquí sólo les daremos un paseo por los conceptos más importantes. Los sistemas FAT tienen las siguientes partes principales:

- Área de sectores reservada, donde se encuentra la información que define el tipo de sistema FAT y sus propiedades, como el número de tablas FAT, el tamaño de cada cluster y el número total de clusters. Para este laboratorio podemos abstraernos de todo esto.
- La tabla FAT (File Allocation Table), que contiene información sobre qué clusters de datos están siendo utilizados y cuáles pertenecen a cada archivo o directorio. Usualmente hay dos copias de la tabla FAT para redundancia de datos.
- El directorio root, que está separado del resto de los directorios sólo en FAT12 y FAT16. En el protocolo FAT32, el directorio raíz está almacenado con el resto de los datos.
- El área de datos, que se divide en los clusters descritos en la FAT. Aquí se encuentran efectivamente los contenidos de los archivos y las tablas de los directorios.

Clusters



0x00 (0)				0x01 (1)				0x02 (2)				0x03 (3)			
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0F	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	04	00	00	00
0x04 (4)				0x05 (5)				0x06 (6)				0x07 (7)			
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
05	00	00	00	06	00	00	00	FF	FF	FF	0F	0A	00	00	00
0x08 (8)				0x09 (9)				0x0A (10)				0x0B (11)			
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27	0x28	0x29	0x2A	0x2B	0x2C	0x2D	0x2E	0x2F
09	00	00	00	0D	00	00	00	0E	00	00	00	FF	FF	FF	0F
0x0C (12)				0x0D (13)				0x0E (14)				0x0F (15)			
0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x3A	0x3B	0x3C	0x3D	0x3E	0x3F
00	00	00	00	FF	FF	FF	0F	FF	FF	FF	0F	00	00	00	00

Ejemplo de tabla FAT. [Fuente original]

El área de datos se divide en clusters de tamaño idéntico, que son pequeños bloques de disco continuos. El tamaño varía dependiendo del tipo de FAT, desde 2K a 32MB.

Cada archivo puede ocupar uno o más de estos clusters. Por lo tanto, cada archivo es representado como una lista ordenada de los clusters que contienen su información (*linked list*), almacenada en la FAT. Sin embargo, estos clusters no necesariamente se encuentran en espacios contiguos en el disco, y cada archivo reserva nuevos clusters a medida que va creciendo. (¿Se acuerdan de defragmentar Windows? Es por esto, ya que un mismo archivo puede estar desperdigado en distintos sectores del disco, ralentizando el proceso de lectura. La defragmentación reubica los clusters de un archivo en espacios continuos).

Si el archivo es un directorio, entonces su cluster contiene una lista de entradas de directorio donde se especifica el nombre de los archivos (y subdirectorios) que están contenidos en él, con sus correspondientes metadatos. Cada entrada de directorio es de tamaño fijo y están almacenadas en espacios contiguos de memoria. Por ende,

podemos saber que ya hemos leído todas las entradas del directorio cuando encontramos el primer archivo con el nombre vacío.

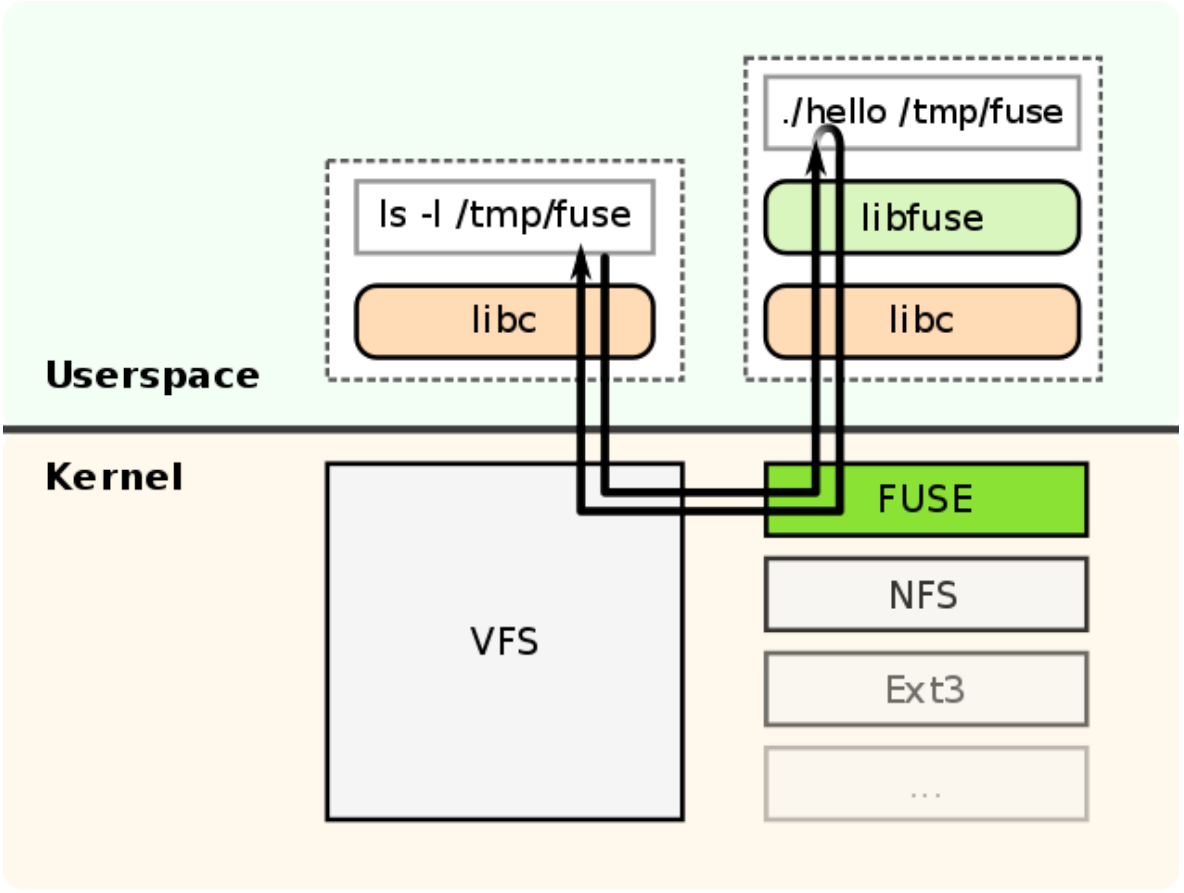
Tabla FAT

La FAT es una tabla donde cada cluster está representado por 12, 16 o 32 bits información contiguos (FAT12, FAT16 y FAT32 respectivamente). Esto también define el número máximo de clusters del sistema. En esta implementación sólo extenderemos las operaciones de escritura para FAT16 y FAT32.

Los valores posibles para una entrada de la FAT son:

- 0x?0000: indica que el cluster está libre
- 0x?0001: indica que el cluster está reservado para el sistema
- de 0x?0002 a 0x?FFEF: indica el siguiente cluster en la cadena para un archivo dado.
- 0x?FFF0 a 0x?FFF6 están reservados.
- 0x?FFF7: indica sector dañado en algunos sistemas.
- 0x?FFF8 - 0x?FFFF: indica el final de la cadena de clusters.

FAT FUSE



Abstracción del funcionamiento de FUSE

FUSE exporta la interfaz de un filesystem a userspace mediante una serie de funciones que definimos en la estructura `struct fuse operations`, muy parecida a `struct file operations` que se utiliza en los LKM. Estas funciones son las que definen de qué forma se organizará el sistema de archivo, cómo serán las funciones de lectura y escritura, etc. FUSE hace de interfaz entre nuestra implementación, en modo usuario, y las llamadas a sistema que realiza el usuario. De esta manera, aseguramos que el código que escribamos sólo accede al volumen sobre el que tenemos permisos, ya que el único código que se ejecuta en espacio kernel es el de FUSE. Por ejemplo, si un programa de usuario quiere abrir un archivo, ejecuta la llamada a sistema open. El sistema operativo, al ver que el volumen está montado al VFS con FUSE, le transfiere la llamada a sistema y FUSE busca en la estructura `fuse_operations` la función correspondiente que nosotros hemos implementado para efectivamente realizar la apertura del archivo.

Con esta arquitectura, FUSE es lo suficientemente flexible como para permitir cualquier tipo de sistema de archivos, de acuerdo a cómo están descritas en las operaciones de `fuse_operations`. En este sistema en particular, hemos implementado dichas operaciones para que sean compatibles con el sistema de archivos FAT, pero se podría

utilizar cualquier otro protocolo, como se muestra en este ejemplo. Incluso puede haber operaciones que no estén definidas, como por ejemplo en un sistema de archivos de solo lectura, sin operaciones de escritura.

Entrega

Requisitos necesarios para aprobar:

- Utilizar el repositorio de Bitbucket, con commits pequeños y nombres significativos. Todos los integrantes del grupo deben colaborar.
- Utilizar clang-format para mantener un estilo de código consistente.
- Pasar los tests provistos por la cátedra.

Consigna

En este laboratorio, vamos a modificar un sistema de archivos FAT en espacio de usuario para agregar:

- Una lista de palabras que son censuradas por el sistema de archivos.
- Operaciones de escritura de archivos que requieran agregar nuevos clusters.
- Otras funciones para la escritura de archivos existentes.

La implementación original que tomamos como base está en este repositorio, aunque hemos introducido muchos cambios.

Parte 1: La censura

Deben modificar el sistema original para que, al leer un archivo, reemplace todas ocurrencias de las siguientes palabras por el caracter 'X'.

```
{"bourgeois", "BOURGEOIS", "Bourgeois", "bourgeoisieclass",
"BOURGEOISIECLASS", "Bourgeoisieclass", "communism", "COMMUNISM",
"Communism", "communist", "COMMUNIST", "Communist", "communists",
"COMMUNISTS", "Communists", "communistic", "COMMUNISTIC",
"Communistic", "conservative", "CONSERVATIVE", "Conservative",
"country", "COUNTRY", "Country", "countries", "COUNTRIES",
"Countries", "family", "FAMILY", "Family", "free", "FREE", "Free",
"freedom", "FREEDOM", "Freedom", "history", "HISTORY", "History",
"historical", "HISTORICAL", "Historical", "justice", "JUSTICE",
"Justice", "moral", "MORAL", "Moral", "philosophy", "PHILOSOPHY",
"Philosophy", "philosophical", "PHILOSOPHICAL", "Philosophical",
"politic", "POLITIC", "Politic", "politics", "POLITICS",
"Politics", "political", "POLITICAL", "Political", "proletarian",
"PROLETARIAN", "Proletarian", "proletariat", "PROLETARIAT",
"Proletariat", "radical", "RADICAL", "Radical", "religion",
"RELIGION", "Religion", "religions", "RELIGIONS", "Religions",
"religious", "RELIGIOUS", "Religious", "social", "SOCIAL",
"Social", "socialism", "SOCIALISM", "Socialism", "socialist",
"SOCIALIST", "Socialist", "socialists", "SOCIALISTS",
"Socialists", "society", "SOCIETY", "Society", "value", "VALUE",
"Value", "world", "WORLD", "World", "work", "WORK", "Work",
"working-class", "WORKING-CLASS", "Working-class", “Friedrich”,
“Engels”}
```

Ejemplo

Si montamos nuestro filesystem con cfuse, y tratamos de leer el manifiesto comunista, veremos:

```
MANIFESTO OF THE xxxxxxxxx PARTY
[From the English edition of 1888, edited by XXXXXXXXX XXXXXX]
A spectre is haunting Europe--the spectre of xxxxxxxxx. All the
Powers of old Europe have entered into a holy alliance to exorcise
this spectre: Pope and Czar, Metternich and Guizot, French
Radicals and German police-spies.
Where is the party in opposition that has not been decried as
xxxxxxxxxxx by its opponents in power?  Where is the Opposition
that has not hurled back the branding reproach of xxxxxxxxx,
against the more advanced opposition parties, as well as against
its reactionary adversaries?
Two things result from this fact.
```

```
I.  xxxxxxxxx is already acknowledged by all European Powers to be
itself a Power.
```

Algunas consideraciones

- Las palabras deben estar contenidas en un arreglo. Pueden definir el arreglo de palabras y otras constantes en un archivo externo (e.g. censorship.h).
- Es más fácil de lo que parece, no tienen que modificar la forma en que se leen los datos sino simplemente el resultado que se muestra al usuario.
- Les dejamos un archivo `test.sh` de donde pueden tomar como ejemplo comandos para probar que el sistema funciona. No es extensivo, puede suceder que los test pasen pero el código todavía tenga errores.

Parte 2: Escribiendo nuevos clusters

La implementación base presentada sólo permite operaciones de escritura sólo en los cluster existentes.

Por ejemplo, supongamos el volumen tienen clusters de 512 bytes, y que un archivo FILE.TXT ocupa 800 bytes. Luego, FILE.TXT tiene reservados 2 clusters de datos. Si queremos escribir 500 bytes desde un offset de 700, entonces necesitamos agregar un cluster más, ya que 500 + 700 > 512 * 2. La función fat_file_pwrite, tal y como se las entregamos, sólo escribirá los bytes que entren en los clusters disponibles, es decir 512 * 2 - 500 - 700.

La tarea es modificar la función para que agregue un nuevo cluster al archivo, y realice la escritura completa.

La función fat_file_pwrite es bastante compleja. Les recomendamos que se tomen un tiempo para entender el código, así cómo el código de fat_table.c. Las funciones que leen y escriben los clusters ya están implementadas en fat_table.c, ¡usen las!

Parte 3: Registro de actividades

Vamos a crear un archivo secreto fs.log en donde se guardará un registro de todas las operaciones de lectura y escritura de archivos realizada por el usuario. Al agregar archivos, deberán familiarizarse con los TADs utilizados para representar en memoria los archivos (fat_file) y el árbol de directorios (fat_fs_tree). Pueden encontrar más información acerca de las implementaciones en el README.

Tarea 1: Creando un archivo oculto para el usuario

La primera tarea es efectivamente crear el archivo y lograr que el usuario no pueda verlo con comandos como ls. El archivo tiene que ser creado luego de montar el volumen, y tienen que ser cuidadosos de no crearlo varias veces. Este punto se puede realizar modificando únicamente fat_fuse_ops.c. Esto nos permite que los TADs de nuestro FS sean (casi) agnósticos a la implementación de la censura.

Al desmontar y volver a montar el volumen, el archivo tiene que ser leído correctamente.

Tarea 2: Escribiendo los logs

La segunda tarea es escribir en el archivo secreto un registro de las operaciones que realiza el usuario. Sólo deben registrar llamadas a read o write. Les proveemos del código que arma el string para escribir en la función fat_fuse_log_activity, pero deben completarla con la lógica que realiza la escritura.

Tarea 1: Ocultando el archivo de otros FS

Para que la censura sea verdaderamente transparente, no se debe alterar el formato FAT de ninguna manera. La imagen debe poder ser montada con otras herramientas, pero tenemos que lograr que ignoren el archivo fs.log, pero que no lo sobreescriban tampoco (en la mayoría de los casos, nunca lo podremos ocultar al 100%). Para esto, les proponemos que el archivo fs.log tenga las siguientes características:

- Que el primer byte de su dentry sea 0xe5. Esto marca el archivo como “pendiente para ser eliminado”, de forma que el FS ignora la entrada, pero no la reutiliza hasta que haya sido propiamente eliminada por otras herramientas.
- Que el campo attribute de la dentry sea System, Device o Reserved, lo cual indica que la entrada no debe ser modificada por las herramientas del FS. De esta manera, logramos que la entrada nunca sea efectivamente eliminada y marcada como libre.

Esta combinación de características poco frecuentes es suficiente para permitirles identificar la entrada de directorio correspondiente al archivo secreto de logs.

Para implementar esta funcionalidad deben:

- Modificar la entrada de directorio del archivo secreto al crearlo.
- Asegurar que su implementación no ignore la entrada de directorio, que ahora está marcada para ser borrada.
- Asegurarse de que al desmontar y volver a montar no se cambie el nombre del archivo secreto.

Nota: en el archivo fat_filename_util.c ya tienen implementada la funcionalidad que crea el filepath y name a partir de una entrada eliminada.

Algunas recomendaciones

- Utilicen generosamente las funciones de logging, como fat_table_print o fat_tree_print_preorder para ayudarse a debuggear.
- Concéntrense sólo en las partes relevantes del problema, y no en los detalles de bajo nivel.
- El sistema no es perfecto y probablemente tenga muchos errores. Si encuentran algo que parece sospechoso, avisen a la cátedra para que lo revisemos.

Puntos estrella

1. Agregar el nombre del usuario en los registros que se escriben al archivo secreto.
2. Arreglar la fecha y hora que se muestra al hacer ls -l (y con ellos se ganan la eterna gratificación de algunos profes).
3. Agregar una caché con el número de los clusters de datos leído de un archivo, que se llena a medida que se leen. De esta forma, evitamos recorrer la tabla FAT por cada operación de lectura o escritura. Cuando el número de file descriptors abiertos para el archivo es 0, la caché se destruye. Les recomendamos implementarlo agregando una GList a la estructura fat_file_s.
4. [Difícil pero probablemente gratificante] Agregar soporte para nombres de archivo de más de 8 caracteres. Hay que implementar la lectura de nuevos tipos de clusters, no sólo datos y directorios.
5. [Difícil pero probablemente gratificante] Agregar soporte para eliminar archivos. Deben implementar la función unlink y rmdir.
6. Agregar un nuevo cluster al directorio cuando este se llena de entradas.

Extractos de coding style
https://www.doc.ic.ac.uk/lab/cplus/cstyle.html

- Comments should justify offensive code. The justification should be that something bad will happen if unoffensive code is used.