PRÁCTICA 3: SISTEMA DE FICHEROS

DISEÑO DE SISTEMAS OPERATIVOS

Curso 2015-2016

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

		GRUPO
JAVIER PRIETO CEPEDA:	**100 307 011**	**82**
SERGIO RUIZ JIMÉNEZ:	**100 303 582**	**81**
MARIN LUCIAN PRIALA:	**100 303 625**	**83**

ÍNDICE

TABLAS DE PRUEBAS

1 Introducción

La práctica número 3 de diseño de sistemas operativos ha sido diseñada e implementada por los alumnos Marin Lucian Priala, Sergio Ruiz Jiménez y Javier Prieto Cepeda.

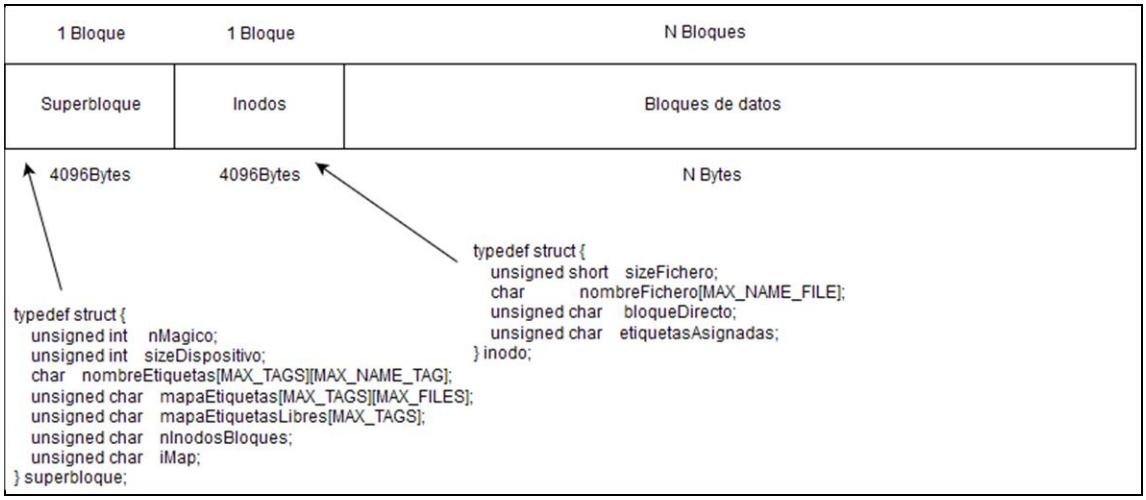
En esta práctica se pide diseñar e implementar un sistema de ficheros. Para ello se ha elegido el modelo de UNIX.

El presente documento se estructurará de la siguiente manera:

1. **Diseño del sistema de ficheros,** donde se detallarán los aspectos más relevantes a nivel general.
2. **Descripción del código,** donde se explicarán las funciones del sistema de ficheros.
3. **Batería de pruebas**, donde se escribirán las pruebas más relevantes y muy detalladas del sistema de ficheros.
4. **Conclusiones**, un texto breve sobre las opiniones y reflexiones de la práctica actual.

2 Diseño del Sistema de Ficheros

Hemos realizado un diseño lo más equilibrado posible, es decir, buscar ocupar lo mínimo en memoria para los metadatos y que sea eficiente. De este modo, nuestro sistema de ficheros, tendría la siguiente estructura:



Por ello hemos creado dos bloques para los metadatos. Un primer bloque que será el superbloque, y el segundo bloque para los inodos del sistema de ficheros.

Cabe destacar que N Bytes del bloque de datos, dependerá del número de bloques de datos, siendo este como máximo 50 y cada uno ocupa 4096B.

Como hemos dicho anteriormente, el máximo de N Bloques será 50.

En este primer bloque se encontrará la información más relevante para el sistema de ficheros. Para el superbloque hemos creado 7 variables, cada una de ellas tiene una función dentro del sistema de ficheros, por orden, desde arriba hacia abajo, daremos su función.

- **nMagico:** Esta variable incluye los últimos 6 dígitos del NIA, y sirve para identificar nuestro sistema de ficheros. Se ha declarado del tipo int, ya que son 6 digitos, por lo tanto como máximo permitiremos el número 999999, que cabe dentro de un int.
- **sizeDispositivo:** Indica el tamaño del dispositivo en bytes. Por lo tanto, con un unsigned int podremos guardar el número de ellos sin problema.
- **mapaEtiquetas:** Indica el nombre de las etiquetas contemplando tanto el máximo de etiquetas como el máximo de nombre de etiqueta. Por lo que se ha decidido crear la variable como un array doble del tipo char, nunca se superaran 256 caracteres.
- **mapaEtiquetas:** Indica que archivos contienen las etiquetas. Con un array doble del tipo char, donde el primer campo es el número máximo de etiquetas, y el segundo del tamaño máximo de ficheros. De esta manera, primero se recorren las etiquetas, y después los archivos asociados a esa etiqueta.
- **mapaEtiquetasLibres:** Indica en un array la posición de las etiquetas que están libres en el sistema. Siendo 1 el valor para estar ocupada, y 0 si se

encuentra libre. Al ser tan solo 30 etiquetas, puede ser declarado del tipo char.
- **nInodosBloques:** Indica el número de inodos que tiene el sistema a la hora de su creación. Como el valor máximo será 50, al ser de tipo char no tendríamos problemas para almacenar el valor.
- **iMap:** Variable que nos indica el primer número del inodo libre. Como máximo serán 50, al declarar de tipo char, podremos almacenar todos los valores deseados.

En el segundo bloque tan solo habrá información relacionada con los ficheros:

- **sizeFichero:** Indica el número de bytes que tiene el fichero asociado al inodo. El valor como máximo será de 4096, por lo tanto, al ser de tipo short puede almacenar el valor.
- **nombreFichero:** Array que almacena el nombre de fichero, como máximo será de 64 bytes. Esto quiere decir que como máximo almacenará 63 caracteres más el caracter de fin de cadena. Al ser del tipo char, permite almacenar sin problema el número de ficheros y sus nombres.
- **bloqueDirecto:** Indica el bloque de datos asociado al inodo. En nuestro caso, un inodo tan solo podrá tener un bloque de datos asociado. Y al existir tan solo bloque de datos como máximo con una variable del tipo char podremos almacenar el valor.
- **etiquetasAsignadas:** Indica el número de etiquetas asociadas al inodo/fichero correspondiente. Como máximo será 3 y como mínimo 0. Por lo tanto con ser del tipo char es suficiente.

Con estos dos bloques fijos, que ocuparan 4096B cada uno, debido a que es el tamaño de bloque mínimo, a pesar de que tan solo ocupa 2500B el primer bloque y el segundo 68 B por cada inodo existente.

Una comprobación previa para saber si el sistema de ficheros era viable, era realizar la subdivisión del bloque de los inodos según el tamaño del inodo, esto es debido a que el sistema de ficheros debe ser capaz de tener 50 ficheros. Con esta comprobación podemos ver que 4096B/68B/inodo = 60 inodos.

Por lo tanto, un fichero irá asociado a un inodo. Permitiendo 50 inodos, incluso más en un futuro si fuera requerido.

Hay ciertos detalles de implementación que consiguen un sistema más eficiente, tanto en memoria como en rapidez, como por ejemplo, el uso de la variable iMap para representar el primer inodo libre del sistema. Con esta variable se evita crear un mapa con 50 inodos. Esto es debido a que nuestro sistema de ficheros no tiene la función para borrar un fichero. Por lo tanto no se debe saber la posición del fichero que se borró. De esta manera si esta variable fuera 7, significaría que existen 6 ficheros y hay posibilidad de crear otro en la posición 7 (primera posición libre).

Esta pequeña mejora descrita anteriormente, nos evita crearnos una variable char de 50 posiciones. Lo que quiere decir 50*1= 50B reservados. Con un char tan solo reservamos 1 byte. O lo que es lo mismo 256 posiciones, que son suficientes.

Otra característica de nuestro sistema de ficheros, es el uso de memoria para trabajar. Evitamos escribir en disco todos los metadatos cada vez que se realiza una

operación, aunque siempre escribimos el bloque de datos. Esto nos permite no estar acudiendo a disco, evitando tiempos de espera demasiado largos. En nuestro caso al ser un sistema de ficheros pequeños, no ocupa nada en memoria y hace que sea muy eficiente. Cuando se realiza la operación unmount, se debe volcar el contenido de memoria de los metadatos al disco, para que la próxima vez que se haga mount, cargarlos en memoria por primera vez y trabajar con ellos.

Es recomendable ejecutar la operación unmount cada pocos minutos, ya que si existieran problemas ajenos al sistema de ficheros, podría perderse información asociada a los datos (metadatos). Aunque los datos serían recuperables con análisis forense, ya que siempre son escritos.

Para las operaciones de escritura y lectura, hemos seguido el modelo de UNIX, es decir, si realizamos una operación de escritura sobre un fichero previamente abierto. El puntero de posición del fichero, apuntará a la última posición escrita por el sistema. Esto quiere decir, que si realizamos una lectura posteriormente, no leeremos nada. Debido a que el puntero al que apunta el fichero, no contiene nada a partir de esa posición. Para poder leer desde la primera posición, se debe realizar un lseek al principio o una operación de cierre y apertura del fichero, para restaurarlo a 0 el puntero.

Para la operación mount, se ha decidido realizar un bzero a las variables de memoria, para evitar inconsistencias a la hora de volver a recuperar los datos del disco e introducirlos en memoria.

Así mismo, en la operación unmount usamos el mismo procedimiento de realizar bzero sobre las variables en memoria para evitar inconsistencias.

Estas dos decisiones han sido tomadas para evitar inconsistencias a la hora de realizar el sistema de ficheros, montarlo, desmontarlo y volver a realizar el sistema de ficheros. Ya que, podía existir la posibilidad de leer metadatos antiguos, que pudieran comprometer la consistencia y seguridad del sistema.

3 Descripción del código

En este apartado se realizará una descripción de la implementación del código por cada una de las funciones pedidas para implementar.

3.1 Gestión del sistema de ficheros

3.1.1 mkFS

Esta función, se encarga de dar formato al disco (la parte de metadatos, es decir, el bloque que contiene el superbloque y el bloque de i-nodos), creando el sistema de ficheros y teniendo en cuenta los parámetros recibidos.

En primer lugar, comprobamos que los parámetros recibidos, tanto de la capacidad del disco como el número de ficheros, estén entre el rango establecido en el enunciado. Por tanto, si la capacidad del dispositivo no está entre 320KB y 500KB o el número de ficheros entre 0 y 50 (número máximo de ficheros soportados por el sistema de ficheros) entonces devolvemos -1 indicando error.

En caso de no ocurrir ningún error, sobrescribimos el bloque de i-nodos por si ha habido algún sistema de ficheros anteriormente, de esta manera nos aseguramos que no hay inconsistencia en los metadatos.

A continuación, rellenamos todos los atributos del superbloque. Asignamos el número mágico, la capacidad del dispositivo y el número de i-nodos según los parámetros recibidos. Las demás variables (etiquetas, mapa de etiquetas y las etiquetas libres) del superbloque las rellenamos con un valor por defecto, siendo este valor cero.

Una vez creado el superbloque con todas las variables asignadas, escribimos en disco el superbloque para hacerlo persistente, pero no montamos el sistema de ficheros.

Finalmente, si el proceso de escritura en el disco, del i-nodo y del superblque, falla devolvemos -1 indicando que ha ocurrido un error, de lo contrario se devuelve un 0 indicando que el sistema de ficheros se ha creado con éxito.

3.1.2 mountFS

Esta función se encarga de realizar el montado del sistema de ficheros. Para realizar el montado, se deben rellenar las estructuras de datos de memoria (superbloque e i-nodos) con los datos que están en el disco.

En primer lugar, se comprueba si el sistema de ficheros se ha montado previamente. En caso de no haber sido así, se limpian las estructuras de memoria, por si hay datos guardados de anteriores sistemas de ficheros, de manera que nos aseguramos que montamos el sistema de ficheros que hay en el disco y no lo que se haya podido quedar guardado en las estructuras.

Posteriormente se lee de disco el bloque que contiene los datos del superbloque y el bloque de i-nodos. En caso de no realizarse correctamente la lectura se devuelve -1 indicando error.

Si la lectura se ha realizado correctamente, se rellena la estructura del superbloque y la de i-nodos con el contenido leído de disco.

Finalmente se comprueba si el número mágico del sistema de ficheros es correcto. En caso de no serlo, se devuelve -1 indicando error, de lo contrario indicamos que el sistema de ficheros ha sido montado y devolvemos 0 para indicar éxito.

3.1.3 umountFS

Esta función se encarga de realizar un volcado de los metadatos del sistema de ficheros a disco.

En primer lugar se comprueba si el sistema de ficheros ha sido desmontado ya, en cuyo caso se devuelve -1 indicando error.

En caso de no haber error, se comprueba que todos los ficheros del sistema de ficheros están cerrados. Si algún fichero está abierto, se devuelve -1, para indicar error y que no se puede desmontar el sistema de ficheros. De lo contrario escribimos en disco los metadatos el superbloque y el bloque que contiene los inodos.

Si la escritura de los metadatos no se ha realizado con éxito entonces devolvemos -1, indicando que ha ocurrido un error. Si la operación de escritura se ha realizado con éxito, entonces indicamos que el sistema de ficheros esta desmontado del sistema y limpiamos las estructuras que contenían los metadatos en memoria.

Finalmente devolvemos 0 indicando que todo se ha realizado correctamente.

Únicamente en esta función se vuelcan los metadatos en disco. En el resto de casos se trabaja con los metadatos que están en memoria. Para poder evitar posibles errores, como por ejemplo, ante caídas de la corriente eléctrica, se debería implementar una función que realizase el volcado de los metadatos cada por tiempo.

3.2 Lectura y escritura de ficheros

3.2.1 createFS

Esta función se encarga de crear un fichero en el sistema de ficheros con el nombre que se recibe por parámetro.

En primer lugar, se comprueba si el sistema de ficheros está montado. Si no lo está, se devuelve -1 indicando de esta manera que ha habido un error al crear el fichero. Del mismo modo si la longitud del nombre del fichero excede 64 caracteres. En nuestro caso, almacenamos 64 caracteres, incluyendo en estos el carácter ‘\0’, de forma, que como máximo, permitimos 63 caracteres + ‘\0’. También si en el sistema de ficheros ya no se pueden crear más ficheros.

A continuación, si se puede crear el fichero, en el caso de ser el primer fichero creado en el sistema de ficheros, creamos el inodo correspondiente rellenando los campos necesarios de dicha estructura. En caso de no ser el primer fichero del sistema de ficheros, entonces recorremos el mapa de i-nodos buscando si ya existe dicho fichero, si existe devolvemos 1 indicando su existencia, de lo contrario al encontrar el primer i-nodo libre lo creamos.

Cabe destacar que si creamos el i-nodo en memoria, no se vuelca a disco, únicamente se vuelcan los metadatos en disco al realizar umountFS, tal como se ha mencionado anteriormente.

Finalmente devolvemos 0 indicando que todo se ha realizado correctamente.

3.2.2 openFS

Esta función se encarga de abrir un fichero del sistema de ficheros mediante el nombre recibido por parámetro.

Para lograr esta funcionalidad, debemos comprobar que el sistema de ficheros está montado. En caso de no estar montado, devolvemos -2 indicando un error general. De lo contrario recorremos todos los inodos del sistema de ficheros. Si no encontramos un fichero con ese nombre, devolvemos -1 indicando error, ya que no existe el fichero con el nombre recibido por parámetro. Si encontramos el fichero, entonces comprobamos que no está abierto. Si está abierto, devolvemos -2, indicando un error general. De lo contrario, abrimos el fichero estableciendo la posición del puntero al comienzo del fichero. Finalmente devolvemos el descriptor de fichero. Cabe destacar que el descriptor del fichero es el bloque directo asociado a dicho fichero (i-nodo).

3.2.3 closeFS

Esta función debe cerrar un fichero abierto del sistema de ficheros mediante el descriptor de ficheros que se recibe por parámetro.

Primero comprobamos que el sistema de ficheros debe estar montado, de lo contrario devolvemos -1 indicando error.

Si el sistema de ficheros está montado entonces buscamos el inodo correspondiente al descriptor recibido y lo cerramos.

Cabe destacar que la información de si un fichero está abierto o no como la posición del mismo se encuentra en una estructura en memoria y no en los metadatos del que recuperamos del disco. Por esta misma razón comprobamos que el sistema de ficheros está montado.

3.2.4 readFS

Esta función se encarga de leer de un fichero tantos bytes como indique el parámetro recibido.

Para lograr esta funcionalidad, debemos comprobar primero que el sistema de ficheros está montado. En caso de no estar montado, devolvemos -1 indicando error. Si lo está comprobamos que el fichero está abierto. En caso de no estarlo, devolvemos -1 indicando error. Después comprobamos el puntero de posición, si está al final del fichero entonces devolvemos 0, de lo contrario leemos el bloque de datos y devolvemos al usuario solamente el número de bytes pedidos si no excede el tamaño del fichero, de lo contrario solamente leemos lo que se pueda leer hasta llegar al final del fichero, devolviendo el número de bytes que se han podido leer.

Finalmente, antes de devolver el número de bytes leídos, actualizamos el puntero de posición del fichero a la nueva posición, es decir, la última posición del fichero leída.

3.2.5 writeFS

Esta función se encarga de escribir en el fichero pasado por parámetro como descriptor.

Si el sistema de ficheros no está montado o el fichero no está abierto entonces devolvemos -1 indicando error.

Si la posición del puntero del fichero está al final del bloque, devolvemos 0 bytes escritos.

En el caso de que el número de bytes que se desea escribir es mayor que el tamaño máximo del fichero o mayor que el número de bytes restantes desde la posición actual del puntero hasta el final del bloque, se devolverá -1 indicando error.

A continuación, leemos el bloque de disco y solamente escribimos los bytes que se desean escribir desde la posición actual del puntero del fichero.

Escribimos los datos en el disco, y posteriormente actualizamos el puntero de posición del fichero. Si ha ocurrido algún error al escribir en disco devolvemos -1 indicando error y actualizamos el tamaño del fichero si ha crecido el fichero.

Finalmente devolvemos el número de bytes que se han podido escribir.

3.2.6 lseekFS

Esta función se encarga de posicionar el puntero del fichero en una posición en concreto, al inicio o al final del mismo según el tipo de posicionamiento que se elija.

Para comenzar, se comprueba si el sistema de ficheros está montado, si no lo está entonces devolvemos -1 indicando error.

Además, si el tipo de posicionamiento es distinto a los tres tipos proporcionados entonces devolvemos -1 indicando error. De igual manera si el fichero no está abierto.

En caso de seleccionar el tipo de posicionamiento BEGIN, se hace caso omiso al segundo parámetro y se establece la posición del fichero al comienzo del fichero.

En caso de seleccionar END, al igual que en BEGIN con la salvedad de colocar el puntero de posición al final del fichero.

Y en caso de seleccionar el SET, se comprueba que la segunda variable, que es la de posicionamiento, no se salga del límite del fichero ni tome valores negativos. Si los toma entonces devolvemos -1 indicando error.

En cualquier caso, se retorna el nuevo puntero de posición.

3.3 Gestión de etiquetas

3.3.1 tagFS

Esta función se encarga de asignar una etiqueta a un fichero.

Si el sistema de ficheros no está montado, devolvemos -1 indicando error. Del mismo modo, si el nombre de la etiqueta es más largo que los propios parámetros establecidos o el fichero no existe devolvemos -1 indicando error. Sin embargo, si el fichero existe y está abierto también devolvemos -1 indicando error.

A continuación, buscamos la etiqueta, si existe y está asociada en el mapa de etiquetas y ficheros, entonces devolvemos 1, la etiqueta está asociada.

3.3.2 untagFS

Esta función se encarga de eliminar la asociación de una etiqueta a un fichero pasados por parámetro sin borrar el fichero ni la etiqueta, solamente se borra la etiqueta si esta deja de estar asociada a ficheros.

Primero comprobamos si el sistema de ficheros está montado, si no lo está devolvemos -1 indicando error.

A continuación, buscamos la etiqueta para ver si existe entre todas las etiquetas disponibles que se estén usando en el sistema de ficheros. Si la etiqueta no existe, entonces devolvemos -1, indicando error. Pero si existe, una vez encontrada buscamos que el fichero exista entre los creados en el sistema de ficheros, al mismo tiempo buscamos la asociación de la etiqueta con el fichero, en caso de encontrar el fichero y la asociación entones borramos la etiqueta del fichero y seguimos mirando si la etiqueta se ha quedado sin ficheros. Si el fichero no existe entonces devolvemos -1, indicando error. Al igual si existe, pero el fichero está abierto. En caso de existir el fichero y no estar asociado a la etiqueta, entonces devolvemos 1, indicando que la etiqueta no está asociada.

Finalmente, si la etiqueta se ha quedado sin ficheros asociados, eliminamos la etiqueta y devolvemos 0 indicando éxito. Pero si la etiqueta se ha eliminado del fichero, pero sigue teniendo ficheros asociados entonces únicamente devolvemos 0, indicando éxito.

3.3.3 listFS

Esta función se encarga de listar todos los ficheros que estén etiquetados con la etiqueta pasada por parámetro.

Primero comprobamos que el sistema de ficheros este montado. Si no lo está entonces devolvemos -1, indicando error.

A continuación, buscamos la etiqueta entre todas las etiquetas que se estén usando en el sistema de ficheros. Si se encuentra, entonces buscamos para todos los ficheros creados en el sistema de ficheros la asociación de etiqueta-fichero, si encontramos algún fichero que tenga asociada dicha etiqueta entonces guaramos el nombre del fichero en la lista e incrementamos el número de ficheros encontrados para dicha etiqueta. Esto se realiza hasta recorrer todos los ficheros existentes en el sistema de ficheros. Una vez finalizada la búsqueda devolvemos el número de ficheros encontrados y el nombre de cada fichero.

Si al finalizar la búsqueda no se encuentra el fichero, etiqueta o asociación entonces devolvemos -1, indicando error.

4 Batería de pruebas

En este apartado se realizarán las pruebas pertinentes para demostrar el correcto funcionamiento del sistema de ficheros.

Para poder ejecutar estas pruebas se necesita:

1) Compilar todos los archivos.
2) Crear un disco limpio.
3) Ejecutar el test.

La prueba número tres aparece al comienzo de la ejecución porque se necesita un disco limpio.

4.1 Pruebas para gestión del sistema de ficheros

Prueba 1	
Objetivo	Probar los casos límites de la capacidad de disco. Crear un FS con una capacidad de disco: 1) Inferior a la mínima. 2) Igual a la mínima establecida. 3) Igual a una capacidad entre el límite inferior y superior. 4) Igual a la máxima establecida. 5) Superior al máximo establecido.
Traza	Considere que el Sistema de Ficheros se utilizará siempre en dispositivos de entre 320 KB y 500 KB.
Resultado esperado	1) Inferior a la mínima. → Error 2) Igual a la mínima establecida. → Éxito 3) Igual a una capacidad entre el límite inferior y superior. → Éxito 4) Igual a la máxima establecida. → Éxito 5) Superior al máximo establecido. → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 1: Prueba 1

Prueba 2	
Objetivo	Probar los casos límites del número de ficheros. Crear un FS con un número de ficheros: 1) Inferior a 0. 2) Igual a 0. 3) Igual a un número entre el 0 y máximo. 4) Igual al máximo establecido. 5) Superior al máximo establecido.
Traza	La cifra exacta de ficheros que puede contener el Sistema de Ficheros se especifica al crear el Sistema de Ficheros, teniendo en cuenta que no podrá exceder de esos 50 ficheros. También traza de la Prueba 1.
Resultado esperado	1) Inferior a 0. → Error 2) Igual a 0. → Éxito 3) Igual a un número entre el 0 y máximo. → Éxito 4) Igual al máximo establecido. → Éxito 5) Superior al máximo establecido. → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 2: Prueba 2

Prueba 3	
Objetivo	Montar un sistema de ficheros que no está creado, y que todo el disco este a 0. Este requisito es imprescindible que se dé. 1) Crear un sistema de ficheros valido y montarlo. 2) Montar de nuevo un sistema de ficheros ya montado.
Traza	No se pide explícitamente en el enunciado, pero se entiende que si el sistema de ficheros no existe no se pueda montar.
Resultado esperado	1) Sistema de ficheros inexistente + mountFS → Error 2) Sistema de ficheros valido + mountFS → Éxito 3) Montar un sistema de ficheros ya montado → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 3: Prueba 3

Prueba 4	
Objetivo	1) Desmontar un sistema de ficheros que no está montado. 2) Montar un sistema de ficheros y desmontarlo después.
Traza	No se pide explícitamente en el enunciado, pero se entiende que si el sistema de ficheros no está montado no se puede desmontar. Además, si no está montado el sistema de ficheros debe poder montarse.
Resultado esperado	1) Sistema de ficheros desmontado + umountFS → Error 2) mountFS + umountFS → Éxito Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 4: Prueba 4

Prueba 5	
Objetivo	1) Crear un sistema de fichero valido, montarlo y finalmente desmontarlo.
Traza	Crear un sistema de ficheros y poder usarlo.
Resultado esperado	1) mkFS → Éxito 2) mountFS → Éxito 3) umountFS → Éxito Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 5: Prueba 5

4.2 Lectura y escritura de ficheros

Prueba 6	
Objetivo	1) Crear un fichero con un nombre de longitud 0. 2) Crear un fichero con un nombre de longitud igual al máximo permitido. 3) Crear un fichero con un nombre de longitud superior al máximo permitido. 4) Crear un fichero que ya está creado. 5) Crear máximo número de ficheros permitidos con nombres únicos. 6) Crear más ficheros que el máximo permitido.
Traza	El tamaño máximo del nombre de un fichero será 64 caracteres Solamente se creará un fichero si no existía previamente. Como mínimo, se deberá controlar el error de intentar crear un fichero cuando el Sistema de Ficheros no pueda almacenar más ficheros.
Resultado esperado	1) Fichero longitud 0 → Error 2) Fichero longitud máxima → Éxito 3) Fichero longitud superior a máxima → Error 4) Fichero ya creado → Error 5) Crear máximo número de ficheros → Éxito 6) Crear más ficheros que los permitidos → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 6: Prueba 6

Prueba 7	
Objetivo	Abrir un fichero: 1) Inexistente. 2) Existente y cerrado. 3) Existente y abierto.
Traza	No está como tal en el enunciado, pero se entiende que, si el fichero no existe o está ya abierto debe fallar. Solamente debe poder abrir un fichero existente y abierto.

Prueba 7	
Resultado esperado	1) Fichero inexistente. → Error 2) Fichero existente y cerrado. → Éxito 3) Fichero existente y abierto. → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 7: Prueba 7

Prueba 8	
Objetivo	Cerrar un fichero: 1) Fichero inexistente. 2) Fichero existente y abierto. 3) Fichero existente y cerrado.
Traza	No está como tal en el enunciado, pero se entiende que, si el descriptor no corresponde a ningún fichero, o a un fichero ya cerrado debe fallar. Solamente debe poder cerrar un fichero existente y abierto.
Resultado esperado	1) Fichero inexistente. → Error 2) Fichero existente y abierto. → Éxito 3) Fichero existente y cerrado. → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 8: Prueba 8

Prueba 9	
Objetivo	Escribir y leer de un fichero: 1) Escritura de todo el fichero con carácter 't'. 2) Lectura del fichero anterior. 3) Posicionar puntero al comienzo del fichero. 4) Escritura de la mitad del fichero con el carácter 'z'. 5) Posicionamiento a ¼ del fichero. 6) Lectura de todo el fichero del fichero. 7) Lectura de todo el fichero. 8) Lectura de un fichero inexistente/cerrado. 9) Escribir en un fichero inexistente/cerrado. 10) Escribir en un fichero con puntero de posición a X y lectura mayor que X + tamaño del fichero.

Prueba 9	
Traza	1, 2, 3 → Deberá devolver el número de bytes leídos/escritos. Posiciona el puntero en la posición inicial del fichero. En este caso se ignorará el segundo parámetro 5→ Lee datos de un fichero desde la posición indicada por el puntero de posición asociado al descriptor y si el número de bytes a leer es mayor que el número de bytes restantes, se leerán todos los posibles, y se deberá devolver esa cantidad. Posiciona el puntero en la posición indicada como parámetro desde la posición actual del puntero 6→ Si se intenta leer cuando no queda ningún byte disponible, se deberá devolver "0". 7,8→ "-1" en caso de error. 9→ "0" si el puntero del descriptor se encuentra en el tamaño máximo del fichero, o "-1" en caso de error. Si el número de bytes a escribir es mayor que el número de bytes restantes a partir del puntero de posición, se considerará un error y no se escribirá ningún byte. 10→ A modo de ejemplo, se considera un error colocar el puntero en una posición superior a la longitud actual del fichero o colocarlo en una posición menor que 0.
Resultado esperado	1) Escritura de todo el fichero con carácter 't'. → tamaño del fichero 2) Lectura del fichero anterior. → tamaño del fichero 3) Posicionar puntero al comienzo del fichero. → 0 4) Escritura de la mitad del fichero con el carácter 'z'. → tamaño del fichero/2 5) Posicionamiento a inicio y luego a ¼ del fichero. → puntero posición primero a 0 y luego a tamaño de fichero/4 6) Lectura de todo el fichero del fichero. → leer solo ¾ partes del fichero al estar el puntero en la ¼ parte. 7) Lectura de todo el fichero. → Al estar al final del fichero debe devolver 0. 8) Lectura de un fichero inexistente/cerrado. → -1 9) Escribir en un fichero inexistente/cerrado. → -1 10) Escribir en un fichero con puntero de posición a X y lectura mayor que X + tamaño del fichero → -1 Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 9: Prueba 9

4.3 Gestión de etiquetas

Prueba 10	
Objetivo	Asociar etiqueta a fichero: 1) Crear una etiqueta inexistente y que no se pueda asociar a ningún fichero. 2) Crear etiqueta inexistente y asociar a dos ficheros. 3) Asociar el fichero una etiqueta ya asignada. 4) Listar los ficheros de la etiqueta creada. 5) Asociar a un fichero tres etiquetas. 6) Asociar a un fichero una cuarta etiqueta. 7) Listar los ficheros de las tres etiquetas. 8) Desvincular las tres etiquetas del fichero. 9) Listar las tres etiquetas. 10) Listar primera etiqueta 11) Desvincular una etiqueta inexistente. 12) Desvincular una etiqueta no asociada 13) Crear una nueva etiqueta y que no hay espacio en el sistema de ficheros, pero si en el fichero.
Traza	Por norma general se aplican a los puntos indicados, sin embargo, ha puntos como por ejemplo los que tiene que dar error o en el caso de listar las listas que aparecen más de una vez por lo que no se ponen todos los puntos donde se prueba cada requisito. 1→Se considerará un error realizar esta operación cuando el fichero esté abierto o no exista 9→nueva etiqueta, haya espacio para crear una nueva etiqueta en el Sistema de Ficheros 4→"1" si la etiqueta ya estaba asociada al fichero 6→el fichero tenga menos de 3 etiquetas asociadas 7→Si, al llamar a esta función, la etiqueta deja de estar asociada a ningún fichero, deberá ser eliminada de los metadatos 12→"1" si la etiqueta no estaba asociada "-1" en caso de error 10→Deberá devolver el número de ficheros encontrados

Prueba 10	
Resultado esperado	1) Crear una etiqueta inexistente y que no se pueda asociar a ningún fichero. → Error 2) Crear etiqueta inexistente y asociar a dos ficheros. → Éxito 3) Asociar el fichero una etiqueta ya asignada. → etiqueta ya asignada al fichero. 4) Listar los ficheros de la etiqueta creada. → 1 fichero 5) Asociar a un fichero tres etiquetas. → Éxito 6) Asociar a un fichero una cuarta etiqueta. → Error 7) Listar los ficheros de las tres etiquetas. → el fichero asociado 8) Desvincular las tres etiquetas del fichero. → Éxito 9) Listar las tres etiquetas. → Error porque se borran la no tener ningún fichero asociado. 10) Listar primera etiqueta → un fichero, antes 2 ficheros. 11) Desvincular una etiqueta inexistente. → Error 12) Desvincular una etiqueta no asociada → 1 13) Crear una nueva etiqueta y que no hay espacio en el sistema de ficheros, pero si en el fichero. → Error Resultado final esperado: SUCCESS
Resultado obtenido	SUCCESS

Tabla 10: Prueba 10

5 Conclusiones

Esta práctica ha resultado muy interesante de realizar, ya que a medida que se iban realizando las funciones podíamos ir viendo cómo se iban reflejando los cambios en el disco.

No ha sido una práctica muy compleja de implementar una vez teníamos el diseño del sistema de ficheros, aunque a medida que íbamos realizando pruebas veíamos ciertos errores que teníamos que corregir. Gracias a ir realizando casos de prueba sobre cada función, previo a la implementación de la siguiente, podíamos avanzar con mayor seguridad de no tener errores que pudieran ir creciendo.

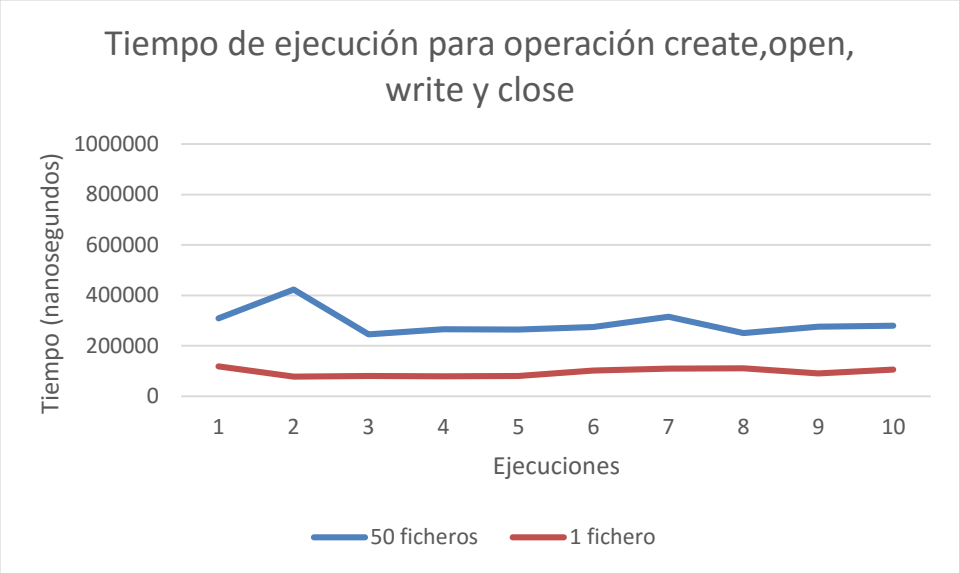
De esta manera, conseguimos depurar poco a poco el código. Finalmente realizamos todas las pruebas, para comprobar que lo que fuimos realizando era correcto en su totalidad.

Por falta de tiempo, se ha decidido no incluir una función extra que fuera sync(), que se encargara de volcar los metadatos del sistema de ficheros en memoria al disco cada cierto tiempo, asegurando la consistencia y tolerancia a fallos de nuestro sistema.

Como extra, hemos tomado tiempo de ejecuciones para ver cómo se comporta el sistema de ficheros a la hora de crear un fichero, abrirlo, escribir un bloque completo y cerrar el fichero.

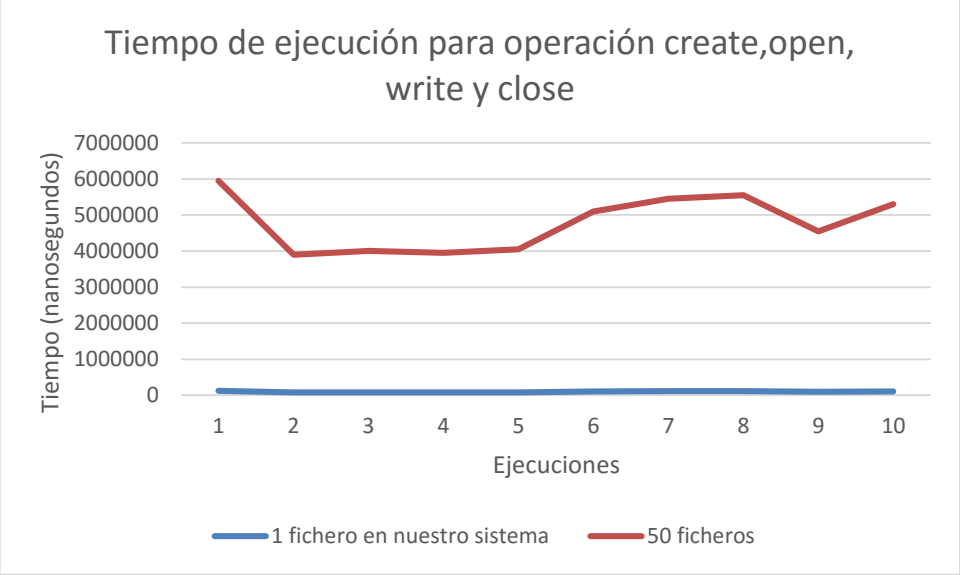
Para ello hemos realizado 10 ejecuciones para 50 ficheros, y 10 ejecuciones para 1 fichero. Y a modo de comparación, hemos introducido una tercera componente que serían 10 ejecuciones si nuestro sistema de ficheros fuera lineal de tiempo para cada fichero, es decir, si un fichero tarda ‘X’ segundos, 2 ficheros 2*’X’ segundos.

Todos los tiempos están tomados en nanosegundos.



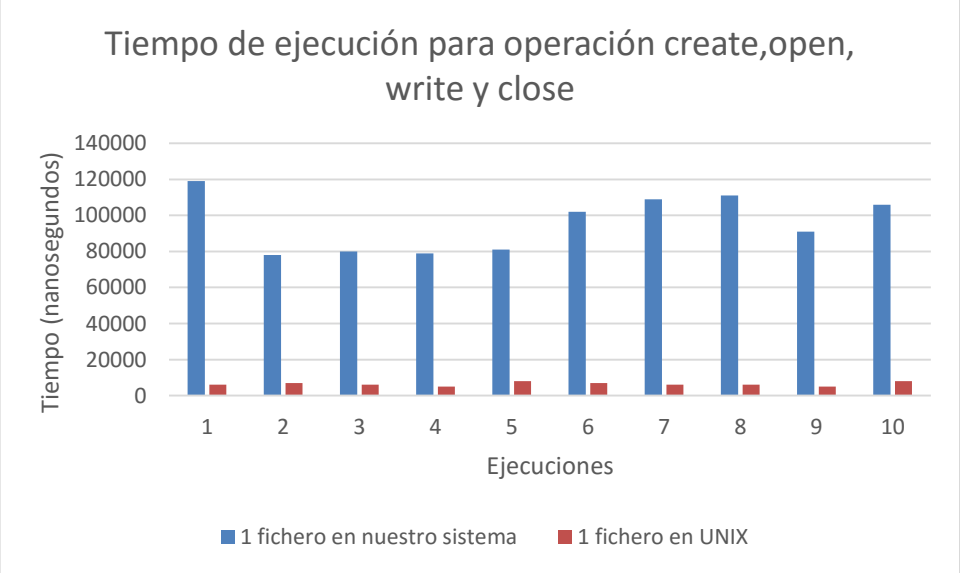
Como podemos ver nuestro sistema parece que es escalable, ya que el tiempo para realizar las operaciones indicadas sobre un fichero no escala de manera lineal para 50 ficheros.

Pero, ¿qué pasaría si nuestro sistema escalara de manera lineal?



Esta gráfica detalla que pasaría si nuestro sistema escalara de manera lineal, comparando el gráfico anterior con este, vemos como el tiempo de 50 ficheros en caso de escalar de manera lineal, se alejan mucho de los tiempos para 50 ficheros creados realmente.

Otra comparación curiosa es comparar nuestro sistema de ficheros con el sistema de fichero de UNIX, obviamente son comparaciones que deberían salir desfavorables ya que es una simulación de un sistema de ficheros propio sobre un sistema de ficheros UNIX.



Como podemos ver, nuestro sistema de ficheros en caso de comportarse de esta manera no sería eficiente. Aunque no podemos extrapolar los resultados, porque lo nuestro es una simulación como hemos dicho anteriormente.