Práctica 2

Todo lo que se realice en el Laboratorio se podrá subir al campus virtual. La entrega se realizará en un único archivo comprimido. Dicha entrega se realizará de forma individual, aunque el trabajo se realice en grupo.

Cada archivo de código fuente o script tendrá una pequeña sección indicando los nombres, apellidos y DNI de los componentes del grupo que han desarrollado dicho código.

A continuación se indican las actividades a realizar para la práctica, así como los ficheros a subir al campus virtual dentro del entregable final.

Ejercicio 1

Se realizará el Ejercicio 1 de la práctica, comprobando las diferencias de velocidad en función del tamaño del buffer. Los ficheros a incorporar en la entrega serán copy`.c` y posibles versiones.

Ejercicio 2

Se realizará el Ejercicio 2 de la práctica. Los ficheros a incorporar en la entrega serán `status.c` y posibles versiones.

Ejercicio 3

Se realizará el Ejercicio 3 de la práctica, comprendiendo el funcionamiento del código. Los ficheros a incorporar en la entrega serán `distr.c` y posibles versiones.

Desarrollo del código de la práctica

Esta sección incluye el desarrollo del código principal de la práctica, incorporando las operaciones borrar fichero (`unlink`) y leer (`read`). Una vez comprobado su correcto funcionamiento con diversas pruebas, se podrán añadir a la entrega todos los archivos .c y .h de la práctica. Opcionalmente, se podrá entregar un archivo Makefile si se utilizó para compilar el proyecto.

Script de comprobación

Esta tarea consiste en implementar el script indicado en el guión de la práctica. Se podrá añadir a la entrega el script bash implementado.

A continuación, se detallan algunas posibles extensiones que se podrán desarrollar si se considera oportuno. En todas ellas, se podrán añadir a la entrega los archivos .c y .h de la práctica. No es necesario añadir todas las versiones de la misma. Como son extensiones acumulables, bastará con entregar la última versión alcanzada.

Extensión 1 - Opción -m

Implementar la opción `-m` en el proyecto, de forma que se permita montar un sistema de ficheros existente. Esto implica implementar la función:

```
int myMount(MyFileSystem* myFileSystem, char* backupFileName);
```

, que llama a su vez a las siguientes funciones:

```
int readBitmap(MyFileSystem* myFileSystem);
int readDirectory(MyFileSystem* myFileSystem);
int readSuperblock(MyFileSystem* myFileSystem);
int readInodes(MyFileSystem* myFileSystem);
```

Extensión 2 - Soporte para enlaces simbólicos

Dar soporte en el sistema de ficheros a enlaces simbólicos.

Para poder realizar este apartado se puede, por ejemplo, modificar la estructura del nodo-i para que contenga un campo entero llamado `tipo` que identifique el tipo de nodo-i, que podrá ser bien un fichero regular (tipo=0, por ejemplo) o bien un enlace simbólico (tipo=1, por ejemplo). Para crear un enlace simbólico se debe implementar la función:

```
int fuse_operations::symlink(const char* path1, const char* path2);
```

Para leer un enlace simbólico se debe completar adecuadamente la función `getattr`, actualizando la estructura stat según el campo tipo mencionado anteriormente

```
stbuf->st_mode = S_IFLNK | 0644;
```

Además, se debe implementar la función:

```
int fuse_operations::readlink(const char*, char*, size_t);
```

Extensión 3 - Soporte para enlaces rígidos

Dar soporte en el sistema de ficheros a enlaces rígidos. Básicamente consiste en realizar las siguientes modificaciones (puede que haya que realizar modificaciones adicionales, dependiendo de cada proyecto):

- Añadir un campo `nlinks` al nodo-i que lleva la cuenta del número de enlaces.
- Modificar la función `copyNode` de `myFS.c` para copiar también este campo.
- Modificar la función `my_mknod`, para que inicialice el número de enlaces a 1 al crear el nodo-i.
- Modificar la función `my_getattr`, para que lea el número de enlaces del nodo-i.
- Modificar la función `my_unlink`, para que decremente el contador y borre sólo el contenido si el contador de enlaces se hace 0.
- Implementar la función link de fuse:
  ```
  int fuse_operations::link(const char *path, const char *lpath);
  ```
  que crea un enlace lpath a un fichero path. Registrar dicha función en la estructura de operaciones de fuse.

Extensión 4 - Añadir soporte parcial para directorios

Modificar la estructura del nodo-i para que aparezca también el tipo (si no se ha hecho en extensiones anteriores), que declararemos como un entero

- `tipo = 0` → Archivo regular
- `tipo = 1` → Enlace simbólico (usado en la Extensión 1, si se ha implementado)
- `tipo = 2` → Directorio

Modificar la funciones de `fuseLib.c` necesarias para que cuando se cree un archivo este aparezca por defecto como tipo archivo. Modificar `my_getattr` para tener en cuenta que nuestro sistema puede tener archivos y directorios.

Modificar las funciones directamente relacionadas con archivos para que sólo puedan ejecutarse si el nodo-i es de tipo archivo regular y den error en otro caso (EISDIR (The named file is a directory)). Añadir la función `my_mkdir`, que cree un directorio (no se va a acceder al directorio, no va a tener contenido, pero cuando hagamos un `ls` debe aparecer como un directorio).