Universidad Nacional Autónoma de México

Facultad de Ingeniería

PROYECTO 02. (Micro) sistema de archivos multihilos

Alumno: Martínez García José Eduardo (320304679)

Profesor: Dr. Gunnar Eyal Wolf Iszaevich

Sistemas Operativos

Grupo teoría: 08

Semestre 2026-1

Fecha de entrega: 20/11/25

Objetivos

PROYECTO "(Micro) sistema de archivos multihilos"

El objetivo del proyecto es desarrollar un sistema de archivos simulado, llamado FiUnamFS, que permita realizar operaciones básicas de gestión de archivos desarrollado bajo las especificaciones académicas de las indicaciones propuestas en https://github.com/unamfi/sistop-2026-1/tree/main/proyectos/2

Descripción General del proyecto

Este sistema simula un dispositivo de almacenamiento en un archivo de longitud fija (fiunamfs.img) y permite realizar operaciones fundamentales de gestión de archivos.

El desarrollo se enfocó en dos pilares clave de los sistemas operativos:

1. **Sistemas de Archivos:** Interpretación y manipulación de estructuras de datos en disco a nivel de byte (Superbloque y Directorio).
2. **Administración de Procesos:** Implementación de concurrencia mediante hilos (*threads*) y mecanismos de sincronización para garantizar la integridad de los datos durante operaciones simultáneas.

Adicionalmente, se integró una interfaz compatible con **FUSE** (Filesystem in Userspace), permitiendo que el sistema operativo Linux reconozca y monte el archivo .img como una unidad de almacenamiento real.

Funcionalidades Implementadas

El programa cumple con los requisitos funcionales especificados, operando sobre la versión 26-1 de FiUnamFS:

1. **Listar contenidos del directorio:**
    - Lee y decodifica las entradas del directorio (clusters 1 a 4).

- Muestra información detallada: Nombre, Tamaño, Cluster Inicial, Fecha de Creación y Fecha de Modificación.
- Filtra correctamente las entradas vacías o eliminadas.

2. **Copiar archivos (FiUnamFS → Sistema Local):**
    - Extrae el contenido binario de un archivo almacenado en el disco simulado.
    - Lo escribe en una ruta especificada del sistema operativo anfitrión.
    - Esta operación se ejecuta en un hilo secundario para no congelar la interfaz.
3. **Copiar archivos (Sistema Local → FiUnamFS):**
    - Lee un archivo externo y busca espacio disponible en el directorio y en el área de datos (asignación contigua).
    - Escribe la información y actualiza los metadatos (nombre, tamaño, fechas) en el directorio.
4. **Eliminar archivos:**
    - Localiza el archivo por nombre y marca su entrada en el directorio como "vacía" (reemplazando el tipo de archivo con el carácter - y el nombre con puntos .....), liberando lógicamente el espacio.
5. **Interfaz Dual (GUI y FUSE):**
    - **Modo Gráfico:** Interfaz amigable desarrollada con Tkinter para usuarios finales.
    - **Modo Montaje:** Implementación de la clase FiUnamFS_FUSE que hereda de fuse.Operations, permitiendo montar el sistema de archivos en Linux y usar comandos de terminal (ls, cp, rm) directamente sobre él.

Entorno y dependencias

1. Lenguaje de programación y entorno

El proyecto fue desarrollado utilizando **Python3**. Se requiere una versión **3.6 o superior**, ya que el código hace uso de características modernas como *f-strings* y manejo avanzado de excepciones. El desarrollo se centró en la compatibilidad multiplataforma, permitiendo la

ejecución de la interfaz gráfica en Windows/Linux y la funcionalidad de montaje FUSE específicamente en entornos Linux.

2. **Bibliotecas**
    - tkinter: Para el diseño y control de la interfaz gráfica de usuario.
    - threading: Para la ejecución de hilos concurrentes y la sincronización de procesos mediante locks.
    - struct: Para la manipulación de datos binarios y conversión de formatos (Little Endian) requeridos por el sistema de archivos.
    - fusepy: Para la integración con el kernel de Linux, permitiendo el montaje del sistema de archivos en el espacio de usuario.
    - os / sys: Para la interacción con el sistema operativo anfitrión y el manejo de argumentos de entrada.
    - datetime: Para el registro y formato de las fechas de creación y modificación de los archivos.

Arquitectura y diseño de codigo

El código se estructura en una arquitectura modular para separar la lógica de bajo nivel de la interfaz de usuario.

1. Clase FiUnamFS_Driver (El Motor)

Es el núcleo del programa. Maneja todas las operaciones de entrada/salida a nivel de bytes sobre el archivo fiunamfs.img.

- **Manejo de Estructuras:** Utiliza la librería struct para desempaquetar valores en formato *Little Endian* (<I), respetando estrictamente los offsets de la especificación 2026 (donde el Cluster inicial precede al Tamaño).
- **Sincronización:** Implementa un objeto threading.Lock() ("el candado"). Cada vez que se va a leer o escribir en el archivo .img, se adquiere este candado. Esto garantiza que si un hilo está escribiendo un archivo, otro hilo no pueda intentar leer el directorio simultáneamente, evitando condiciones de carrera y corrupción de datos.

2. Clase FiUnamFS_FUSE (El Puente)

Esta clase actúa como un traductor entre las llamadas al sistema de Linux y los métodos de nuestro Driver. Implementa métodos estándar de POSIX como getattr, readdir, read, write, create y unlink, redirigiendo estas peticiones a la lógica interna de FiUnamFS.

3. Interfaz Gráfica y Concurrencia

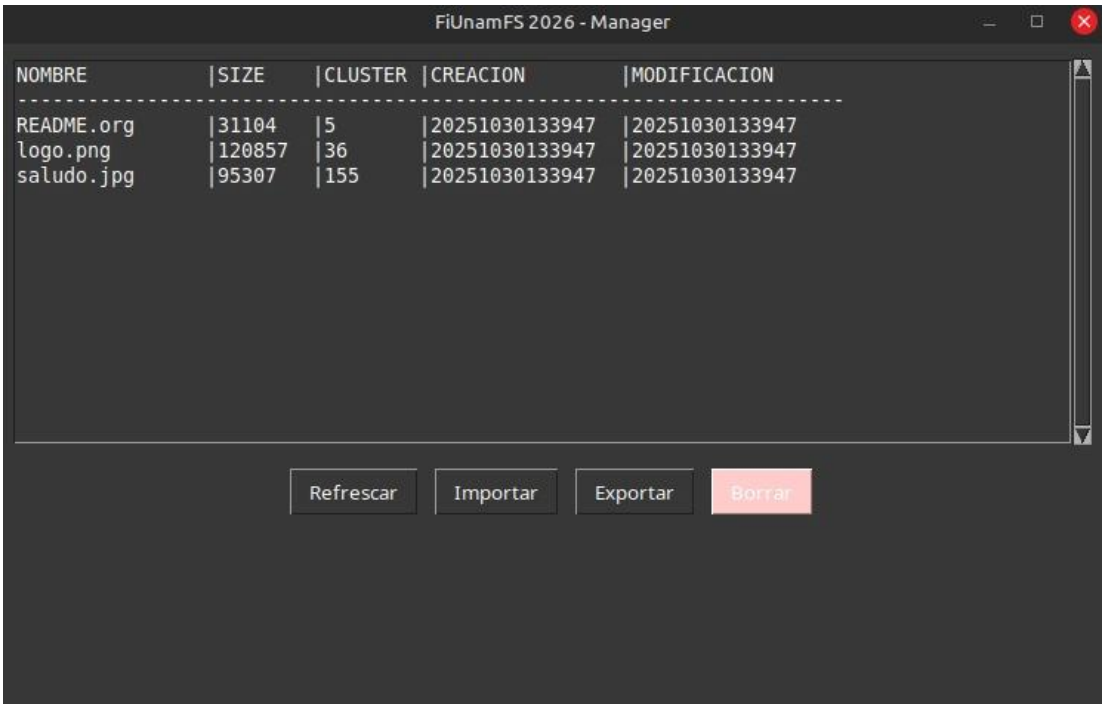
La interfaz visual (iniciar_gui) gestiona la interacción con el usuario. Para cumplir con el requisito de multihilo:

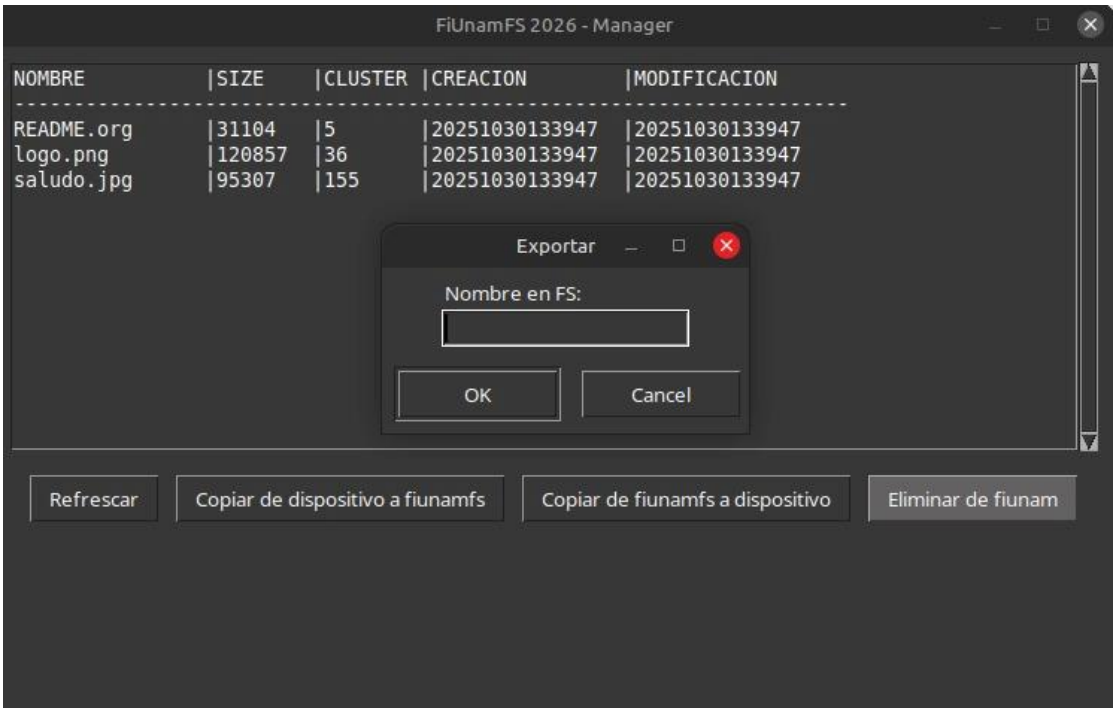
- **Hilo Principal:** Mantiene la ventana responsiva y escucha eventos.
- **Hilos Trabajadores (*Workers*):** Las operaciones pesadas, como copiar archivos grandes (copy_out), se delegan a nuevos hilos (threading.Thread). Esto permite que la barra de título se pueda mover y la interfaz no se quede en "No responde" mientras se transfieren datos.

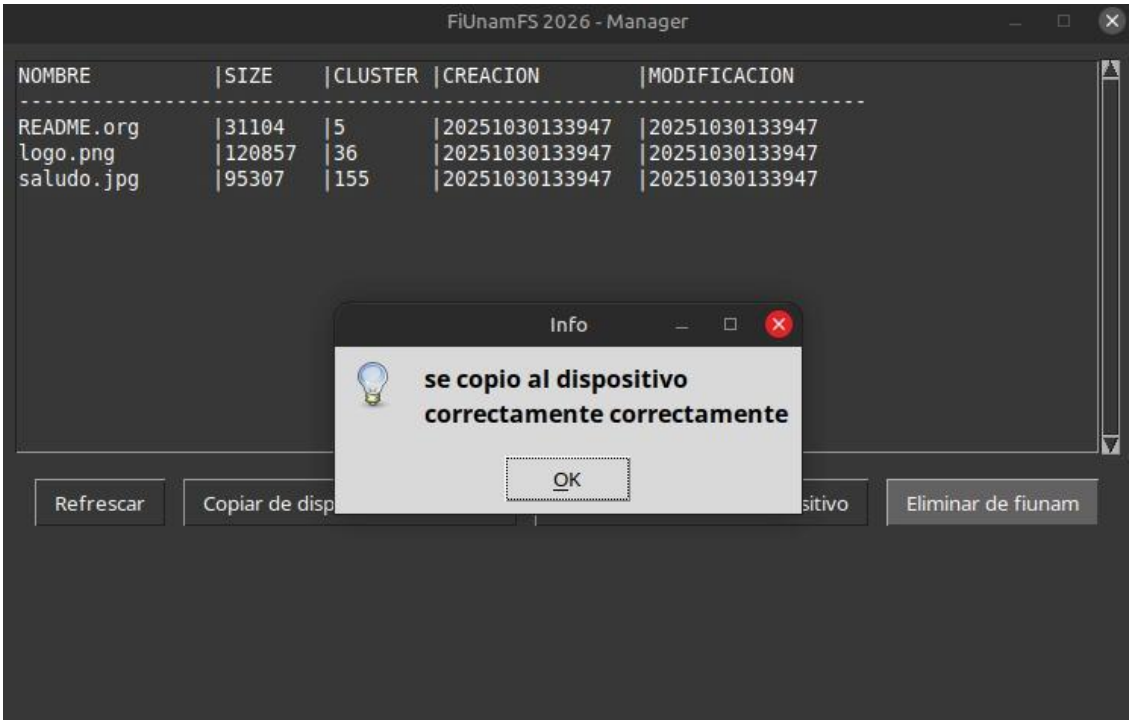
Intrucciones

1. Verificar Archivo .img: Antes de comenzar, asegúrate de que el sistema de archivos fiunamfs.img esté listo y sea válido dentro de la carpeta donde se encuentra “Proyecto2”.
2. Verifica el correcto listado de Archivos: Observa que los archivos se encuentran descritos en la interfaz gráfica con el nombre, tamaño, fecha de creación y fecha de última modificación de cada archivo.
3. Copiar a dispositivo desde fiunamfs: Para extraer un archivo, selecciona "Copiar a dispositivo desde fiunamfs". Ingresa el nombre exacto del archivo con todo y extensión que deseas obtener de fiunamfs y selecciona la carpeta de destino en tu computadora.
4. Copiar a fiunamfs desde el dispositivo: Para guardar un archivo nuevo en el disco simulado, usa "Copiar a fiunamfs desde dispositivo". Se abrirá un explorador para que selecciones el archivo local que deseas transferir.
5. Eliminar Archivo de fiunamfs: Si deseas borrar un archivo, haz clic en "Eliminar Archivo" e ingresa el nombre correspondiente. El sistema marcará el espacio como disponible.
6. Cerrar Programa: Para finalizar la ejecución de forma segura, utiliza el botón "x" ubicado en la parte superior derecha del recuadro o ok en su defec
7.

Pruebas


FiUnamFS 2026 - Manager
NOMBRE |SIZE |CLUSTER |CREACION |MODIFICACION
README.org |31104 |5 |20251030133947 |20251030133947
logo.png |120857 |36 |20251030133947 |20251030133947
saludo.jpg |95307 |155 |20251030133947 |20251030133947
Refrescar
Importar
Exportar
Borrar

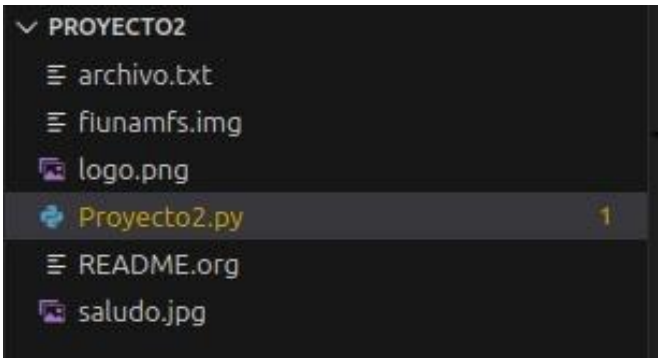

FiUnamFS 2026 - Manager
NOMBRE |SIZE |CLUSTER |CREACION |MODIFICACION
README.org |31104 |5 |20251030133947 |20251030133947
logo.png |120857 |36 |20251030133947 |20251030133947
saludo.jpg |95307 |155 |20251030133947 |20251030133947
Exportar
Nombre en FS:
OK
Cancel
Refrescar
Copiar de dispositivo a fiunamfs
Copiar de fiunamfs a dispositivo
Eliminar de fiunam

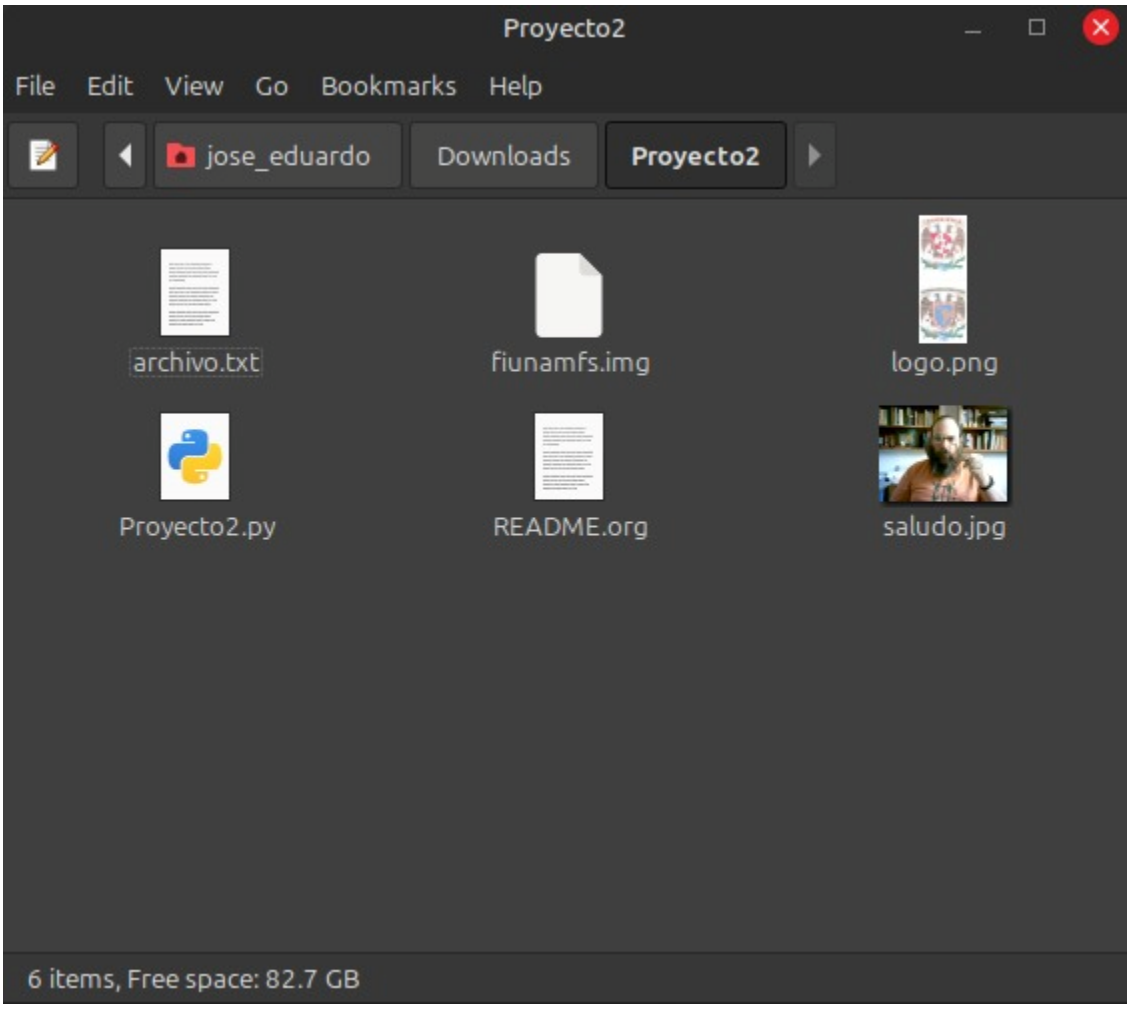


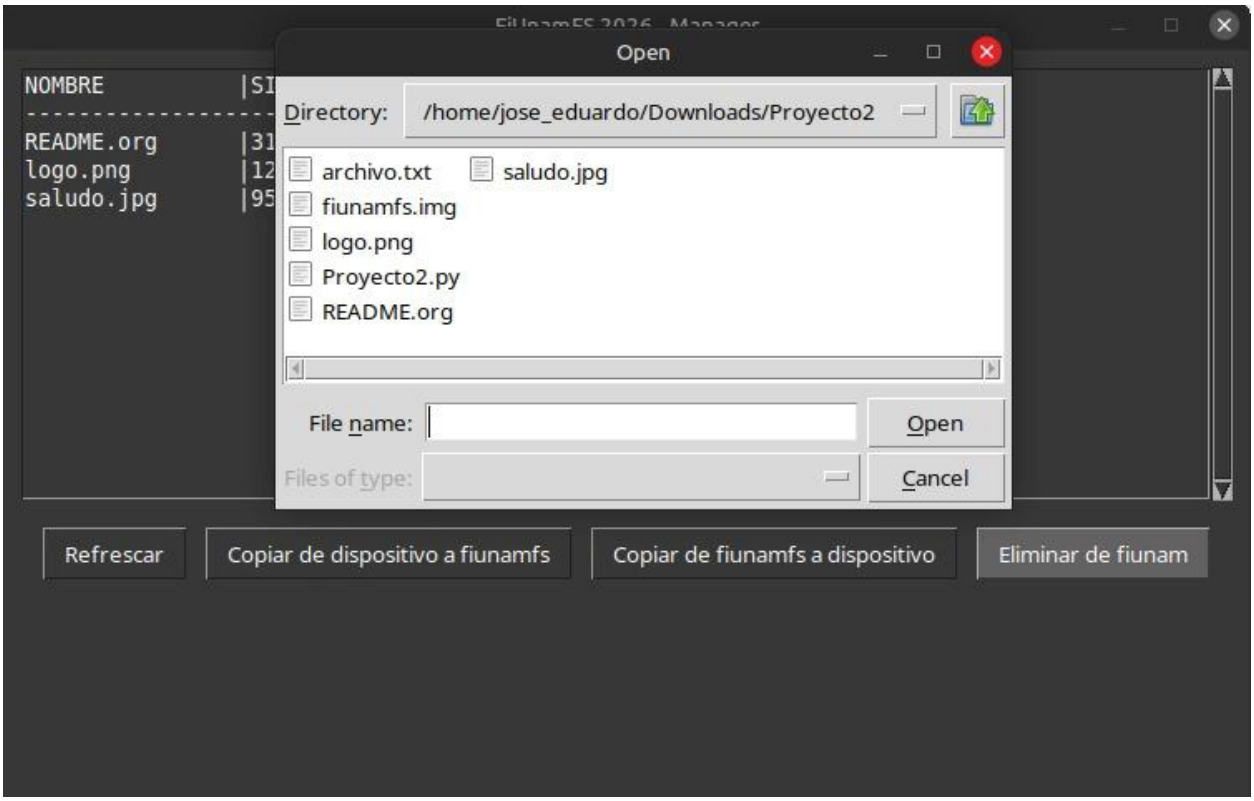
Archivos que se encontraban en fiunamfs:

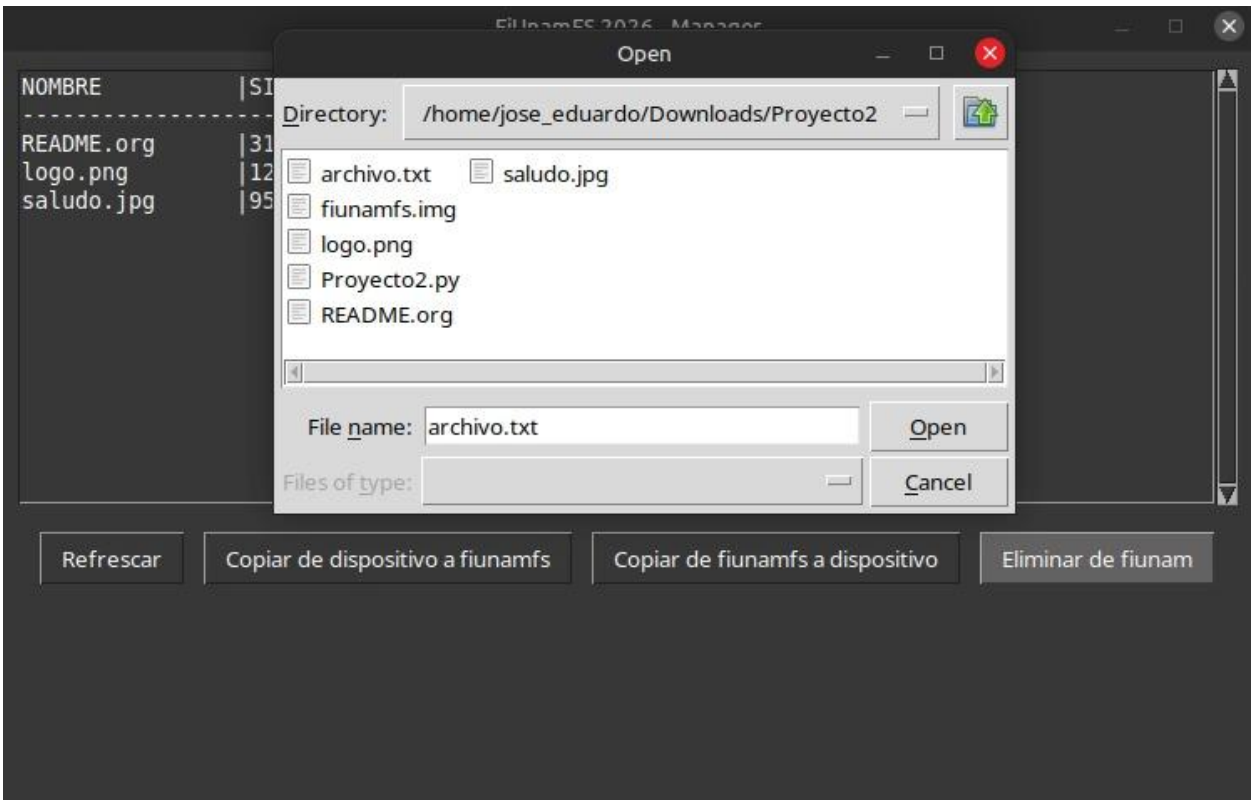
Muy estiloso profe!!!!

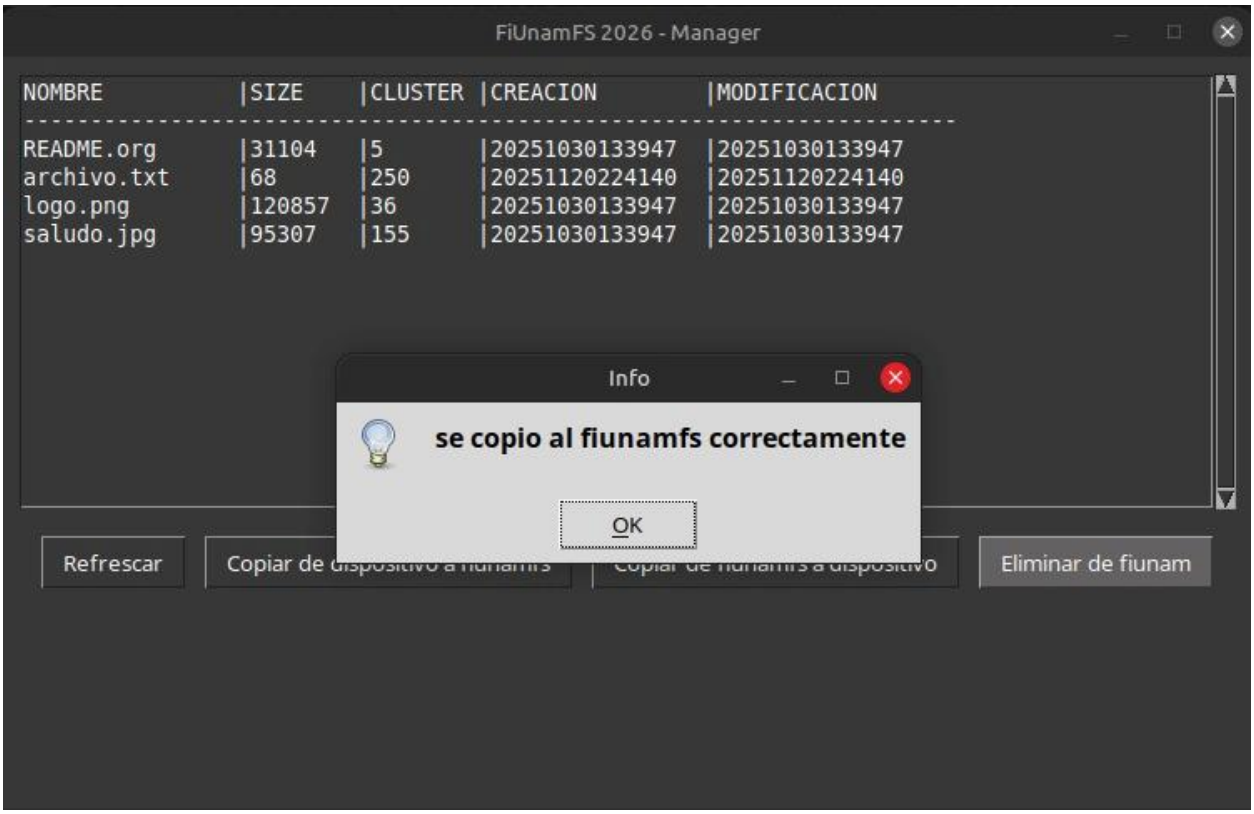
Prueba de archivos

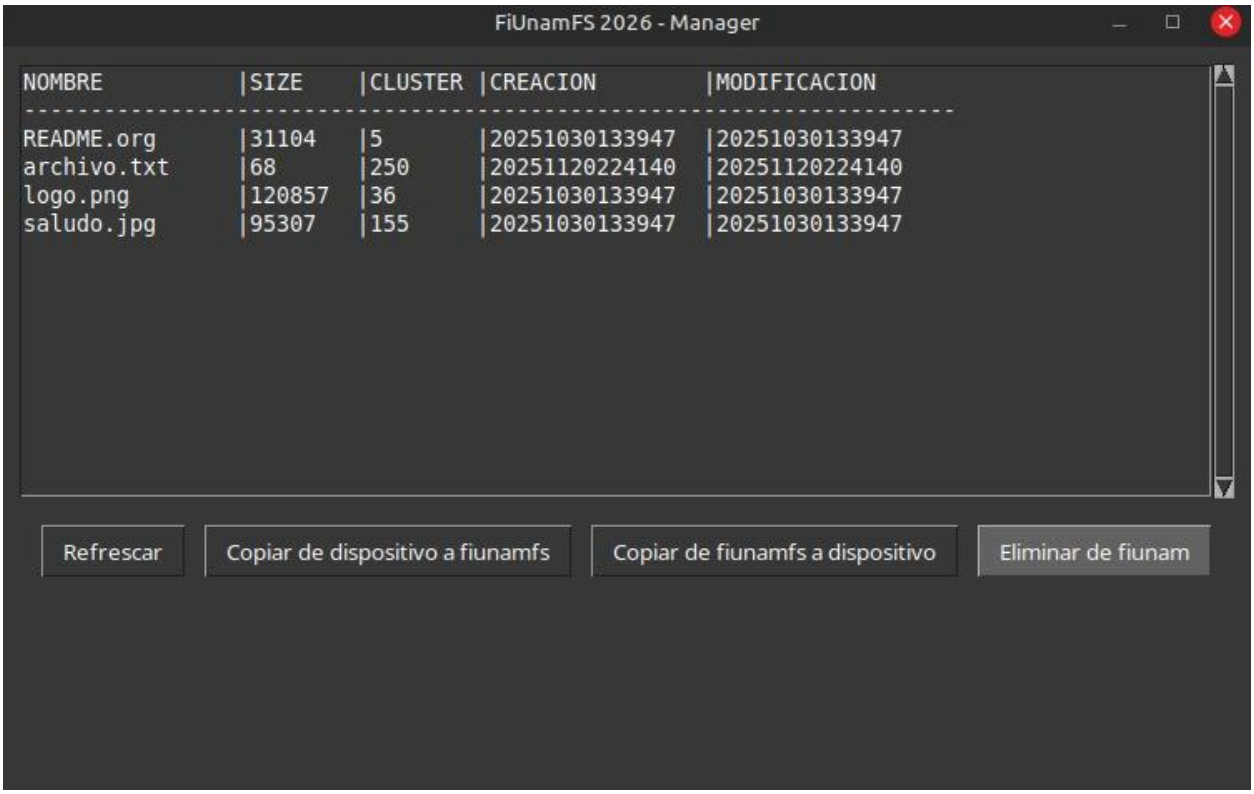


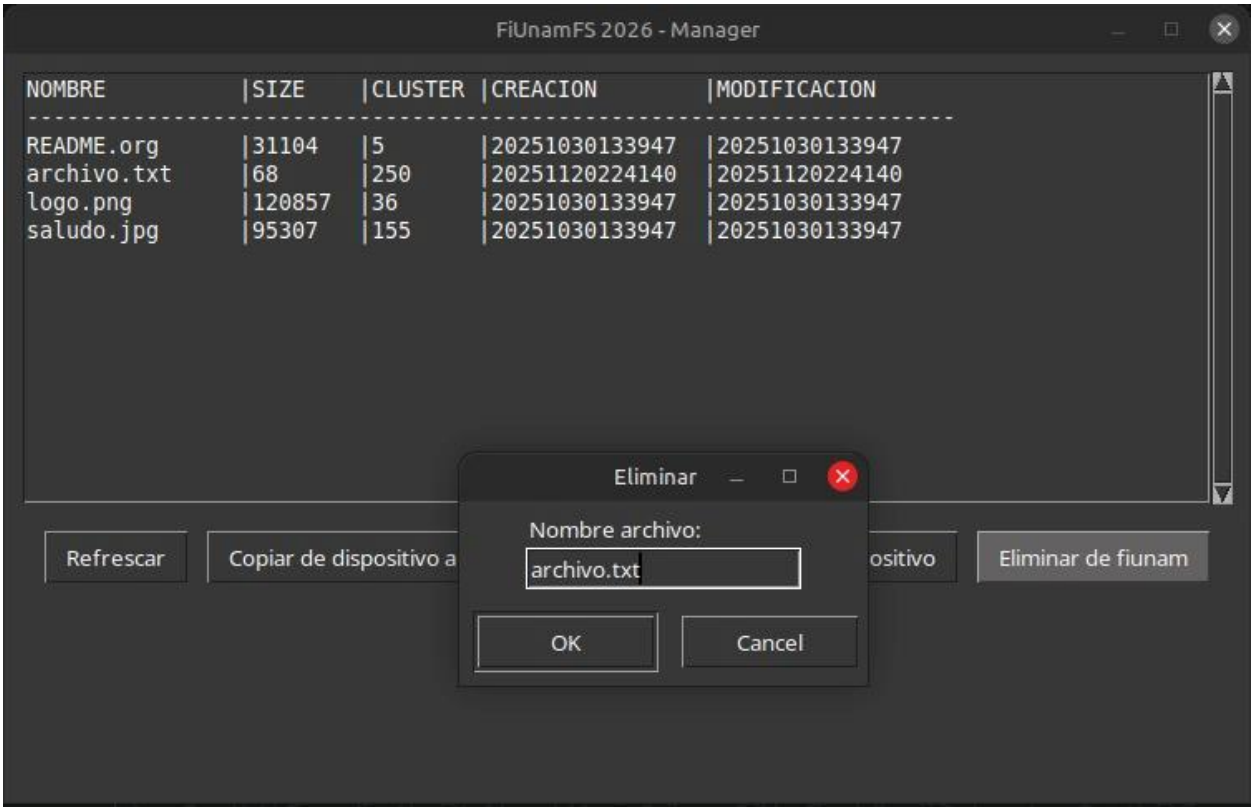



NOMBRE |SI
README.org |31
logo.png |12
saludo.jpg |95
Open
Directory: /home/jose_eduardo/Downloads/Proyecto2
archivo.txt
saludo.jpg
fiunamfs.img
logo.png
Proyecto2.py
README.org
File name:
Open
Files of type:
Cancel
Refrescar
Copiar de dispositivo a fiunamfs
Copiar de fiunamfs a dispositivo
Eliminar de fiunam


NOMBRE |SI
README.org |31
logo.png |12
saludo.jpg |95
Open
Directory: /home/jose_eduardo/Downloads/Proyecto2
archivo.txt
saludo.jpg
fiunamfs.img
logo.png
Proyecto2.py
README.org
File name: archivo.txt
Open
Files of type:
Cancel
Refrescar
Copiar de dispositivo a fiunamfs
Copiar de fiunamfs a dispositivo
Eliminar de fiunam


FiUnamFS 2026 - Manager
NOMBRE |SIZE |CLUSTER |CREACION |MODIFICACION
README.org |31104 |5 |20251030133947 |20251030133947
archivo.txt |68 |250 |20251120224140 |20251120224140
logo.png |120857 |36 |20251030133947 |20251030133947
saludo.jpg |95307 |155 |20251030133947 |20251030133947
Info
se copio al fiunamfs correctamente
OK
Refrescar
Eliminar de fiunam


FiUnamFS 2026 - Manager
NOMBRE |SIZE |CLUSTER |CREACION |MODIFICACION
README.org |31104 |5 |20251030133947 |20251030133947
archivo.txt |68 |250 |20251120224140 |20251120224140
logo.png |120857 |36 |20251030133947 |20251030133947
saludo.jpg |95307 |155 |20251030133947 |20251030133947
Refrescar
Copiar de dispositivo a fiunamfs
Copiar de fiunamfs a dispositivo
Eliminar de fiunam



conclusión

El desarrollo de este proyecto me permitió aterrizar conceptos teóricos que a menudo parecen abstractos en clase. Al implementar el driver para FiUnamFS, comprendí la importancia crítica de la precisión en el manejo de bytes y offsets; un solo byte mal calculado puede significar leer basura o corromper todo el sistema de archivos.

La implementación de la concurrencia fue particularmente reveladora. Entendí que "hacer varias cosas a la vez" conlleva la responsabilidad de proteger los recursos compartidos. El uso del Lock no es solo un requisito, sino una necesidad vital para evitar que el sistema colapse cuando dos procesos intentan modificar el disco al mismo tiempo.

Además, la integración con FUSE fue un reto interesante que elevó el nivel del proyecto, pasando de un simple "lector de archivos" a un sistema que se integra realmente con el sistema operativo anfitrión. En resumen, este proyecto unificó la gestión de memoria secundaria con la administración de procesos, mostrándome cómo interactúan estas capas

para que, como usuarios, podamos simplemente "copiar y pegar" sin preocuparnos por lo que ocurre detrás.

REFERENCIAS

- Python Software Foundation. (2024). threading — Thread-based parallelism. Recuperado de docs.python.org.
- Wolf, G. (2024). fuse_in_python_guide. GitLab. Recuperado de gitlab.com/gunnarwolf.
- Documentación de la biblioteca struct de Python para manejo de datos binarios.
- Apuntes/diapositivas del curso Sistemas Operativos, Facultad de Ingeniería, UNAM.