uc3m | Universidad Carlos III de Madrid

Memoria P2

Diseño de sistemas operativos

Realizado por:
Gonzalo Fernández García; NIA: 100383212
Daniel Romero Ureña; NIA: 100383331
Marcelino Tena Blanco; NIA: 100383266

Índice

Introducción

Este documento se divide en cuatro grandes bloques centrados en explicar detalladamente el desarrollo de la práctica:

- **Diseño implementado:** en esta sección del documento se explica el diseño implementado en la práctica, la justificación a la elección de dicho diseño frente a otros posibles, las estructuras de datos que este utiliza y los algoritmos y optimizaciones que este diseño contiene.
- **Explicación del diseño implementado:** este apartado se centra en explicar el funcionamiento de la funcionalidad principal del programa en lenguaje C, mediante el código implementado, así como las funcionalidades centradas en integridad y enlaces simbólicos.
- **Batería de pruebas:** en esta sección se documentan varios conjuntos de pruebas para comprobar el correcto funcionamiento del sistema de ficheros previamente implementado, mostrando diferentes casos de prueba donde que cuentan con los siguientes parámetros: objetivo, procedimiento, entrada esperada y salida esperada.
- **Conclusiones sobre la práctica:** en este apartado del documento se presentan unas conclusiones finales sobre el desarrollo de la práctica, así como los problemas encontrados durante el desarrollo de la misma.

Descripción del diseño implementado

El diseño seleccionado consta de una arquitectura constituida por i-nodos con una asignación discontinua, en concreto un mecanismo enlazado, donde se indica en el inodo los siguientes bloques.

Se ha decidido optar por un mecanismo enlazado, ya que este cuenta con la ventaja de poder dispersar los bloques de un determinado fichero a lo largo de todo el espacio de disco, ya que se cuenta con un puntero de siguiente bloque almacenado en el i-nodo del fichero. De esta forma, se simplifica la gestión y el añadido de nuevos ficheros al sistema, ya que únicamente se debe almacenar en bloques de datos que se encuentren libres, en lugar de compactar el disco para agrupar bloques contiguos pertenecientes al mismo fichero como sí que se hace en el caso de una asignación continua, evitando de esta manera la posible fragmentación externa del disco.

Este diseño contempla únicamente un espacio de almacenamiento desde 460KiB hasta 600Kib, teniendo como máximo tamaño de fichero 10KiB pudiendo almacenarse la información en bloques de un tamaño fijo de 2048 bytes. Además de esto, se ha establecido un bloque por i-nodo, siendo cada i-nodo encargado de direccionar a un único fichero. De esta forma, el máximo número de bloques de i-nodos que puede darse en el sistema es de 48, siendo este el número máximo de archivos que puede contener el sistema.

Se ha tomado la decisión de asignar cada i-nodo en un bloque distinto para fragmentar la información contenida en ellos, facilitando de esta forma el acceso a su información.

Otro beneficio que aporta esta decisión es el hecho de facilitar la creación y posicionamiento de i-nodos, ya que a la hora generar uno nuevo solo se deberá buscar un bloque de i-nodos libre en el mapa de i-nodos, evitando el hecho de comprobar en cada uno de los bloques ocupados si hay espacio suficiente para poder añadir este nuevo i-nodo.
Como contraparte negativa tenemos el hecho de que al realizar este tipo de estructura se produce una falta de optimización del espacio de disco, ya que el espacio completo de un bloque solo es usado para almacenar la información de un único i-nodo, lo que supone un desperdicio en el espacio de bloque.
En cuanto a los bloques de datos, se ha establecido que un fichero pueda tener hasta un máximo de 6 bloques de datos, eso es debido a que además de los 5 bloques de 2048 bytes necesarios para abarcar el tamaño máximo de archivo (10KiB), se ha establecido un último bloque de fichero denominado bloque EOF el cual tendrá como función señalizar el fin de fichero, el cual estará presente para cada uno de los ficheros generados.
En lo referido al tamaño de puntero, el sistema de ficheros cuenta con punteros de 16 bits debido a que de esta forma se permite direccionar el tamaño de disco especificado, además de reducir el espacio de almacenamiento de dichos punteros, lo que supone un aumento del espacio disponible para otros atributos o parámetros dentro de los bloques.
Mediante estos datos se puede deducir que el número máximo de bloques de datos, así como el número máximo de bloques de datos vendrán dados por la división del máximo tamaño de disco posible (600 KiB) entre el tamaño de bloque (2048 bytes).
En lo referido a los enlaces simbólicos, se ha establecido que, tras borrar un fichero, los enlaces simbólicos asociados a este no sean eliminados, siguiendo como referencia los sistemas de ficheros utilizados por Windows o Posix. Además de esto, se contempla la posibilidad de realizar enlaces simbólicos de otros enlaces simbólicos, ya que la mayoría de sistemas de ficheros actuales, en concreto los anteriormente mencionados, cumplen dichas características.

Estructura del sistema de ficheros:

Superbloque	Bloque de mapa de i-nodos	Bloque de mapa de datos	Bloques de i-nodos	Bloques de datos	EOF
1 bloque	1 bloque	1 bloque	48 bloques	n bloques	

- **Superbloque:** es el primer bloque del sistema de ficheros y almacena datos importantes del mismo, siendo estos almacenados al inicio de dicho bloque en el siguiente orden:

- Número mágico: este campo se encarga de almacenar el número que identifica el sistema de ficheros, siendo este el NIA de uno de los integrantes del equipo.
- Primer i-nodo: almacena la posición del primer bloque de i-nodos dentro del sistema de ficheros.
- Número de bloques de mapa de i-nodos: indica el número de bloques de mapa de i-nodos presentes en el sistema de ficheros.
- Número de bloques de i-nodos: almacena el número de bloques de i-nodos presentes en el sistema de ficheros, teniendo un valor máximo de 48.
- Primer bloque de datos: almacena la posición del primer bloque de datos dentro del sistema de ficheros.
- Número de bloques de mapas de datos: indica el número de bloques de mapas de datos que se encuentran en el sistema de ficheros.
- Número de bloques de datos: representa el número de bloques que contiene el sistema de ficheros.
- Ficheros en el sistema: indica el número de ficheros presentes en el sistema, todo ello con el objetivo de prevenir el uso de más de 48 ficheros, lo cual superaría el límite establecido.
- Tamaño del dispositivo: indica el tamaño del dispositivo de almacenamiento a gestionar por el sistema de ficheros.

- **Bloque de mapa de i-nodos:** es el encargado de almacenar el mapa de inodos, el cual tiene como función el registrar si cada uno de los inodos presentes en el sistema de ficheros se encuentra libre u ocupado mediante un array de bits que marca con un 0 si dicha posición del array está libre o con cualquier otro valor si está ocupada.
- **Bloque de mapa de datos:** tiene la función de almacenar el mapa de datos del sistema de ficheros, el cual se basa en un array de bits que almacena la disponibilidad de los bloques de datos mediante el uso de 0 para indicar que dicho bloque se encuentra libre y cualquier otra cosa distinta de 0 para indicar que se encuentra ocupado.
- **Bloques de i-nodos:** contiene los diferentes bloques que almacenan todos los i-nodos presentes en el sistema, teniendo un i-nodo almacenado por bloque, por lo que el número máximo de bloques que podrá darse en esta sección es de 48.

  Cada uno de los i-nodos contenidos en los bloques contará con los siguientes campos:
  - Nombre del fichero: contiene el nombre del fichero asociado a dicho i-nodo.
  - Referencias: se basa en un array de 6 elementos tipo short que almacenan las diferentes referencias de los bloques de datos que contendrán los datos del fichero.
  - Tamaño: almacena el tamaño del fichero al que está asociado.

    - Integridad: En caso de que el fichero asociado tenga integridad contiene el CRC del fichero.
- **Bloques de datos:** esta sección incluye los bloques encargados de almacenar la información de los diferentes ficheros presentes en el sistema de ficheros, pudiendo darse el caso de verse fragmentado un fichero en varios bloques de datos.

Para gestionar los ficheros abiertos, vamos a hacer uso de descriptores de fichero que serán almacenados únicamente en memoria, pudiendo haber un total de 48 descriptores de fichero debido a la limitación del número máximo de ficheros en el sistema.

Los descriptores de fichero contarán con los siguientes campos:

- Nombre: nombre del fichero abierto que representa el descriptor.
- Puntero al i-nodo: puntero al inodo asociado al fichero abierto.
- Puntero RW: puntero de lectura-escritura del fichero, el cual indicará la posición dentro del mismo donde se comenzará a leer o escribir.
- Abierto con integridad: número que indicará mediante un 1 si el archivo se ha abierto con integridad o con un 0 si se ha abierto sin ella.

En cuanto a decisiones de diseño que afectan a la optimización de la implementación del sistema de ficheros en código C se ha decidido hacer uso de short como tipo de variable en lugar de int debido a que short ocupa un menor tamaño en bytes, permitiéndonos de esta manera ahorrar espacio de bloque. Además de esto, se ha decidido establecer la declaración mediante unsigned en la declaración de parámetros dentro de las estructuras con el objetivo de evitar el almacenamiento del signo, todo ello con el objetivo de ahorrar espacio en almacenamiento.

Explicación del diseño implementado

Funcionalidad básica:

En primer lugar, dentro de esta sección se va a explicar el funcionamiento de las funcionalidades básicas estas son funciones creadas con el objetivo de gestionar: el dispositivo, los ficheros en disco y por último la interacción con los ficheros.

- mkFS: función encargada de generar la estructura del sistema de ficheros diseñado. En primer lugar la función crea el superBloque con ayuda de la función createSuperBloque() que se encarga de rellenar los campos de la estructura superbloque definida previamente en el documento, después de esto escribe esta estructura en el bloque 0 del disco. Su siguiente acción es la de crear el mapa de bits de los bloques de Inodos y los de datos, iniciando el mapa a 0 en todas sus posiciones ya que en este momento el sistema de ficheros está vacío, estas dos estructuras se escriben en los bloques 1 y 2 del disco respectivamente. Su siguiente función es la de crear el bloque EOF usado para indicar el final de un archivo, este se escribe en el último bloque del disco. Su última función es la de crear los inodos vacíos y escribirlos en los

bloques correspondientes del disco. Antes de acabarse libera de la memoria las estructuras que ha utilizado para iniciar el sistema de fichero.

- mountFS: esta función se encarga de montar la partición del sistema de ficheros en memoria, para ello, se deben pasar los contenidos de este sistema previamente generado mediante la función mkFS. Para realizar este montaje en memoria se deberá leer cada una de las secciones del sistema, las cuales son: superbloque, bloque de mapa de i-nodos, bloque de mapa de datos, bloques de i-nodos y bloques de datos. Cada una de estas secciones será leída de forma individual para asegurar su correcta generación, para que una vez se verifique esto, se pase a reservar espacio en memoria para los datos contenidos en cada sección del sistema de ficheros mediante la función malloc. Una vez se reserva este espacio en memoria se pasará a almacenar los datos de cada sección en memoria mediante el uso de diferentes funciones.
  En el caso del superbloque, se hace uso de la función chartoSB, la cual se encargará de realizar una copia de cada uno de los parámetros contenidos en el superbloque mediante la función memcpy dentro del espacio de memoria reservado. En el caso de las secciones restantes salvo los bloques de y nodos se utiliza la función memcpy para realizar la copia del contenido dentro de la memoria reservada, diferenciándose únicamente la sección de bloques de i-nodos por utilizar la función chartoInodo para hacer una copia de los parámetros almacenados en cada i-nodo mediante el uso de memcpy.
- unmountFS: método encargado de desmontar el sistema de fichero. Su funcionalidad consiste en escribir en disco los datos que el sistema de ficheros tiene en memoria para ello usa la función syncDisk() que escribe en el disco el contenido del superbloque, los mapas de bits y los Inodos. Tras realizar estas escrituras en el disco libera mediante free la memoria de los datos escritos en el disco.
- createFile: función que crea un nuevo fichero vacío en el el sistema de ficheros. Antes de realizar su función comprueba que no exista un fichero con el mismo nombre que el que se pretende crear, que no se haya alcanzado el número máximo de ficheros y que el nombre recibido por parámetros no ocupe más de 32 bytes. Una vez realizadas las comprobaciones busca un inodo vacío en el mapa de inodos mediante un for y la función bitmap_getbit(), si no se encuentra ninguno libre se devuelve -2 y se lanza un perror(). En caso de encontrar un Inodo vacio lo inicia escribiendo el campo de nombre del fichero con el nombre recibido por parámetro además de actualizar el mapa de inodos mediante bitmap_setbit() escribiendo un 1 en la posición del mapa correspondiente al inodo utilizado y por ultimo actualiza en el superbloque el número de ficheros totales.
- removeFile: esta función se implementa con el objetivo de que elimine un archivo del sistema de ficheros. En primer lugar, comprueba la longitud del nombre que recibe la función para evitar que sea superior a 32 bytes, en segundo lugar, comprueba mediante existeFichero si alguno de los inodos

tiene como campo de nombre el nombre que recibe la función. En caso de que estas comprobaciones sean satisfactorias cierra el fichero en caso de que se encuentre abierto, ya que se intuye que se quiere eliminar, vacía los campos del inodo asociado al archivo, actualiza los mapas de bits de los inodos y los datos para indicar que están libres y por último reduce una unidad al valor del superbloque que indica la cantidad de ficheros total que hay en el disco.

- openFile: el objetivo principal de esta función es el de crear un descriptor de fichero, para ello, tras realizar diferentes comprobaciones de los parámetros de entrada a la función, se comprueba que el fichero exista dentro del sistema de ficheros, si se da el caso de que ya existe, se devolver el descriptor donde este está abierto. En caso de que el fichero no exista, se busca un descriptor de fichero vacío y tras abrir el fichero mediante el uso de la función memcpy y situar el puntero de lectura-escritura al comienzo de este, se devuelve el descriptor de fichero vacío previamente encontrado.
- closeFile: la funcionalidad que realiza este método es la de cerrar un archivo abierto en el sistema de ficheros y para llevarla a cabo antes de nada realiza dos comprobaciones en primer lugar que el descriptor de fichero que recibe es válido es decir ni negativo ni superior a 48 y en segundo lugar comprueba que ese descriptor de ficheros corresponda a un archivo que está abierto. Si las pruebas son correctas procede a cerrar el fichero, eliminado el descriptor de fichero del array de descriptores. Además de esto, se incluye una comprobación que se encarga de confirmar si el fichero se ha abierto mediante integridad, por lo que en caso afirmativo se producirá un error.
- readFile: tiene como finalidad el leer el contenido de un fichero y devolver el número de bytes leídos, para ello se hará uso de un puntero char denominado lectura bloques, el cual almacenará el contenido leído del archivo, además de esto, se hará uso de un puntero de lectura, el cual estará situado inicialmente donde está situado el puntero lectura-escritura.

  Para realizar la lectura del archivo, se hace uso de un bucle while en el cual se irán obteniendo los diferentes bloques que contienen al fichero hasta llegar al bloque EOF o que el número de bytes leídos lleguen a ser los mismos que los que se desean leer. Una vez se obtenga un bloque en cada iteración del bucle, se hará una llamada a la función bread mediante la cual se leerá el contenido del bloque en cuestión, actualizando el número de bytes leídos y el índice de bloque para la obtención del siguiente bloque a leer.

  Una vez leído el fichero, se comprueba si el número de bytes leidos ha excedido el número de bytes introducidos por parámetro, adaptando la copia de los datos leídos en función de esta situación. Además de esto, se actualiza el puntero de lectura-escritura mediante el puntero de lectura y se devuelve el número de bytes leídos durante el bucle while.

  Cabe destacar, que el número de bytes máximo que esta función puede leer es de 6159 bytes, siendo necesarias varias llamadas a esta función para leer tamaños de archivo superiores a este.

- writeFile: esta función tiene la finalidad de escribir en un cierto fichero, un determinado número de bytes contenidos en un buffer como parámetro de entrada. El funcionamiento de este método comienza obteniendo los bloques del fichero donde se quiere escribir, ya que se puede sobre escribir en el fichero. Para ello, y para escribir necesitamos el número de bloques necesarios para almacenar la información que desea escribir en el fichero. Después de obtener el contenido anterior, se procederá a escribir de forma iterativa en los diferentes bloques dados por el descriptor de fichero del parámetro de entrada. En caso de necesitar más de un bloque para la escritura en el fichero, será necesario obtener bloques de datos libres mediante el mapa de bits, pasando estos a estar ocupados, en estos nuevos bloques se escribirá el contenido del buffer mediante la función bwrite de forma iterativa en el bucle hasta que el número de bloques escritos sea igual al número de bloques necesarios.
  Una vez realizada la escritura se actualizará el puntero lectura-escritura al igual que en la función readFile.
- lseekFile: el objetivo de esta función es el de modificar el valor del puntero de posición de un fichero para ello recibe el descriptor del fichero a modificar, un valor llamo whence que indica si la posición del puntero pasa a ser el inicio del documento, el final de documento o la posición actual del puntero más un desplazamiento y por último la cantidad de posiciones que se debe desplazar el puntero. Antes de nada, realiza comprobaciones para verificar que es un descriptor de fichero válido y que está en uso, es decir que hay un fichero abierto asociado a él. Si estas comprobaciones son positivas se ejecuta un switch con whence en el que el caso 0 se interpreta como mover el puntero desde la posición actual más el offset y se realiza siempre que la nueva posición del puntero no exceda los límites del fichero, el segundo caso supone poner el puntero en la posición 0 del archivo y el tercer caso indica poner la posición del puntero al final del fichero. El caso default supone un error al no haber introducido un whence válido.

Implementación de integridad:

En esta parte vamos a explicar el funcionamiento de las funciones creadas para la implementación de la integridad y la comprobación de esta sobre los distintos archivos existentes en el sistema de ficheros.

- checkFile: es el encargado de comprobar la integridad de un fichero, para llevarlo a cabo la función se obtiene el i-nodo del fichero cuyo nombre ha sido introducido por parámetro y se obtiene su descriptor. Una vez obtenidos estos valores, se pasa a comprobar si el fichero cuenta con integridad actualmente mediante el mediante su i-nodo para finalmente ser contrastado con una nueva integridad creada a partir de este mismo i-nodo, verificando la integridad del fichero si ambas integridades son iguales.
- includeIntegrity: función que se encarga de incluir la integridad en un fichero que no la tiene. Tras realizar comprobaciones acerca del nombre de fichero

recibido y de si este tiene ya implementada integridad llama a la función crearIntegridad() que recoge el contenido de todos los datos del fichero y realiza el CRC 32 sobre ellos. El valor que retorna la función es guardado en el campo integridad del inodo asociado al fichero.

- openFileIntegrity: su principal funcionalidad es la de abrir un fichero con integridad realizando un check durante este proceso. En primer lugar comprueba que exista el fichero, que no esté abierto ya o la longitud del nombre recibido, tras estas comprobaciones llama a la función check para asegurarse de que mantiene la integridad y por último realiza un proceso similar al descrito para la función openFile() con la diferencia de que marca el campo abierto con integridad que será usado más tarde para obligar al usuario a cerrar con integridad el fichero.
- closeFileIntegrity: su cometido es el de cerrar un archivo con integridad para realizar esta tarea comprueba que el descriptor de fichero recibido sea válido y esté abierto con integridad en otro caso enviará un error. Tras las comprobaciones mediante crearIntegridad() actualiza el campo de la integridad en el inodo correspondiente además de liberar el descriptor de fichero.

Enlaces simbólicos:

Para finalizar esta sección del documento procedemos a explicar las funciones creadas para la implementación de los enlaces simbólicos.

- createLn: el objetivo de la función es el de crear un enlace simbólico a un archivo ya existen en el sistema de ficheros. En primer lugar, comprueba que los parámetros que recibe sean válidos y si el fichero existe en el sistema obteniendo así el inodo que le corresponde, una vez realizadas estas operaciones busca el primero de los inodos libres para asignarlo al enlace. Finalmente rellena el inodo disponible igualándolo al inodo del fichero y después modificando el campo del nombre además de aumentar una unidad el número de archivo ya que los enlaces cuentan como ficheros en nuestro diseño y actualizando el mapa de inodos.

- removeLn: tiene como finalidad eliminar el enlace simbólico de un determinado fichero cuyo nombre es recibido como parámetro de entrada. Para llevar a cabo esta finalidad, se busca la ubicación del fichero en el disco, obteniendo el número del i-nodo asociado al dicho archivo, en caso de que este exista. Para finalizar se eliminará el descriptor de fichero asociado y se realizará un formateo del i-nodo relacionado a este fichero estableciendo el bloque de i-nodos que contiene a este i-nodo como libre mediante el mapa de i-nodos.

Batería de pruebas

Funcionalidad básica

Objetivo	Procedimiento	Salida esperada	¿Salida esperada obtenida?
Funcionamiento correcto de las funcionalidades básicas.	Ejecución de las siguiente funciones en el orden: montaje, creación, abrir, escritura, crear, escribir, lseek, leer, cerrar, eliminar, desmontar.	Todas las funcionalidades funcionan correctamente.	SI
Intentar crear un fichero con nombre de otro ya existente.	Se crea un fichero, para posteriormente crear otro con el mismo nombre.	La prueba detecta el fallo que indica que existe un fichero con el mismo nombre.	SI
Crear un fichero con tamaño de nombre no válido.	Crear un fichero con tamaño de nombre superior a 32 bytes.	La prueba detecta el fallo que indica que el tamaño del nombre no es válido.	SI
Abrir un fichero que no inexistente.	Pasar a la función open un nombre de un fichero que no se ha creado.	La prueba detecta el fallo que indica que el fichero no existe.	SI
Cerrar un fichero no abierto previamente.	Cerrar un fichero que previamente no se ha abierto.	La prueba detecta el error que indica que el fichero no está abierto.	SI
Eliminar un fichero abierto.	Crear un archivo, abrirlo y tratar de eliminarlo sin haberlo cerrado antes.	La prueba se ejecuta correctamente cerrando el archivo antes de eliminarlo.	SI
Eliminar un fichero que no existe.	Pasar a la función removeFile por parámetro un fichero que no existe.	La prueba detecta el error e indica que el fichero no existe.	SI
Leer más número de bytes que el tamaño de fichero.	Llamada a read con un número de bytes a leer mayor que el tamaño del fichero.	No da error devuelve el número de bytes realmente leídos.	SI

Leer más bytes que el tamaño máximo del buffer de lectura.	Llamada a read para que lea más bytes que la capacidad del buffer.	La función agranda el buffer para que entre la cantidad de bytes solicitados	SI
Escribir para sobrepasar el tamaño máximo del fichero.	Llamada a write para que escriba más bytes que la capacidad máxima del fichero.	La prueba se ejecuta correctamente escribiendo únicamente los bytes hasta llegar al límite del fichero.	SI
Escribir un fichero de más de un bloque.	Llama a write pasandole un buffer de más de 2048 bytes para que los escriba.	Funcionamiento correcto de la función write.	SI
lseek a posiciones negativas o más grandes que el tamaño de fichero.	Realizar lseek con un offset muy grande ya sea negativo o positivo.	La prueba detecta el error e indica que el offset hace exceder al puntero lo límites del fichero.	SI
lseek haciendo uso de un whence inválido.	Llamar a lseek pasando como argumento whence un 3.	Indica un error de que el parámetro whence no es válido.	SI
Superar el máximo de ficheros permitidos.	Crear tantos fichero como para sobrepasar la cantidad máxima del sistema de ficheros	Sale un error indicando que se ha alcanzado el máximo de ficheros	SI
Abrir más archivos del máximo permitido.	Realizar llamadas a la función open sobre muchos archivos de tal manera que sobrepasen la cantidad máxima de ficheros abiertos.	Sale un error indicando que se ha alcanzado el máximo de ficheros y que no se encuentra el archivo que tratamos de abrir ya que no se ha podido crear.	SI
Sobrescribir una parte del fichero	Realizar un llamada a write con el puntero WR en medio del archivo	Funcionamiento correcto del programa	SI
Leer un fichero cuyo contenido es igual al tamaño máximo de fichero.	Crear y escribir un fichero completo y luego ejecutar read sobre este.	Es capaz de leer hasta 6159 bytes en caso de intentar una cantidad mayor sale un error.	SI

Desmontar el sistema de ficheros sin cerrar todos los ficheros.	Usar la función unmountFS sin cerrar un archivo.	El funcionamiento es correcto ya que unmountFS cierra todos los fd abiertos	SI

Implementación de integridad

Objetivo	**Procedimiento**	**Salida esperada**	**¿Salida esperada obtenida?**
Funcionamiento correcto de integridad	Crear archivo, dar integridad, abrir con integridad, escribir sobre él, cerrar con integridad y chequear.	Funcionamiento correcto	SI
Verificar integridad de archivo sin integridad.	Llamar a la función checkFile con un fichero que no tiene integridad.	Un mensaje de error indica que el fichero no tiene integridad.	SI
Verificar integridad de archivo abierto.	Llamar a la función check con un fichero abierto previamente.	Un mensaje de error indica que el fichero ha sido abierto previamente.	SI
Verificar un archivo que se ha corrompido.	Abrimos un archivo con integridad lo escribimos y lo cerramos con integridad, después lo abrimos sin integridad lo modificamos lo cerramos sin integridad y hacemos el check.	Un mensaje de error indica que el archivo esta corrupto.	SI
Abrir un archivo con integridad y cerrarlo sin ella.	Llamar a openIntegrity y después closeFile.	Un mensaje de error indica que el fichero se abrió con integridad.	SI
Dar integridad a un archivo que ya la tiene.	Llamar a la función includeIntegrity dos veces sobre el mismo fichero.	Un mensaje de error indica que el archivo tiene integridad.	SI

Abrir con integridad un archivo sin integridad.	Llamada a openIntegrity pasando un fichero que no tiene integridad.	Un mensaje indica que el fichero no tiene integridad.	SI
Abrir un archivo sin integridad y cerrarlo con ella.	Llamar a openFile y después closeIntegrity.	Un mensaje de error indica que el fichero se abrió sin integridad.	SI

Enlaces simbólicos

Objetivo	**Procedimiento**	**Salida esperada**	**¿Salida esperada obtenida?**
Funcionamiento correcto de la creación y eliminación de los enlaces	Llamar a la función createLn, utilizar el enlace y llamar a removeLn después mediante read confirmar que el archivo no se ha eliminado	Funcionamiento correcto del programa	SI
Crear enlace a un fichero que no existe.	Llamar a la función createLn pasando un fichero que no existe en el sistema de fichero.	Un error indica que no existe el archivo	SI
Crear un enlace a otro enlace.	Llamar a la función createLn pasando un nombre de un enlace.	Ambos enlaces se crean sin ningún problema	SI
Eliminar un enlace que no existe.	Llamar a la función deleteLn pasando un fichero que no existe en el sistema de fichero .	Un error indica que el archivo no existe	SI
Eliminar el archivo y no se elimina el enlace	Crear un enlace a un fichero y eliminar dicho fichero	El enlace sigue existiendo	SI

Nota: El código de estas pruebas ha sido incluido en el archivo test.c en forma de comentario, hemos divido estas pruebas en pruebas de error y pruebas funcionales. Las pruebas de error se encargarán de detectar los errores producidos en una situación no aceptada por el sistema, por lo que obtener SUCCESS significa que se ha producido el error esperado. En el caso de las pruebas funcionales, estas se centrarán en verificar el correcto funcionamiento de las funcionalidades implementadas en el sistema de ficheros, por lo que obtener SUCCESS en una prueba significa un funcionamiento correcto de la funcionalidad.

Conclusiones sobre la práctica

En términos generales, consideramos que esta práctica ha sido muy constructiva para conocer el funcionamiento de los sistemas de ficheros y el proceso para realizar su diseño e implementación, debido a que se establece un marco de trabajo con ciertas limitaciones y requisitos, pero que no impide que el proceso de diseño del sistema de fichero sea bastante libre. Este diseño se completa con la creación de no solo funcionalidades básicas sino también algunas más avanzadas como la integridad y los enlaces simbólicos, que requieren un extra al tener que adaptar el diseño a estas necesidades.

En cuanto a los problemas encontrados durante la realización de la práctica han sido principalmente dos por una parte entender el funcionamiento de los archivos proporcionados para la realización de la práctica y por otra parte que en algunas funciones como writeFile o readFile teníamos que tener en cuenta una gran cantidad de diferentes situaciones y posibles errores.

En conclusión, creemos que se trata de una práctica bastante instructiva y útil de cara a aumentar y reforzar nuestros conocimientos acerca de los sistemas de ficheros y su diseño.