TP3: Filesystem FUSE

---

e comenzar, verificar que se tiene instalado **el software necesario**.

plementaremos nuestro propio sistema de archivos (o *filesystem*) para Linux. El sistema de archivos utilizará el mecanismo de FUSE (*Files*
to por el kernel, que nos permitirá definir en *modo usuario* la implementación de un *filesystem*. Gracias a ello, el mismo tendrá la interfaz VF
syscalls y programas habituales (`read` , `open` , `ls` , etc).

n del *filesystem* será enteramente en memoria: tanto archivos como directorios serán representados mediante estructuras que vivirán en n
nos un sistema de archivos que apunte a la velocidad de acceso, y no al volumen de datos o a la persistencia (algo similar a `tmpfs`). Aún
estarán representados *en disco* por un archivo.

sario

esto de varios componentes, los principales son:

el kernel (que se encarga de hacer la delegación)
le usuario que se utiliza como framework

trabajo se necesitará un sistema operativo que cuente con el kernel Linux, y que disponga de las librerías de FUSE versión 2.

todas las dependencias con el siguiente comando:

```
e && sudo apt install pkg-config libfuse2 libfuse-dev
```

eleto viene con un *filesystem* FUSE trivial, para poder probar las dependencias. Si las mismas están correctamente instaladas, deberían po
del esqueleto de la siguiente forma:

```
3 -O2 -Wall -std=c11 -Wno-unused-function -Wvla fisopfs.c  -D_FILE_OFFSET_BITS=64 -I/usr/include/fuse -lfuse -pthrea
```

n directorio

```
s un directorio vacío
rueba
amos nuestro binario y le decimos dónde queremos que monte nuestro filesystem
fs prueba/
camos que se haya montado correctamente
 grep fisopfs
labs/fisopfs/fisopfs on /vagrant/labs/fisopfs/prueba type fuse.fisopfs (rw,nosuid,nodev,relatime,user_id=1000,group_
```

re el directorio

```
prueba/

x 2    1717 root       0 Dec 31  1969 .
x 1 vagrant vagrant  352 Nov 13 12:41 ..
- 1    1818 root    2048 Dec 31  1969 fisop
eba/fisop
pfs!
```

```
ount prueba
```

FUSE

ejecución de un cliente FUSE es algo distinta. El esqueleto ya está preparado (en el `Makefile`) para compilar incluyendo los flags de comp
e recomienda leer este artículo antes de arrancar, y antes de introducir modificaciones en el `Makefile` . En particular, se utiliza la utilidad
le compilación adecuados.

bién podrán encontrar una explicación sobre cómo utilizar la librería de FUSE e implementar sus propias funciones. En el caso del esquelet
das 3 primitivas del sistema de archivos: `getattr`, `readdir` y `read`.

```
fuse_operations operations = {
tr = fisopfs_getattr,
ir = fisopfs_readdir,
= fisopfs_read,
```

oficial de FUSE es muy útil para revisar las firmas de las funciones y los campos de cada struct de la librería. Sin embargo, hay que tener
n es **para la versión 3**, y por lo tanto podría tener algunas diferencias en la API/firma de algunas funciones que utilizaremos en el TP.

para la versión 2 (la última, 2.9.9) más precisa es el código del *header* (`fuse.h`) en sí, el cual pueden encontrarlo en el repositorio. En el r
los *structs*, funciones auxiliares y firmas; junto con comentarios útiles.

Si tienen errores al momento de compilar, o ven alguna discrepancia con la documentación, es posible que estén usando FUSE versión 3. P
`on apt list "libfuse*"`

ón

`fisopfs`, un *filesystem* de tipo FUSE definido por el usuario. El mismo deberá implementar un subconjunto de las operaciones que sopor
las necesarias para soportar la lista de operaciones que figura a continuación.

queridas

to que el sistema de archivos soporte las siguientes funcionalidades:

ón de archivos (`touch`, redirección de escritura)
ón de directorios (con `mkdir`)
a de directorios, incluyendo los pseudo-directorios `.` y `..` (con `ls`)
a de archivos (con `cat`, `more`, `less`, etc)
ra de archivos (sobre-escritura y append con redirecciones)
er a las estadísticas de los archivos (con `stat`)
uir y mantener fecha de último acceso y última modificación
mir que todos los archivos son creados por el usuario y grupo actual (ver `getuid(2)` y `getgid(2)`)
o de un archivo (con `rm` o `unlink`)
o de un directorio (con `rmdir`)
ón de directorios debe soportar al menos un nivel de recursión, es decir, directorio raíz y sub-directorio.

las funcionalidades pedidas, se espera que la misma se pueda corroborar montando el sistema de archivos e interactuando con el mismo
la misma forma que se haría con cualquier otro *filesystem*.

ena forma de saber qué *operaciones* hacen falta es implementar las mismas vacías con `printf` de debug; y montar el *filesystem* en prime
`...`. Esto permitirá loggear todas las operaciones/syscalls que generan los procesos al interactuar con el sistema de archivos.

funcionalidad no implementada, el sistema de archivos debe escribirlo por pantalla (basta con un `printf` a `stderr`, o con el prefijo `[de`
. Pueden tomar los errores definidos en `errno.h` (ver `errno(3)`) como inspiración, viendo qué errores arrojan otros sistemas de archivos
n: `ENOENT`, `ENOTDIR`, `EIO`, `EINVAL`, `EBIG` y `ENOMEM`, entre otros.

n del sistema de archivos

ivo implementado debe existir en memoria RAM durante su operación. La estructura en memoria que se utilice para lograr tal funcionalida
. Deberá explicarse claramente, con ayuda de diagramas, en el informe del trabajo (i.e. el archivo `fisopfs.md`); las decisiones tomadas y
nas.

consistirá en el diseño de las estructuras que almacenarán toda la información, y cómo se accederán en cada una de las operaciones. Para
la segunda parte del trabajo práctico.

n de diseño

explicar los distintos aspectos de diseño:

tructuras en memoria que almacenarán los archivos, directorios y sus metadatos
el sistema de archivos encuentra un archivo específico dado un *path*
ipo de estructuras auxiliares utilizadas
nato de serialización del sistema de archivos en disco (ver siguiente sección)
uier otra decisión/información que crean relevante

omienda utilizar una estructura *flat* para los directorios, limitando el *filesystem* a un único nivel de recursión en los directorios. O bien imple
nodes similar a los sistemas de archivo *Unix-like*, y permitir múltiples niveles de directorios. De todas formas, es recomendable definir un lí
th o en la cantidad de directorios anidados soportados.

ir un tamaño máximo para el sistema de archivos, y arrojar `ENOSPC` (*No space left on device*), o similar, si nos excedemos del mismo. Se re
s de tamaño estático para tal fin, para simplificar el manejo de memoria.

n disco

de archivos puede vivir enteramente en RAM durante su operación, también será necesario persistirlo a disco al desmontarlo y recuperarlo

o se representará como un único archivo en disco, con la extensión `.fisopfs`; y en el mismo se serializará toda la estructura del *filesyster*
era que toda esa información se lea de disco en memoria, y la operación continúe exclusivamente en memoria. Cuando el *filesystem* se des
a explícita a `fflush`), la información debe persistirse nuevamente en disco. De esta forma, a través de múltiples ejecuciones, los datos pe

n disco

to que el sistema de archivos se persista en disco
único archivo, de extensión `.fisopfs`
ar el *filesystem*, se debe especificar un nombre de archivo, si no se hace, se elige uno por defecto
hivo especificado se lee todo el *filesystem*, y se inicializan las estructuras acordemente (esto ocurre en la función `init`)
re un `flush` o cuando el sistema de archivos se desmonta (esto ocurre en la función `destroy`), la data debe persistirse en el archivo nue

las de ejemplo

implementación de `fisopfs` también será necesario **incorporar pruebas** de caja negra sobre lo implementado. Las mismas deben consis
nandos pensadas para generar un escenario de prueba, junto con la salida esperada del mismo. Un ejemplo, es lo presentado en la secció

d implementada debe incluir una prueba asociada. Las salidas de las pruebas deben incluirse como una sección en el informe.

mendable pensar y escribir las pruebas incluso antes de arrancar con la implementación, para tener una guía del comportamiento del siste

ompilación

eleto del TP3 se encuentra disponible en **fisop/fisopfs**.

leer el archivo `README.md` que se encuentra en la raíz del proyecto. Contiene información sobre cómo realizar la compilación de los archiv
nateo de código.

stro *filesystem*, se puede:

das aquí no son obligatorias, pero suman para el régimen de final alternativo.

n de más operaciones para `fisopfs`

quisitos obligatorios, los grupos podrán optar por implementar *al menos* dos de las siguientes funcionalidades adicionales.

a enlaces simbólicos

lementarse la operación `symlink`
luirse pruebas utilizando `ln -s`

a hard links

lementarse la operación `link`
luirse pruebas utilizado `ln`
ahora el borrado *real* de un archivo solo debe ocurrir si no quedan más *hard links* asociados al mismo.

a múltiples directorios anidados

os niveles de directorios
mplementar una cota máxima a los niveles de directorios y a la longitud del *path*

daciones de permisos

ar si el usuario que accede tiene permisos para leer el archivo/directorio
tar las operaciones `chown` y `chmod` para modificar permisos y ownership de un archivo/directorio

las operaciones elegidas deben implementarse incluyendo pruebas de la misma forma que para el resto de las funcionalidades.

il

presentan algunos enlaces y bibliografía útiles como referencia.

ulo 39: *Interlude: Files and Directories* (PDF)
ulo 40: *File System Implementation* (PDF)
ogramming Interface, capítulo 14: *File systems*