Universidad Nacional Autónoma de México

Proyecto 02 - (Micro) sistema de archivos multihilos

Curso: Sistemas Operativos

Profesor: Dr. Gunnar Eyal Wolf Iszaveich

Autor: Chávez López B. Alejandro

Cuenta: 421112601

Grupo: 08

November 21, 2025

1 Descripción del Proyecto

El objetivo de esta práctica fue crear un programa capaz de manipular el sistema de archivos *FiUnamFS*. La idea principal es simular el funcionamiento de un disquete de 1.44 MB, interactuando directamente con los bytes del archivo `fiunamfs.img` sin depender del sistema operativo.

El proyecto consistió en implementar funciones para listar, extraer y eliminar archivos. Además, para la escritura de nuevos archivos, se cumplió con el requisito de usar programación concurrente (hilos). Finalmente, para mejorar la experiencia de usuario, se desarrolló una interfaz gráfica (GUI) que permite realizar todas las operaciones de manera visual.

2 Herramientas Utilizadas

Para el desarrollo utilicé el lenguaje **Python**. La elección se debió a que es un lenguaje con el que ya he desarrollado proyectos anteriores y me resulta familiar.

Se utilizaron las siguientes librerías (nativas):

- `struct`: Para manejar datos binarios y conversiones *Little Endian*.
- `threading` y `queue`: Para la implementación de hilos y sincronización.
- `tkinter`: Para el desarrollo de la interfaz gráfica de usuario, elegida por venir incluida en la instalación estándar de Python.

3 Detalles de la Implementación

3.1 Lectura de Datos

Dado que el sistema utiliza el formato *Little Endian*, fue necesario utilizar funciones de empaquetado y desempaquetado de bytes constantemente. Me basé en las especificaciones del documento para definir el tamaño del cluster en 1024 bytes y las entradas de directorio en 64 bytes.

3.2 Escritura y Espacio Libre

Para guardar archivos, implementé una lógica de asignación contigua. El programa revisa todo el directorio para mapear qué clusters están ocupados y busca el primer "hueco" continuo donde quepa el archivo que se quiere copiar. Si no hay espacio continuo suficiente, la operación se cancela.

3.3 Uso de Hilos

Para cumplir con la parte de concurrencia, utilicé el modelo de Productor-Consumidor. Un hilo se encarga de leer el archivo de mi computadora y poner los datos en una cola, mientras que el segundo hilo saca esos datos y los escribe en la imagen del disco. Esto permite que la lectura y escritura ocurran de forma sincronizada.

3.4 Interfaz Gráfica

Se implementó una interfaz visual utilizando `tkinter` que reutiliza la lógica del backend. La interfaz presenta una tabla con los contenidos del disco y botones para importar, extraer y eliminar archivos. Apliqué los colores que uso normalmente en mis reportes (catppuccin).

3.5 Pruebas de Funcionamiento

A continuación se muestra la interfaz gráfica en funcionamiento, visualizando los archivos contenidos en la imagen de disco:

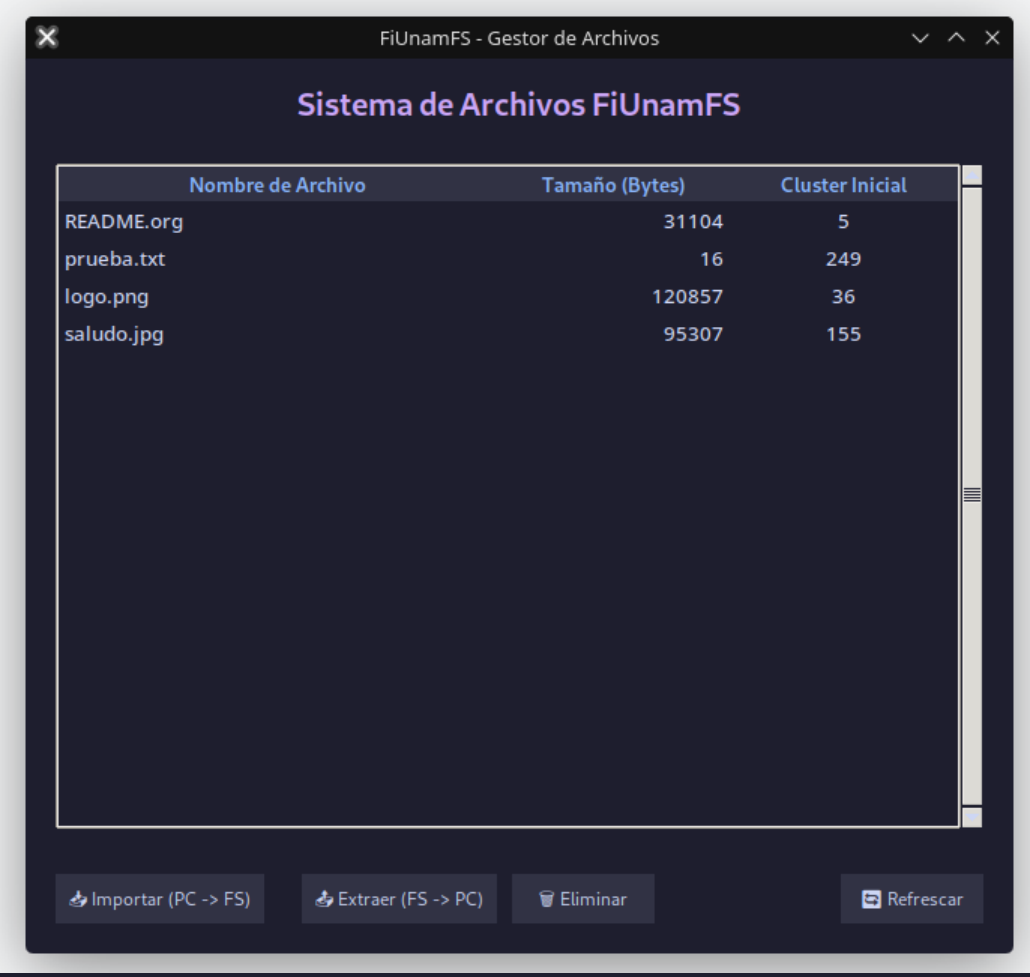

Figure 1: Interfaz Gráfica mostrando archivos

4 Problemas Encontrados y Soluciones

Durante el desarrollo surgieron varios detalles técnicos que requirieron ajustes en el código:

- **Versión del Sistema:** Al principio, el validador fallaba porque la documentación indicaba la versión "26-2", pero el archivo `.img` descargado tenía la versión "26-1". Tuve que ajustar la validación para aceptar ambas.

- **Limpieza de Nombres:** Tuve problemas al comparar los nombres de los archivos porque el método `strip()` no eliminaba los caracteres nulos (`0x00`) que el sistema usa como relleno. Esto se solucionó limpiando primero los nulos y después los espacios.
- **Archivos Duplicados:** Noté que el programa permitía guardar el mismo archivo varias veces, creando entradas repetidas. Para corregirlo, agregué una verificación al inicio de la función de copiado que revisa si el nombre ya existe antes de empezar a escribir.

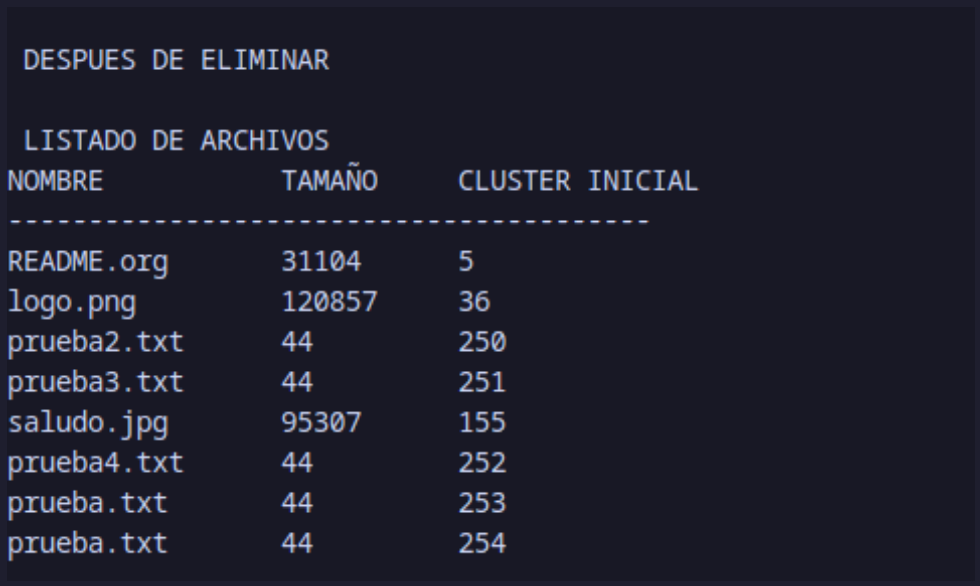

Figure 2: Captura de pantalla mostrando el error de duplicados (antes de la corrección)

- **Longitud del Nombre:** Implementé una restricción de 15 caracteres para los nombres. Si se permitían nombres más largos, se sobrescribía el byte donde inicia el cluster, corrompiendo el archivo.

5 Conclusiones

Realizar este proyecto me permitió comprender mejor cómo se estructuran los datos dentro de un sistema de archivos a bajo nivel. Aunque la lógica general de copiar y borrar parece sencilla, manipular directamente los bytes requiere mucho cuidado con los offsets y los tipos de datos.

La separación de la lógica del sistema (backend) de la interfaz de usuario permitió integrar la GUI de manera sencilla. El resultado final es una herramienta robusta que cumple con los requisitos de concurrencia y ofrece una experiencia de usuario amigable y visualmente agradable.