Diseño de un Driver para Sistema de Ficheros con FUSE

Grado de Ingeniería de Computadores
Diseño de Sistemas Operativos
Máximo García Aroca. Rodrigo Hernández Barba

Resumen **- Este artículo presenta el diseño y la implementación de un controlador para un sistema de archivos utilizando FUSE. El controlador soporta operaciones esenciales del sistema de archivos, incluyendo la creación, lectura, escritura, renombrado y apertura de archivos. Detallamos el proceso de desarrollo, destacando la flexibilidad y funcionalidad de FUSE para interactuar con sistemas de ficheros en el espacio de usuario. La implementación pretende demostrar aplicaciones prácticas de conceptos teóricos en el diseño de sistemas de archivos, optimizar el rendimiento mediante una gestión eficiente de los recursos y garantizar la compatibilidad y la eficiencia. Funcionalidades clave como getattr, readdir, open, read y write se discuten en profundidad, mostrando su papel en la consecución de un sistema de archivos robusto y escalable. Este trabajo contribuye a la comprensión de las capacidades de FUSE y su potencial para diversas aplicaciones, desde el acceso remoto a archivos hasta la comunicación entre dispositivos. Los resultados subrayan la eficacia del controlador implementado a la hora de proporcionar una interfaz de sistema de archivos coherente y eficiente.**

I. INTRODUCCIÓN

La idea de que "todo es un archivo" es una de las principales aportaciones del sistema operativo Unix, ofreciendo una representación común para diversos objetos, desde el almacenamiento de información hasta las estructuras internas del sistema. Esta forma de pensar ha posibilitado asignar muchos objetos a la idea abstracta de archivos, facilitando la interacción con ellos.

Hasta hace poco, esta manera de pensar ha avanzado hacia la adopción de Sistemas de Archivos en el Espacio de Usuario (FUSE). FUSE habilita a los usuarios a visualizar su interacción con un objeto a través de una organización de directorios y funciones de sistema de archivos, seguido de la creación de un sistema de archivos FUSE para ofrecer dicha interacción. Esto se consigue mediante la utilización de funciones de archivos como open(), read() y write().

Los sistemas de archivos FUSE son altamente flexibles, facilitando desde la conexión remota a archivos en un servidor diferente sin depender de NFS o CIFS, hasta la creación de un sistema de archivos para comunicarse con dispositivos que emplean el protocolo de Transferencia de Medios. Básicamente, las opciones son tan vastas como la creatividad del usuario.

Este documento analiza detalladamente la ejecución y las usos de FUSE, resaltando su flexibilidad y capacidad para transformar la manera en que nos relacionamos con los sistemas de archivos.

II. OBJETIVOS PLANTEADOS

A. ***Implementación de un Controlador FUSE para Operaciones Básicas de Sistema de Archivos***

El objetivo principal es diseñar e implementar un controlador FUSE (*Filesystem in Userspace*) que soporte las operaciones básicas de un sistema de archivos. Estas operaciones incluyen:

- Operaciones de entrada y salida: Incluye la capacidad de crear, leer, escribir, renombrar y abrir archivos.
- Gestión de directorios: Implementación de funciones para crear, eliminar, leer, etc.
- Manipulación de atributos de archivos: Como establecer los atributos extendidos, obtener los atributos de un archivo e implantar los tiempos de acceso y modificación de un archivo.
- Otras funcionalidades: marcadas como opcionales para la extracción de metadatos, creación de enlaces y modificación de permisos.

B. ***Evaluación de la Compatibilidad entre Proyectos FUSE mediante Especificaciones Compartidas***

Este objetivo se centra en analizar y evaluar la compatibilidad entre distintos proyectos FUSE mediante la utilización de especificaciones compartidas.

C. ***Integración de i-Nodos y Conocimientos Teóricos Para un Desarrollo Exitoso del Proyecto***

Para lograr un desarrollo exitoso en el proyecto, es crucial integrar adecuadamente la teoría de sistemas de archivos explicados durante las clases teóricas de la asignatura.

D. ***Optimización del Rendimiento del Driver mediante el Diseño y Manejo Adecuado de Estructuras i-Nodo y Metadatos***

Optimizar el rendimiento del controlador FUSE es un objetivo clave, que se logrará a través del diseño eficiente y la gestión adecuada de las estructuras de i-nodos y metadatos. Esto incluye:

- Diseño eficiente de i-nodos: Crear estructuras i-nodo que minimicen el tiempo de acceso y maximicen la utilización del espacio.
- Gestión de la caché de metadatos: Implementar técnicas de caché para reducir el número de accesos a

disco, mejorando así la velocidad de las operaciones de archivo.
- Optimización de algoritmos de búsqueda y acceso: Utilizar algoritmos avanzados para la búsqueda y acceso de i-nodos y metadatos que reduzcan el tiempo de respuesta.
- Monitoreo y ajuste: Implementar herramientas para monitorear el rendimiento del sistema de archivos y ajustar dinámicamente las estrategias de gestión de i-nodos y metadatos.

E. Comunicación Intergrupal

La comunicación efectiva entre los miembros del grupo y otros grupos es esencial para el éxito del proyecto. Para ello, se establecen los siguientes objetivos:
- Establecimientos de canales de comunicación: Utilizar herramientas de comunicación para mantener a todos los miembros informados y coordinados.
- Documentación y seguimiento del progreso: Mantener una documentación detallada ya actualizada del progreso del proyecto, incluyendo decisiones de diseño, cambios en el código y resultado de pruebas.
- Revisión y retroalimentación: Revisión regular del trabajo realizado y proporcionar retroalimentación constructiva para mejorar continuamente el proyecto.

Estos objetivos formarán la base para un desarrollo exitoso y bien estructurado del proyecto, garantizando no solo la funcionalidad, sino también la eficiencia, la compatibilidad y la colaboración efectiva entre los miembros del grupo.

III. TRABAJOS SIMILARES

El proyecto *basicFUSE* previamente desarrollado en una de las prácticas de la asignatura nos hace una breve introducción al sistema de ficheros FUSE y el cómo este interactúa con el controlador y las operaciones que se implementan.

IV. DESCRIPCIÓN DE MECANISMOS Y HERRAMIENTAS UTILIZADAS

A. Librería FUSE

fuse_lib.h define estructuras y funciones relacionadas con la implementación de un sistema de archivos utilizando FUSE. El i-nodo raíz y el bloque de control son los elementos clave en esta implementación.

i. **Estructura *s_fuseInode***

Esta estructura proporciona un modelo detallado para representar archivos y directorios en un sistema de archivos FUSE, incluyendo metadatos cruciales como la identificación, nombre, ruta, tamaño, bloques de datos, relaciones jerárquicas, puntero a datos, permisos y tiempos. Esta estructura es fundamental para las operaciones de gestión de archivos y directorios en el sistema de archivos.

ii. **Estructura *s_controlBlock***

La estructura *s_controlBlock* es fundamental para el funcionamiento de un sistema de archivos. Mantiene información global y estados necesitados para la gestión eficiente del sistema de archivos, incluyendo la gestión del espacio libre, el rastreo de la creación y modificaciones del sistema, y el seguimiento de montajes. Esta estructura es utilizada para inicializar y mantener el sistema de archivos en un estado coherente y funcional durante su operación.

iii. **Función *init_fuseInode***

Esta función crea e inicializa un nodo raíz para un sistema de archivos, configurando todos sus campos a valores adecuados para representar el directorio raíz. Esta función es esencial para establecer la estructura básica del sistema de archivos y garantizar que el i-nodo raíz esté correctamente configurado con los metadatos necesarios.

iv. **Función *init_controlBlock***

La función *init_controlBlock* inicializa y configura un bloque de control (*s_controlBlock*) para el sistema de archivos. Establece los valores iniciales necesarios para los contadores de bloques e i-nodos libres, los tiempos de creación y montaje, los bitmaps de bloques, y crea el i-nodo raíz. Esta función es crucial para preparar el sistema de archivos para su uso, asegurando que todos los metadatos que todos los metadatos globales estén correctamente inicializados.

V. ESTRATEGIAS DE DISEÑO E IMPLEMENTACIÓN

A. I-Nodos

Los i-nodos son una parte esencial en el diseño de un sistema de archivos basado en FUSE, proporcionando una manera estructurada de almacenar y gestionar la información de los archivos y directorios. La implementación eficiente y robusta de i-nodos es crucial para el rendimiento, la seguridad y la funcionalidad general del sistema de archivos.

Algunas de las características que presentan los i-nodos son las siguientes:
- **Identificador único**: Cada i-nodo tiene un identificador único dentro del sistema de archivos.
- **Metadatos del archivo**: Almacenan información como el tamaño del archivo, permisos, tiempos de modificación, creación y acceso, y el número de enlaces.
- **Punteros a bloques de datos**: Contienen punteros directos e indirectos a los bloques de datos que contienen el contenido del archivo.
- **Atributos extendidos**: Pueden almacenar información adicional como comentarios del usuario y otros metadatos específicos del sistema de archivos.
- **Estructura fija**: La estructura de un i-nodo es generalmente fija y predefinida, lo que permite un acceso eficiente y consistente a los metadatos del archivo.

Ventajas de los i-nodos en un sistema de archivos FUSE:
- **Eficiencia en la gestión de archivos**: Acceso rápido a los metadatos al centralizar la información del archivo, sin tener que leer el contenido del archivo y gestión eficiente del espacio a través de los punteros

directos e indirectos, ajustándose dinámicamente a los tamaños de archivo.
- **Flexibilidad**: Soporte para múltiples tipos de archivos, dado que los i-nodos pueden representar archivos regulares, directorios, enlaces simbólicos, y otros tipos de archivos, proporcionando una estructura versátil. Además, los atributos extendidos permiten almacenar información adicional específica de la aplicación, mejorando la flexibilidad del sistema de archivos.
- **Resiliencia y recuperación**: Integridad de datos al separar los metadatos de los datos y el contador de enlaces ayuda a gestionar correctamente la eliminación de archivos solo cuando ya no hay enlaces activos.

No obstante, los i-nodos presentan algunas desventajas:
- **Complejidad de implementación**: Implementar y gestionar correctamente los i-nodos y sus punteros puede ser complejo y propenso a errores. Además, las operaciones que requieren la actualización de metadatos (como la modificación de permisos) pueden ser costosas en términos de tiempo de CPU y I/O.
- **Limitaciones de tamaño**: Los i-nodos tienen un tamaño fijo, lo que puede limitar la cantidad de información que se puede almacenar directamente con ellos, especialmente en sistemas de archivos con muchos atributos extendidos.
- **Uso de memoria**: Se requiere estrategias de caché eficientes para minimizar el impacto en rendimiento, lo cual añade una capa de complejidad adicional.

B. Almacenamiento de Metadatos del Sistema en una Estructura de Control de Bloques

El almacenamiento de metadatos del sistema en una estructura de control de bloques es una técnica crucial en el diseño de sistemas de archivos FUSE. Proporciona una forma organizada y eficiente de gestionar la información sobre archivos y directorios, facilitando el acceso rápido y la gestión efectiva del espacio. Sin embargo, también introduce complejidades y desafíos que deben ser abordados cuidadosamente para asegurar un rendimiento y fiabilidad óptimos del sistema de archivos.

Una estructura de control de bloques puede incluir varias **componentes claves**:
- **Superbloque**: Es el bloque que contiene la información global del sistema de archivos, como el tamaño total, el número de bloques y el punto de incio de las estructuras principales.
- **Tabla de i-nodos**: Una lista o matriz de i-nodos, donde almacena información sobre un archivo o directorio individual.
- **Mapa de bits de bloques**: Un mapa de bits que lleva un seguimiento de los bloques de datos utilizado y libres en el sistema de archivos.
- **Bloques de directorios**: Estructuras que contienen las entradas de los directorios, mapeando nombres de archivos a i-nodos.

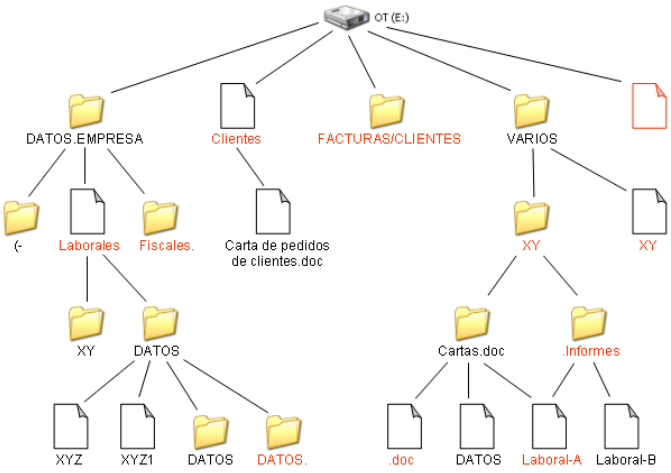
Ventajas de usar una estructura de control de bloques:
- **Eficiencia en la gestión de espacio**: Al tener mapas de bits para bloques i-nodos, se facilita la gestión y reutilización eficiente del espacio de almacenamiento.
- **Acceso rápido a metadatos**: Centraliza y organiza los metadatos, permitiendo un acceso rápido y eficiente.
- **Consistencia de datos**: Estructura bien definidas ayudan a mantener la consistencia de los datos y simplifican las operaciones de recuperación en caso de fallo.
- **Escalabilidad**: Permite que el sistema de archivos escale fácilmente a medida que se añaden más archivos y directorios.

Desventajas y Consideraciones:
- **Complejidad de implementación:** Requiere una implementación cuidadosa y detallada, lo cual puede ser complejo y propenso a errores.
- **Overhead de almacenamiento:** Las estructuras de control ocupan espacio adicional en el almacenamiento, lo que puede ser significativo en sistemas con muchos archivos pequeños.
- **Rendimiento:** Las operaciones intensivas en metadatos pueden ser costosas en términos de rendimiento, especialmente en sistemas con ata concurrencia de accesos.

C. Parámetros Relevantes del Sistema

D. Funciones de Inicialización del Árboles-B

Un árbol-B es una estructura de datos altamente eficiente y escalable, ideal para gestionar grandes volúmenes de datos en sistemas de archivos. En el contexto de FUSE, un árbol-B permite organizar los archivos y directorios de manera que las operaciones de búsqueda, inserción y eliminación sean rápidas y eficientes, mejorando significativamente el rendimiento del sistema de archivos.



Ejemplo de Árbol-B en un sistema de ficheros

Características:

- **Autobalanceo**. Los árboles-B se mantienen balanceados mediante operaciones que aseguran que todas las hojas están al mismo nivel, lo que garantiza un rendimiento consistente para las operaciones.
- **Nodos con múltiples hijos**. Cada nodo puede contener múltiples claves y punteros, lo que reduce la necesidad de acceder a los nodos superiores del árbol con frecuencia.
  En este caso, cada i-nodo tiene un ID, cuyo rango es [0-*MAX_FILES*], siendo *MAX_FILES* una macro cuyo valor es 100.
- **Altura baja.** Un árbol-B está diseñado para mantener una altura baja, lo que permite que las operaciones de búsqueda, inserción y eliminación sean rápidas, incluso para grandes volúmenes de datos.
- **Operaciones de rango.** Permiten operaciones eficientes sobre rangos de clave, realizar búsqueda de prefijos, y otras operaciones comunes en sistemas de ficheros.

VI. RESULTADOS Y FUNCIONALIDAD

A. *Resultados*

El controlador FUSE diseñado e implementado en este proyecto ha demostrado ser efectivo en términos de compatibilidad, eficiencia y rendimiento. Las funciones clave como '*getattr'*, '*readdir'*, '*open'* y '*read'* han sido desarrolladas y optimizadas para asegurar una gestión eficiente de recursos y un rendimiento robusto del sistema de archivos. Además, se han incorporado mecanismos de monitoreo y ajustes dinámicos que permiten mantener un desempeño óptimo bajo diferentes condiciones de uso.

B. *Evaluación de la Funcionalidad:*

- **Compatibilidad y Eficiencia:**

La implementación del controlador FUSE asegura compatibilidad con diversas aplicaciones que requieren operacions de sistema de archivos en espacio de usuario, proporcionando una interfaz consistente y eficiente para operaciones básicas y avanzadas.

- **Rendimiento:**

Se ha optimizado las estructuras de datos, como los i-nodos y la cache d metadatos, mejorando así el rendimiento global del sistema de archivos.

- **Gestión de Recursos:**

El diseño incorpora mecanismos para la gestión eficiente de recursos, como el espacio en disco y los descriptores de archivos, asegurando el uso óptimo del sistema.

- **Monitoreo y Ajustes Dinámicos:**

Se implementan herramientas para el monitoreo del rendimiento del sistema de archivos y ajustes dinámicos en las estrategias de gestión, lo cual permite mantener un rendimiento óptimo bajo diversas condiciones de carga.

VII. FUNCIONES IMPLEMENTADAS

A. *Funciones Implementadas*

i. ***getattr***

La función *getattr* se utiliza para obtener los atributos de un sistema de archivos FUSE. Los atributos incluyen información como permisos, tamaño, ID del usuario propietario, ID del grupo propietario y fechas de acceso y modificación.

Los pasos seguidos para el desarrollo de la función son los siguientes:

- Inicialización de la estructura *stbuf* con ceros.

Mediante la función *memset* se inicializa la estructura *stbuf*, estableciendo todos los bytes en cero, garantizando que no haya valores residuales en los campos.

Esta estructura se utiliza para almacenar los atributos del archivo o directorio que se devolverán al sistema de archivos FUSE.

- Obtención del i-nodo correspondiente al path.

A través de la llamada a la función *get_inode_by_path(path)*, se obtiene el i-nodo asociado al path especificado.

Si el i-nodo no existe, se devulve un error con el código *-ENOENT*.

- Asignación de los atributos del i-nodo al stbuf.
- Retorno de '0' para indicar éxito.

ii. ***readdir***

La función *readdir* es utilizada para leer las entradas de un directorio y llenar un búfer con los nombres de los archivos y subdirectorios dentro de se directorio.

Su funcionamiento es el siguiente:

- La función comienza con un contador *i* en 1 y declara un puntero a una estructura *s_fuseInode* llamada *inode*.
- Luego, obtiene el i-nodo correspondiente al path utilizando la función *get_inode_by_path(path).*
- Si no se encuentra el i-nodo, se devuelve un error -ENOENT.
- Se comprueban los **permisos** para poder leer el directorio.
- Si el i-nodo no tiene hijos (es decir, *inode->children* es *null*), también devuelve un error –ENOENT puesto que debe de tener al menos el hijo puntero a sí mismo y a su padre.
- A continuación, llena el búfer con las entradas de directorio, agregando las entradas "." Y ".." al búfer (representando el directorio actual y el directorio padre, respectivamente). Luego, recorre los hijos del i-nodo y agrega sus nombres al búfer.
- Finalmente, devuelve '0' para indicar que la función se ejecutó correctamente.

iii. ***open***

El propósito de esta función es abrir un archivo, especificado por ruta, en el sistema de archivos.

- En primer lugar, **se obtiene el i-nodo del archivo**. A través de la función *'get_inode_by_path(path)'* toma la ruta del archivo y devuelve un puntero a la estructura *'s_fuseInode'* correspondiente al i-nodo del archivo. Si el archivo no existe, devuelve *'NULL'*.
- Tras ello, **se verifica si el i-nodo es *'NULL'***. Si lo es, significa que no se encontró el archivo en la ruta especificada, por lo que la función devuelve *'-ENOENT'*, un error estándar que indica que el archivo o directorio no existe.
- Se comprueban los **permisos** para poder leer el archivo.
- **Se verifica si el i-nodo es un directorio**. En ese caso, se retorna *'-EISDIR'*, un código de error estándar que indica que el archivo es un directorio.
- Por último, **se asigna el identificador del archivo y se retorna éxito**.
  En primer lugar, busca el i-nodo correspondiente al archivo y si el i-nodo no se encuentra o si l archivo es un directorio, devuelve el error correspondiente. Si el archivo es válido, guarda el puntero al i-nodo en el campo *'fh'* de la estructura *'fuse_file_info'* y retorna '0' para indicar éxito.

iv. read

La función se utiliza para leer datos desde un archivo en el sistema de archivos FUSE. Su funcionamiento es el siguiente:

- **Obtención del Inodo.** Se llama a la función '*get_inode_by_path(path)*' para obtener el inodo correspondiente a la ruta.
- **Impresión de Información del Inodo.** Se imprime en *stdeer* el ID del inodo y los datos del archivo.
- **Verificación de la Existencia del Inodo.** Si *'get_inode_by_path'* no encuentra el inodo, retorna '*-ENOENT'*, indicando que el archivo no existe.
- Se comprueban los **permisos** para poder leer el archivo.
- **Comprobación del Directorio.** Si el inodo corresponde a un directorio (indicado por el modo '*S_IFDIR'*), devuelve '*-EISDIR'* porque no se puede leer un directorio como si fuera un archivo.
- **Revisión Desplazamiento Fuera de los Límites.** Si el desplazamiento es mayor o igual al tamaño del archivo ('*inode->size'*), no hay nada que leer, así que devuelve 0.
- **Ajuste Tamaño de Lectura.** Si la cantidad de bytes a leer más el desplazamiento excede el tamaño del archivo, se ajusta para que no se lea más allá del final del archivo.
- **Impresión Tamaño del Archivo.**
- **Copia Datos al Buffer.** Se copian los datos del archivo, empezando desde '*offset*' y hasta '*size'* bytes, al buffer '*buf'*.
- **Retorno del Tamaño Leído.** Finalmente, se retorna el número de bytes leídos, que será el valor ajustado del '*size'*.

En resumen, la función *read* se encarga de leer datos de un archivo en un sistema de archivos FUSE. Realiza verificaciones necesarias como la existencia del archivo, si es un directorio y si el desplazamiento es válido. Luego ajusta el tamaño de lectura si es necesario y copia los datos del archivo al buffer proporcionado. La función devuelve el número de bytes leídos o un error en caso de que ocurra algún problema.

v. create

Este fragmento de código implementa la lógica necesaria para crear un nuevo archivo en un sistema de ficheros basado en FUSE, manejando adecuadamente la asignación de memoria, la actualización de estructuras de datos y la validación de las condiciones de error.
Los pasos que sigue la función son los siguientes:

- **Obtener el i-nodo del directorio padre**, extrayendo la ruta del directorio padre usando *get_parent_path*, obteniendo el i-nodo del directorio padre usando *get_inode_by_path* y si el directorio padre no existe, devuelve *"-ENOENT"*.
- **Verificar el espacio disponible**. Se comprueba si hay i-nodos libres y si no los hay, devuelve "*-ENOSPC"*.
- **Asignar ID al nuevo i-nodo,** calculando el ID del nuevo i-nodo y actualizando el bitmap y, por último, decrementando el contador de i-nodos libres.
- **Crear y configurar el nuevo i-node**. Asigna memoria para el nuevo i-nodo, configura los atributos del i-nodo e inicializa listas de hijos y otros atributos del i-nodo.
- **Actualizar el i-nodo padre**, añadiendo un nuevo i-nodo a la lista del hijos del i-nodo padre y se reasigna memoria para la lisra de hijos del padre y se actualiza el contador de hijos.
- **Actualizar el *Control_Block***. Se inserta un nuevo i-nodo en el *control_block* y se actualiza el bitmap de i-nodos, se realiza la liberación de memoria asignada a *parent_path* y devuelve '0' indicando el éxito de la operación.

vi. mkdir

Esta función crea un nuevo directorio en el sistema de archivos, asigna un ID único al i-nodo del directorio y lo agrega como hijo al i-nodo padre y, actualiza los contadores y estructuras de datos necesarias para mantener la jerarquía de directorios.

Cabe destacar que **se obvian los metadatos en el size de los i-nodos** y, por tanto, **solo se indicará el tamaño de datos (bloques)**.

- La función comienza verificando si hay suficientes i-nodos disponibles en el sistema de archivos. Si no hay i-nodos libres, devuelve un error *-ENOSPC* (sin espacio disponible)
- Calcula el ID del nuevo i-nodo restando la cantidad de i-nodos librs al máximo de archivos (*MAX_FILES*).

- Busca un índice libre en el mapa de bits de i-nodos (*controlBlock->inode_bitmap*). Si encuentra uno, asigna el ID del nuevo i-nodo a ese índice.
- Actualiza el contador de i-nodos libres.
- Crea un nuevo i-nodo (*new_inode*) y asigna valores a sus campos.
- Actualiza el i-nodo padre (*parent_inode*).
- Por último, reserva memoria para el nuevo hijo en el i-nodo.

vii. rmdir

La función *rmdir* elimina un directorio en el sistema de archivos. Verifica si el directorio está vacío, actualiza las estructura de datos y libera la memoria asociada al i-nodo eliminado.

Los pasos realizados por la función son los siguientes:

- En primer lugar, se declara un i-nodo y se le asigna el resultado de buscar el nodo del directorio en el sistema de archivos basado en una ruta proporcionada. Si el i-nodo es *NULL*, significa que no se encontró el nodo del directorio y se devuelve un mensaje de error.
- Se comprueban los permisos para poder eliminar el directorio.
- Verificación de si el directorio está vacío. Si el número de hijos (archivos o subdirectorios) en l i-nodo es mayor que 2, la función devuelve -*ENOEMPTY*, indicando que el directorio no está vacío. Esto asegura que no se elimine un directorio que aún contenga archivos o subdirectorios.
- Obtención de un nodo padre.
- Actualización de la lista de hijos del nodo padre. Se busca el i-nodo en la lista de hijos del i-nodo padre. Si se encuentra, se reorganiza la lista de hijos para eliminar el i-nodo (moviendo los nodos siguientes hacia atrás) y se reduce el contador de hijos en el nodo padre.
- Liberación de recursos y actualización del *control_block*. Se libera la memoria ocupada por el i-nodo, se marca el bitmap de i-nodos para indicar que el i-nodo correspondiente está libre, se establece el puntero al i-nodo en l *control_block* como *NULL* para, finalmente, incrementar el contador de i-nodos libres (*free_inode_count*).
- En último lugar, se devuelve '0' para indicar que la eliminación del directorio fue exitosa.

viii. write

La función principal de esta función es escribir datos en un archivo en el sistema de archivos FUSE. Para ello, realiza verificaciones de permisos (se comprueba si el usuario tiene permisos de escritura sobre el archivo), existencia del inodo (se verifica que el inodo no sea '*NULL'*, lo que indicaría que el archivo no existe), tipo de archivo (se verifica que no sea un directorio, ya que no se puede escribir en él) y tamaño antes de copiar los datos al archivo (se comprueba que la escritura no exceda el tamaño máximo permitido para un archivo).

Al igual que la función anterior, se declara una variable *inode*, de tipo *struct s_fuseInode**, se le asigna el resultado de la búsqueda del nodo del archivo o directorio de la ruta proporcionada y se realiza un ajuste del tamaño del archivo y asignación de bloques, de la siguiente manera:

- Si el desplazamiento más el tamaño de escritura supera el tamaño actual del archivo: se verifica si hay suficientes bloques libres en el sistema de archivos para acomodar el nuevo tamaño, se calcula el número de bloques necesarios para el nuevo tamaño, se redimensiona el campo *data* del i-nodo para acomodar los nuevos bloques, se actualiza el tamaño del archivo y el contador de bloques, se reduce el contador de bloques libres en el sistema de archivos y, por último, se actualiza el bitmap de bloques para marcar los bloques asignados al archivo.

Finalmente, se copian los datos desde el búfer al archivo representado por el i-nodo, comenzando desde el desplazamiento y con el tamaño especificado, se actualiza la marca de tiempo de modificación del inodo y se ajusta el tamaño del directorio padre. Finalmente, se devuelve el tamaño de los datos que se han escrito.

ix. utimes

Esta función se utiliza para establecer los tiempos de acceso y modificación de un archivo o directorio en el sistema de archivos FUSE.

x. rename

La función de esta función es renombrar un archivo o un directorio en un sistema de archivos FUSE, actualizando los i-nodos y las estructuras de directorios involucradas.

Los pasos seguidos son los siguientes:

- Obtención del i-nodo del archivo o directorio correspondiente a la ruta de origen y si no se encuentra se devuelve un mensaje de error.
- Obtención del i-nodo del directorio padre en la ruta de destino.
- Creación de una copia de la ruta de destino.
- Extracción del nombre del archivo o directorio de destino, buscando la última aparición del carácter '/' en la ruta de destino y señalando el puntero *name* al nombre del destino.
- Actualización del i-nodo con el nuevo nombre y ruta.
- Agregación del i-nodo al directorio padre de destino, realizando *realloc* para agregar el i-nodo al array de hijos del directorio padre e incrementando su contador.
- Eliminación del i-nodo del directorio padre de origen, buscando el i-nodo correspondiente en el array de hijos, eliminándolo y ajustando el contador de hijos.
- Se decrementa el contador de hijos del directorio padre de origen, el tamaño (size) y e lnúmero de links y se devuelve '0' para indicar éxito.

xi. setxattr (Falta)

Esta función se encarga de establecer un atributo extendido para un archivo especificado por una ruta.

Primero, busca el i-nodo correspondiente al archivo, Si el i-nodo no se encuentra, devuelve un error indicando que

el archivo o directorio no existe. Si el nombre del atributo es '*user.comment'*, libera cualquier comentario existente y asigna el nuevo valor, retornando éxito. Finalmente, si el nombre del atributo no es '*user.comment*', devuelve un error indicando que la operación no está soportada.

xii. unlink

Esta función se encarga de eliminar un archivo en un sistema de archivos FUSE, realizando las siguientes acciones:

- **Obtención del Inodo del Archivo a Eliminar.** Se llama la función '*get_inode_by_path(path)*' para obtener el inodo correspondiente a la ruta. Si el inodo no existe, se devuelve '*-ENOENT'*, indicando que el archivo no existe.
- **Verificación de la Existencia del Archivo.**
- **Adquisición del Inodo del Directorio Padre.** Se obtiene el inodo del directorio padre del archivo a eliminar.
- **Eliminación del Inodo del Archivo de la Lista de Hijos del Directorio Padre.** Se busca el inodo del archivo a eliminar en la lista de hijos del directorio padre. Si se encuentra, se elimina de la lista desplazando los elementos siguientes una posición hacia atrás y luego se ajusta el tamaño de la lista con '*realloc'*.
- **Actualización de los Contadores Correspondientes en el Directorio Padre y en el Bloque de Control.** Se decrementa el contador de enlaces ('*links')* y el contador de hijos ('*children_count')* del directorio padre. Además, se incrementa el contador global de inodos libres *('free_inode_count')* en el bloque de control del sistema de archivos ('*controlBlock'*)., así como se reduce el tamaño del inodo padre en consecuencia con el borrado del hijo.
- **Liberación de la Memoria del Inodo y de los Datos del Archivo.** Si el archivo tiene datos asociados ('*inode->data'*), se libera la memoria de esos datos. Finalmente se libera la memoria del inodo.

xiii. statfs

Esta función devuelve una serie de valores estadísticos del sistema de ficheros extraídos de las macros definidas en el sistema o del *controlBlock* a través de la escritura en un ***struct statvfs *stbuf***.

xiv. chmod

Esta función encuentra el i-nodo al que se le quiere cambiar los permisos con *get_inode_by_path(path).* Si no es encuentra (inode == NULL) se devuelve –ENOENT. Si no se actualiza el campo mode del i-nodo con el mode pasado por parámetros y se devuelve finalmente 0 para indicar que la ejecución fue correcta.

VIII. CONCLUSIONES OBTENIDAS

La implementación del driver para el sistema de archivos con FUSE ha demostrado ser una tarea desafiante en diversos aspectos, pero sumamente educativa. A través del desarrollo y evaluación de este proyecto, se han obtenido las siguientes conclusiones:

1. **Flexibilidad y Funcionalidad FUSE:** FUSE ha demostrado ser una herramienta extremadamente flexible que permite la creación de sistemas de archivos en especio de usuario con relativa facilidad. Las funciones básicas implementadas (*'getattr', 'readdir', 'open', 'read', 'write'*) han demostrado ser eficaces para gestionar archivos y directorios, proporcionando un comportamiento consistente y esperable en las operaciones de entrada y salida.
2. **Optimización y Rendimiento:** La correcta gestión de los inodos y metadatos es crucial para el rendimiento del sistema de archivos. La implementación de técnicas de caché para metadatos y optimización de los algoritmos de búsqueda han mejorado significativamente la velocidad de las operaciones. Sin embargo, las operaciones intensivas en metadatos pueden ser costosas en términos de rendimiento, especialmente en sistemas con alta concurrencia de accesos.
3. **Desafíos en la Implementación:** La implementación detallada y cuidadosa requerida para desarrollar un sistema de archivos FUSE es compleja y propensa a errores. El manejo correcto de las estructuras de control y la memoria es esencial para evitar fugas de memoria y asegurar la integridad del sistema de archivos.
4. **Aplicaciones Prácticas:** La versatilidad de FUSE permite su aplicación en diversas áreas, desde el acceso remoto a archivos en servidores hasta la comunicación con dispositivos mediante el Protocolo de Transferencia de Medios. Este proyecto ha subrayado como FUSE puede transformar la interacción con los sistemas de archivos, proporcionando una plataforma robusta para múltiples aplicaciones.

En resumen, el proyecto ha proporcionado una compresión profunda de las capacidades y limitaciones de FUSE, destacando su potencial para la creación de sistema de archivos personalizados y eficientes en el espacio de usuario.

IX. VALORACIÓN DE EXPERIENCIA Y PROCESOS DE APRENDIZAJE REALIZADOS

El desarrollo de este driver ha supuesto un desafío interesante en el que el desarrollo y testeo del funcionamiento del mismo ha sido francamente muy distinto a lo habitual.

Lo más valorable de este proyecto ha sido el aprendizaje necesario para poder testear y sacar fallos del sistema de ficheros para poder encontrar las soluciones oportunas para un correcto funcionamiento de las funciones implementadas.

También cabe resaltar la reflexión previa al desarrollo para modular un diseño correcto y eficiente de estructuras y mecanismos necesarios para almacenar los datos y la estructura del árbol de i-nodos.

X. MEJORAS O TRABAJOS FUTUROS

La persistencia del sistema con la escritura en archivos ha generado problemas a los que no le hemos encontrado solución. El almacenamiento de datos en los **.bin** funciona cuando se ejecuta el binario con la "*flag*" de debugger *–d*. Sin embargo si se ejecuta sin esta" flag" el mismo código genera errores de escritura en los ficheros.

Funciones a implementar futuras:

- Las funciones relacionadas con los enlaces duros y simbólicos (**readlink, symlink, link).**
- Las funciones para guardar y leer en y de los archivos binarios que contienen las estructuras del montaje, los datos almacenados y el controlBlock para asegurar la persistencia del sistema y la compatibilidad con el proyecto del otro grupo.
- Se utilizan las funciones *save_inode, save_controlBlock* para guardar los datos y *load_inodes, load_controlBlock* para inicializar el punto de montaje. Las funciones que guardan datos están implementadas pero falta el testeo para comprobar si se puede optimizar el diseño de guardado y si es óptimo para la carga de datos. Además hay que valorar si separar el archivo que almacena el controlBlock de otro archivo para los datos es una buena estrategia.

XI. LÍNEAS DE TRABAJO NO IMPLEMENTADAS

La persistencia para el desarrollo de un sistema que se puede montar y desmontar incluso en otros proyectos fuse para que sea compatible y versátil no está del todo acabado, falta por arreglar el problema mencionado antes, solo funciona cuando se ejecuta el binario en modo debugger.

Por lo tanto falta por terminar el apartado de compatibilidad con el otro grupo, y también un trabajo opcional para poder crear y eliminar enlaces duros y simbólicos.

XII. REFERENCIAS

[1] Pfeiffer, J. J., Jr., Ph.D. (2018, Feb. 4). Writing a FUSE Filesystem: a Tutorial. [Online]. Disponible: New Mexico State University: cs.nmsu.edu/~pfeiffer/fuse-tutorial/

[2] CORE. (Fecha de acceso 2024, Junio 14). Diseño e Implementación de un Sistema de Ficheros Simple con FUSE. [Online]. Disponible: core.ac.uk/download/pdf/288499367.pdf

[3] Harvey Mudd College. (2011). CS135 FUSE Documentation. [Online]. Disponible: www.cs.hmc.edu/~geoff/classes/hmc.cs135.201109/homework/fuse/fuse_doc.html

[4] Ask Ubuntu. (2022, May 19). How to safely install fuse on Ubuntu 22.04? [Online]. Disponible: askubuntu.com/questions/1409496

[5] LinuxQuestions.org. (2009, Noviembre 16). [SOLVED] FUSE filesystem SETATTR function not implemented. [Online]. Disponible: linuxquestions.org/questions/programming-9/fuse-filesystem-setattr-function-not-implemented-769391/

[6] GitHub. (n.d.). FUSE: C-based FUSE for macOS SDK - Example Directory. [Online]. Disponible: github.com/osxfuse/fuse/tree/master/example

[7] GitHub. Issue 296. (2018, Aug. 27). Error encountered when using gcsfuse: *fuseops.GetXattrOp not implemented. [Online]. Available: github.com/GoogleCloudPlatform/gcsfuse/issues/296

[8] Stack Overflow. (2013, Jun 7). Which fuse version in my kernel? [Online]. Disponible: stackoverflow.com/questions/16991414

[9] Stack Overflow. (2008, Nov 18). Why is fuse not using the class supplied in file_class? [Online]. Disponible: stackoverflow.com/questions/300047

[10] S. Venkateswaran, "Essential Linux Device Drivers," 1ª ed. Upper Saddle River, NJ, EE. UU.: Prentice Hall, 2008

[11] J. Corbet, A. Rubini, G. Kroah-Hartman, "Linux Device Drivers," 3ª ed. Sebastopol, CA, EE. UU.: O'Reilly Media, Inc., 2005

[12] R. Love, "Linux Kernel Development," en Linux Kernel Development, 3ª ed. Upper Saddle River, NJ, USA: Pearson Education, 2010

[13] D. P. Bovet, M. Cesati, "Understanding the Linux Kernel," en Understanding the Linux Kernel, 3ª ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005