UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Materia: Sistemas Operativos

Profesor: Gunnar Wolf

Semestre 2026-1

Alumnos:

Coronado Perez Diego - No. Cuenta: 320252460

Luna Quintero Diego Alejandro - No. Cuenta: 320225888

Proyecto 2: **(Micro) sistema de archivos multihilos**

Entrega: 20/11/2025

1. Descripción del Proyecto

El proyecto consiste en la implementación completa de un sistema capaz de **leer, manipular y administrar el micro-sistema de archivos FiUnamFS**, siguiendo estrictamente la especificación proporcionada en el curso de Sistemas Operativos.

El programa desarrollado permite:

1. Listar los contenidos del directorio de FiUnamFS
2. Copiar archivos del FiUnamFS a la computadora del usuario
3. Copiar archivos locales hacia el FiUnamFS
4. Eliminar archivos del sistema de archivos

Este programa incluye:

✔ Manejo de superbloque

✔ Lectura y manejo del directorio

✔ Escritura contigua

✔ Sincronización con mutex

✔ Interfaz gráfica completa

✔ Validación de nombre y versión para evitar corrupción

✔ Uso de animaciones y actualización en vivo

1.1 Especificaciones de FiUnamFS

El sistema cumple estrictamente con las especificaciones del volumen:

- El sistema se almacena en un archivo de **1440 KB**, simulando un disquete.
- Todas las cadenas de texto son **ASCII 8-bit**.
- Todos los enteros dentro del sistema se representan con **32 bits en formato little endian** (`<I`).

- La superficie está dividida en **sectores de 512 bytes** y clusters de **1024 bytes**.
- No tiene particiones; el archivo completo es el volumen.
- El sistema maneja un **directorio plano**, sin subdirectorios.
- El cluster **#0 contiene el superbloque** con:
    - Bytes 0–8: Nombre del sistema (`FiUnamFS`)
    - Bytes 10–14: Versión (26-1)
    - Bytes 20–35: Etiqueta del volumen
    - Bytes 40–44: Tamaño del cluster
    - Bytes 45–49: Número de clusters del directorio
    - Bytes 50–54: Total de clusters de la unidad
- Los clusters del directorio son del 1 al 4.
- Cada entrada del directorio mide **64 bytes**, con nombre, tamaño, tipo, cluster inicial y fechas.

El sistema implementado respeta exactamente estas reglas, validando nombre, versión y estructura.

2. Lenguaje y entorno de desarrollo

El proyecto fue desarrollado en:

- **Python 3.11**
- Entorno: **Visual Studio Code (VS Code)**
- Sistema operativo principal: **Windows 11**
- Extensión usada: **Python (Microsoft)**

Bibliotecas utilizadas (todas estándar):

- `os` – Manejo de archivos y rutas
- `struct` – Lectura y escritura en formato little endian
- `threading` – Manejo de hilos
- `datetime` – Manejo de fechas
- `tkinter` – Interfaz gráfica
- `ttk` – Widgets avanzados

No se requiere ninguna instalación adicional en Windows.
En Linux, debe instalarse Tkinter (en caso de no tenerlo): sudo apt install python3-tk

3. Estrategia empleada

El proyecto se dividió en **tres componentes principales**:

1. Manejo interno del sistema de archivos
2. Operaciones del directorio
3. Interfaz gráfica + concurrencia

3.1 Parámetros del proyecto

Los parámetros implementados corresponden al estándar:

- Tamaño del disquete: **1440 KB**
- Tamaño de cluster: **1024 bytes**
- Tamaño de entrada de directorio: **64 bytes**
- Directorio ubicado en clusters 1–4

- Superbloque en cluster 0
- Sistema de asignación **contigua**

Estas constantes se encuentran definidas al inicio del código, facilitando su modificación y lectura.

3.2 Sincronización de operaciones concurrentes

Para evitar conflictos cuando múltiples hilos leen o modifican el archivo `.img`, se implementó:

archivo_mutex = threading.Lock()

Protege operaciones críticas como:

- Escritura de entradas del directorio
- Escritura en clusters de datos
- Lectura del superbloque

✔ Un evento para comunicación entre hilos:

Usado en:

- Animación de “Procesando…”
- Finalización de la operación real

✔ Uso de dos hilos por operación:

- Hilo 1: Realiza la tarea (copiar, listar, eliminar)
- Hilo 2: Controla animación

Esto garantiza una interfaz fluida y evita congelamientos.

3.3 Clase `SistemaArchivosFiUnamFS`

Esta clase administra todo el sistema de archivos.
Principales funciones:

verificar_existencia_imagen()

Comprueba que el archivo `.img` existe.

leer_superbloque()

- Valida nombre: `FiUnamFS`
- Válida versión: `26-1`
- Lee:
    - tamaño de cluster
    - número de clusters del directorio
    - número total de clusters
- Usa `struct.unpack('<I')` para little endian

leer_entradas_directorio()

- Recorre los clusters 1–4
- Lee cada entrada de 64 bytes
- Filtra entradas vacías
- Construye un diccionario con los datos del archivo

_obtener_mapa_clusters()

Genera un mapa booleano de clusters ocupados para asignación contigua.

3.4 Clase `OperacionesFS`

Contiene la lógica central del proyecto.

listar()

Lee el superbloque, devuelve la lista de entradas.

copiar_a_pc()

- Busca el archivo en el directorio
- Ubica su cluster inicial
- Exporta los datos al sistema real

copiar_a_fiunam()

Implementa **asignación contigua**:

1. Calcula el tamaño en clusters
2. Genera mapa de clusters ocupados
3. Aplica **First-Fit** para encontrar espacio
4. Escribe datos en clusters contiguos
5. Agrega entrada al directorio

eliminar()

- Marca byte 0 como `'-'`
- Rellena nombre con `. . . . . .`
- Limpia metadatos

3.5 Interfaz gráfica (Tkinter)

La GUI proporciona:

- Selección de archivo `.img`
- Visualización del superbloque
- Listado de archivos en `TreeView`
- Botones para todas las operaciones
- Etiqueta de estado con actualización dinámica

Widgets utilizados:

- `ttk.LabelFrame`
- `ttk.Button`
- `ttk.Label`
- `ttk.Treeview`
- `filedialog`

3.6 Manejo de hilos y animaciones

La interfaz usa dos hilos por operación:

✔ Hilo de trabajo

Ejecuta:

- Copiar
- Eliminar
- Importar
- Listar

✔ Hilo de animación

Muestra en pantalla:

Procesando... |

Procesando... /

Procesando... -

Procesando... \

✔ Comunicación

Se logra mediante `threading.Event()`:

- Cuando el trabajo termina → se detiene la animación
- Se permite iniciar nuevas operaciones

3.7 Características adicionales

- Interfaz gráfica clara y fácil de usar
- Animaciones no bloqueantes
- Validación estricta del FS
- Mensajes de error en GUI
- Actualización automática del directorio tras cada operación
- Nombres ASCII limitados a 14 caracteres

4. Descripción de la sincronización empleada

4.1 Sincronización de operaciones

Se utiliza:

- **1 mutex global (`archivo_mutex`)**
  Evita carreras de datos en operaciones de lectura/escritura.

4.2 Clases relacionadas con hilos

Dentro de `FiUnamFS_GUI`:

- `_animacion_ui()` → hilo que actualiza el estado
- `_hilo_trabajo_wrapper()` → hilo que ejecuta operaciones

4.3 Verificación del sistema de archivos

Cada operación inicia leyendo el superbloque.
Si no es válido:

- se arroja error
- se evita corrupción

5. Cómo se usa y ejemplos de ejecución

Para ejecutar el proyecto basta con abrir el archivo **`proyecto2.py`** desde su IDE favorito (como VS Code, PyCharm o Thonny) o directamente desde la terminal con:

```
python proyecto2.py
```

Ahora bien, surge una pregunta importante:
¿Dónde debe estar el archivo `fiunamfs.img` para que el programa funcione?

Existen **dos formas válidas** de usar el sistema de archivos:

✔ Opción 1: Guardar `fiunamfs.img` en la misma carpeta que `proyecto2.py`

Éste es el método más sencillo.
Si el archivo de imagen se encuentra en la misma carpeta donde se ejecuta el proyecto:

```
/MiProyecto/
   proyecto2.py
   fiunamfs.img
```

Entonces el programa lo detectará **automáticamente**, cargará el sistema de archivos en cuanto se inicie y podrá utilizarse sin configuraciones adicionales.

Con esta opción, usted simplemente ejecuta el programa y puede inmediatamente:

- Listar archivos
- Copiar archivos
- Importar archivos
- Eliminar archivos

✔ Opción 2: Guardar `fiunamfs.img` en otra ubicación

Si prefiere mantener el sistema de archivos guardado en otra carpeta —por organización, seguridad o comodidad **no hay ningún problema**.

El programa incluye una opción integrada que permite:

- Buscar el archivo `fiunamfs.img` en cualquier lugar del sistema
- Cargarlo manualmente desde la interfaz gráfica
- Continuar trabajando normalmente con él

Esta función hace uso del cuadro de selección de archivos (`filedialog.askopenfilename`), lo que permite elegir la imagen desde cualquier ruta del equipo.

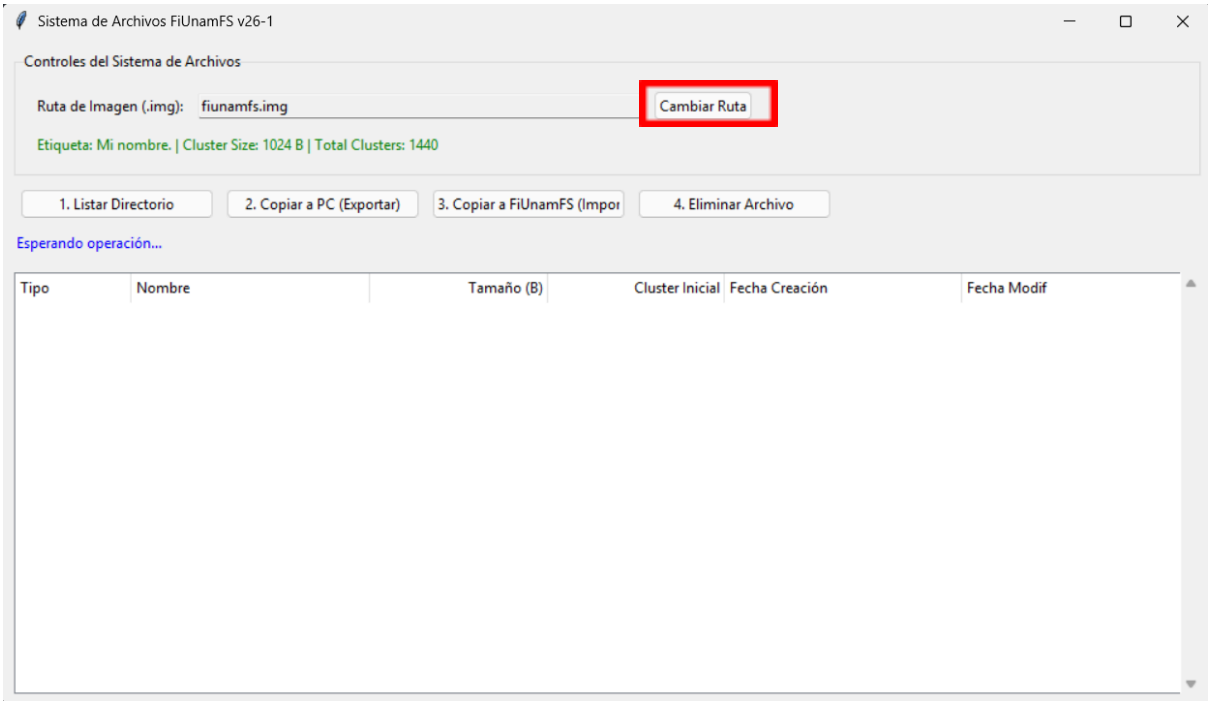
Resultado final

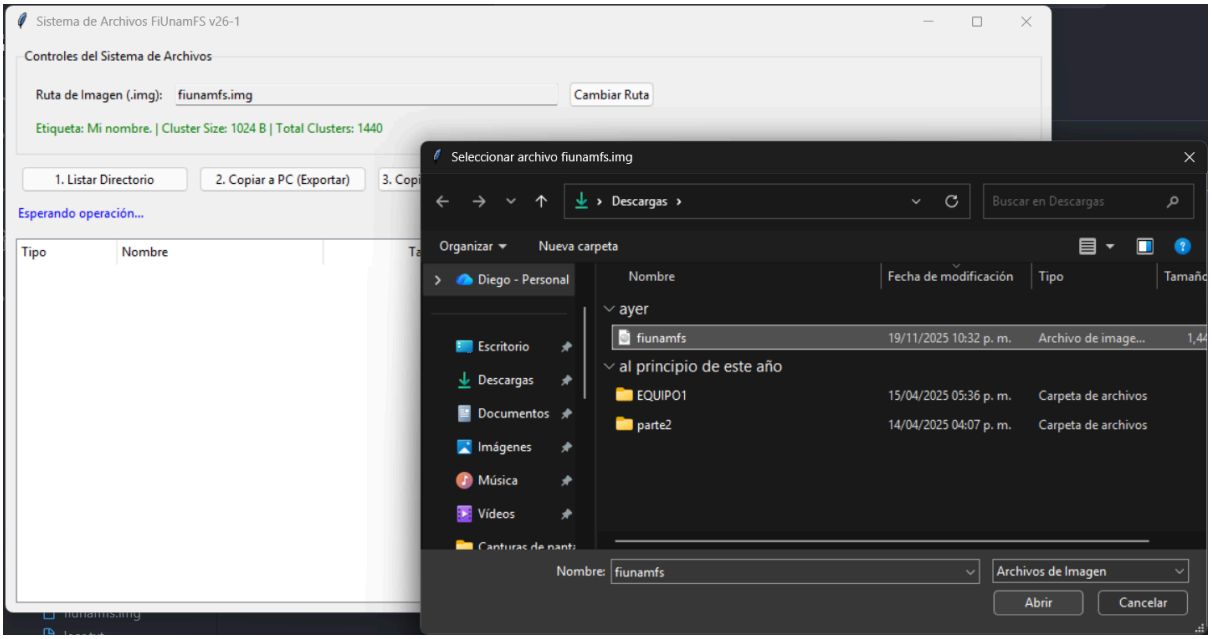
De esta manera, puede trabajar con FiUnamFS:

- **Automáticamente**, si el archivo está junto al ejecutable
- **Manual y libremente**, si se encuentra en otra carpeta

Bien ahora pasamos con un par de ejecuciones para las evidencias

Esta es la pantalla inicial y como se mencionó anteriormente puede buscar la ruta del archivo fiunamfs con el botón “Cambiar Ruta” y si el archivo fiunamfs esta dentro de la misma carpeta de ejecución del proyecto pues abrira solo y basta con ejecutar las opciones disponibles

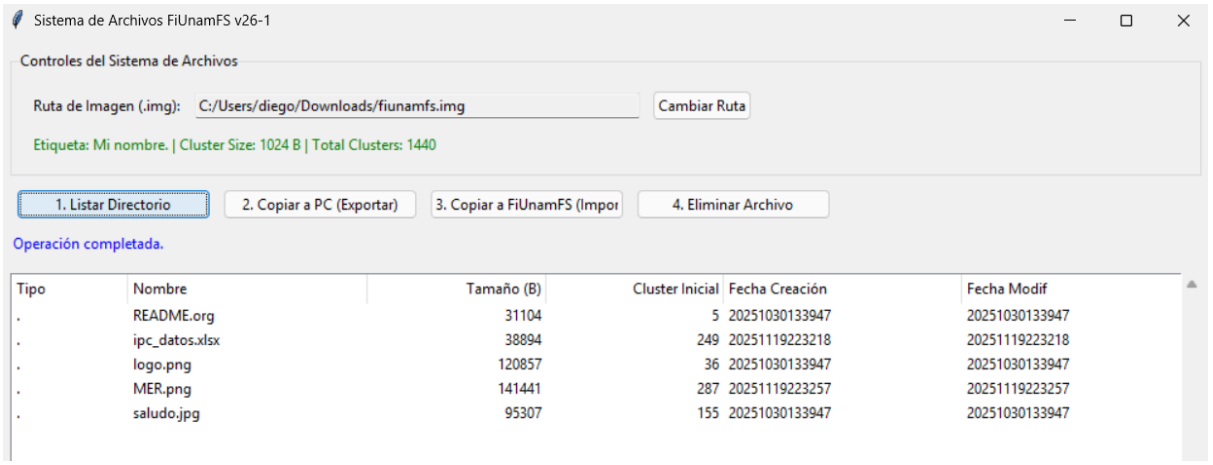




Buscamos el archivo fiunamfs donde lo hayamos guardado y abrimos ahora tocara probar la funcionalidad de cada opción

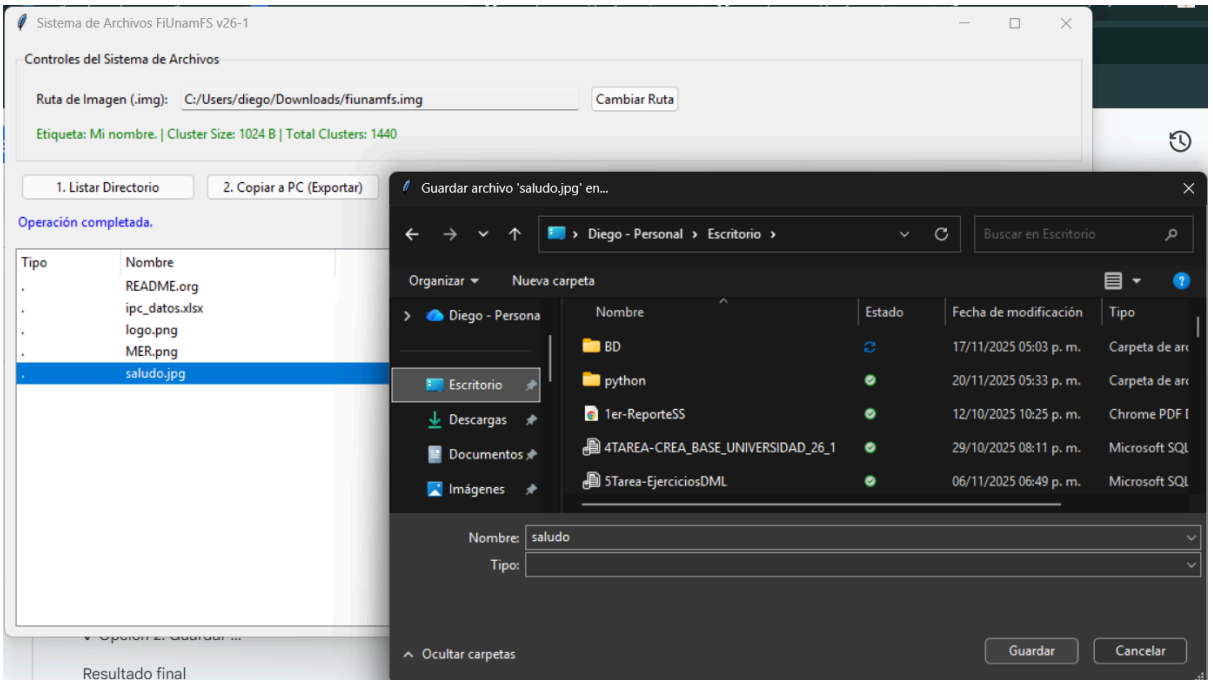
5.1 Listado del directorio

Presionar “Listar Directorio” para ver los archivos almacenados.

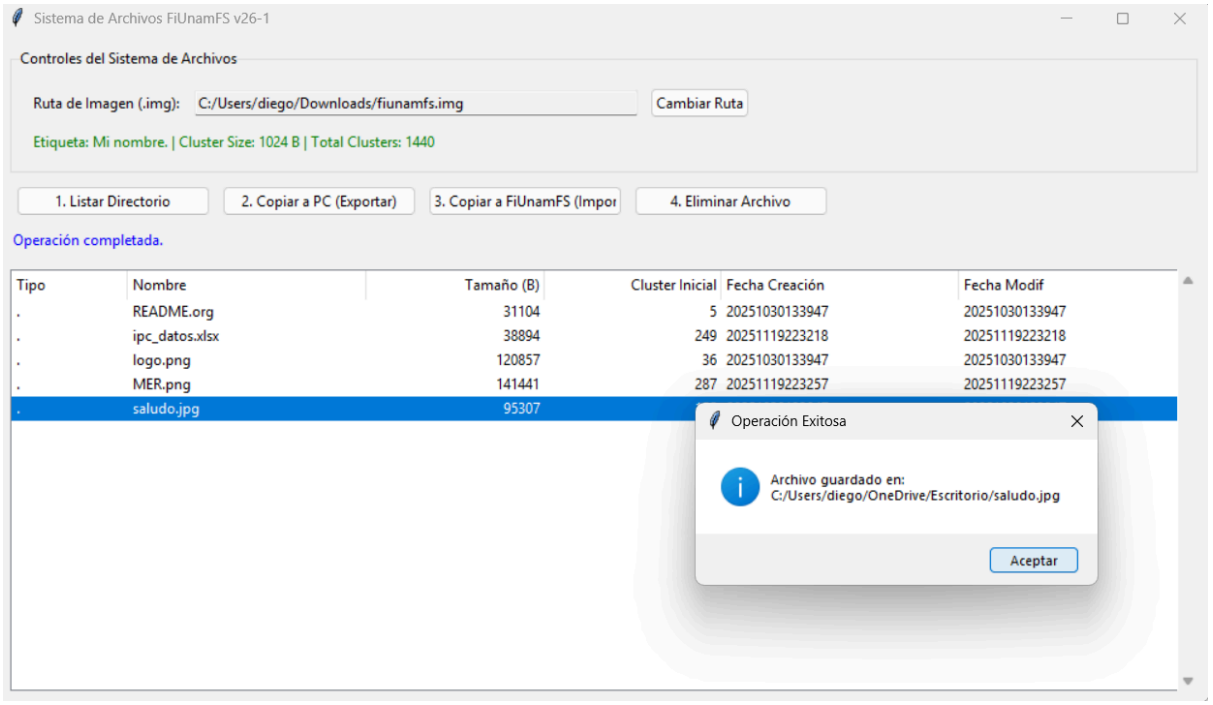


5.2 Copiado desde FiUnamFS

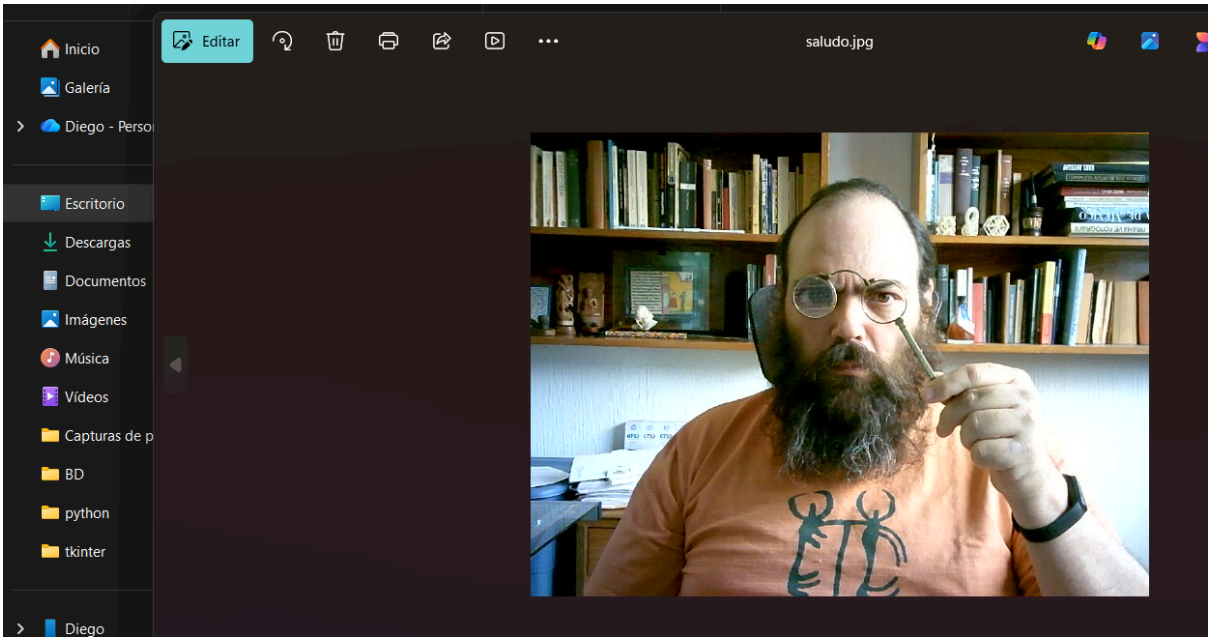
Seleccionar un archivo → “Copiar a PC” → elegir destino.



Una vez seleccionada la ruta destino damos clic en guardar e inmediatamente nos muestra el siguiente mensaje

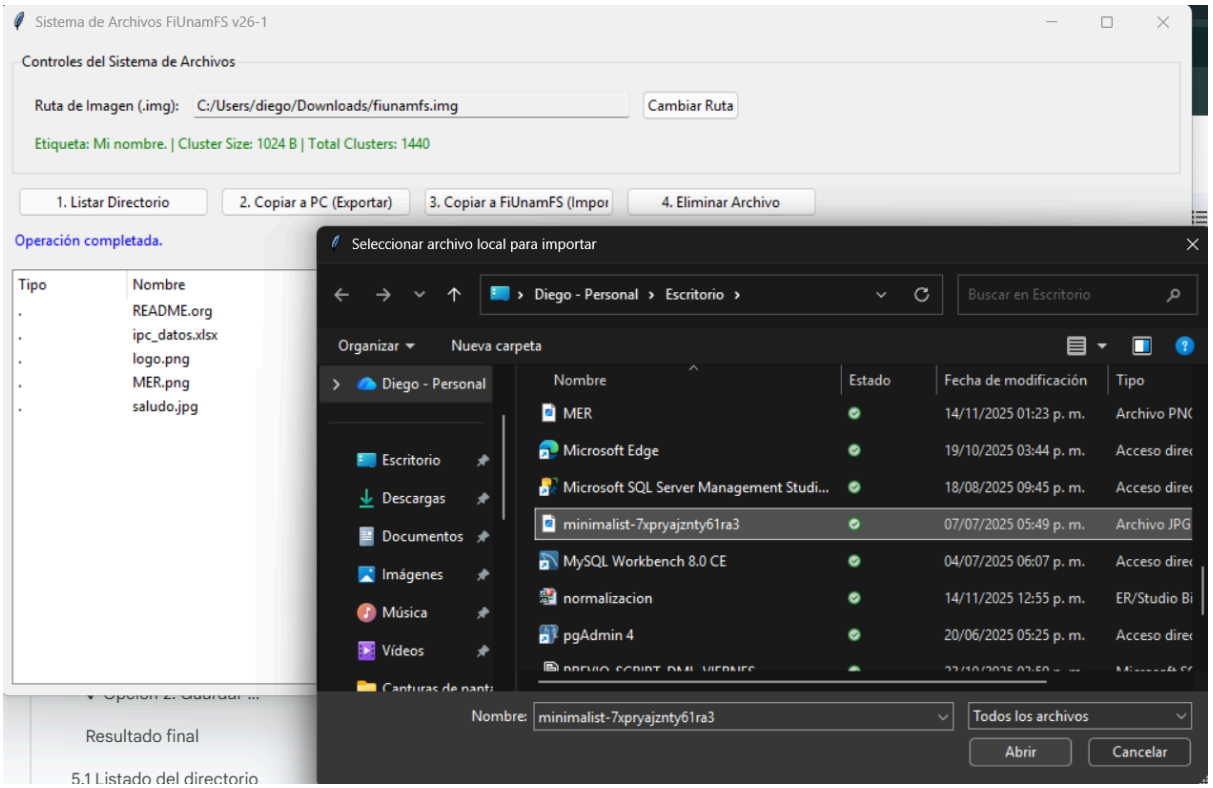


comprobamos que se haya copiado a la ruta seleccionada y abrimos el archivo

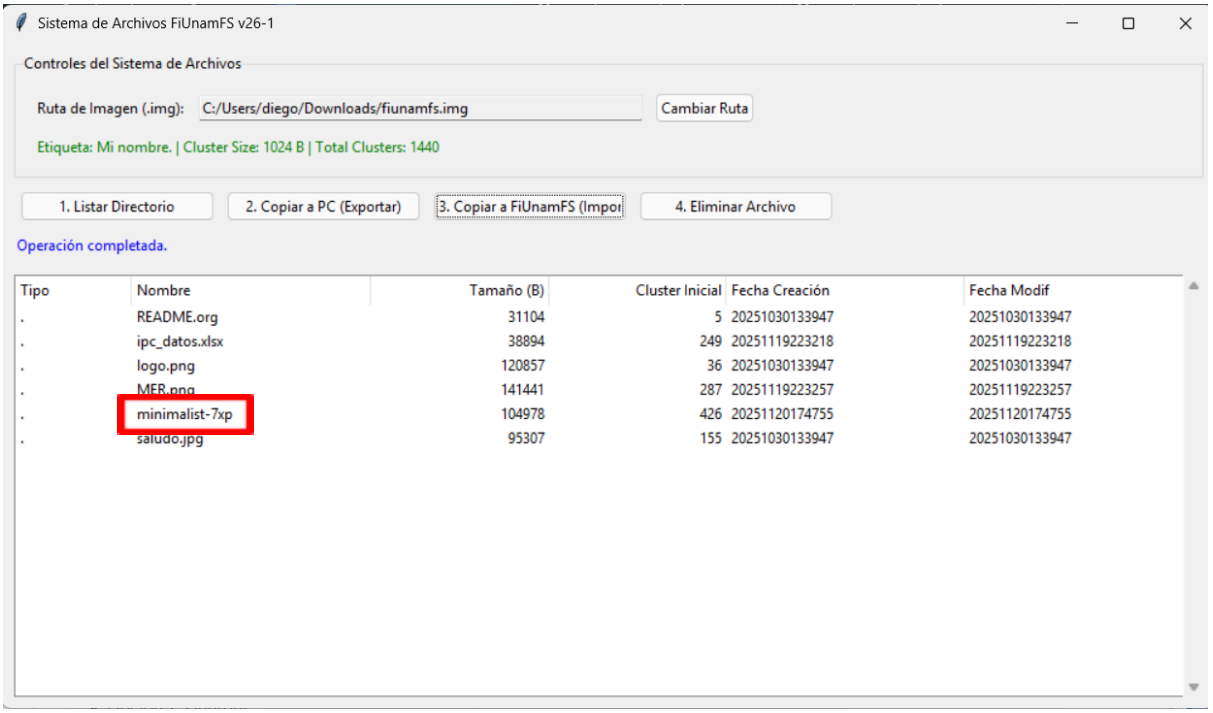


5.3 Copiado hacia FiUnamFS

Click en “Copiar a FiUnamFS” → elegir archivo local.

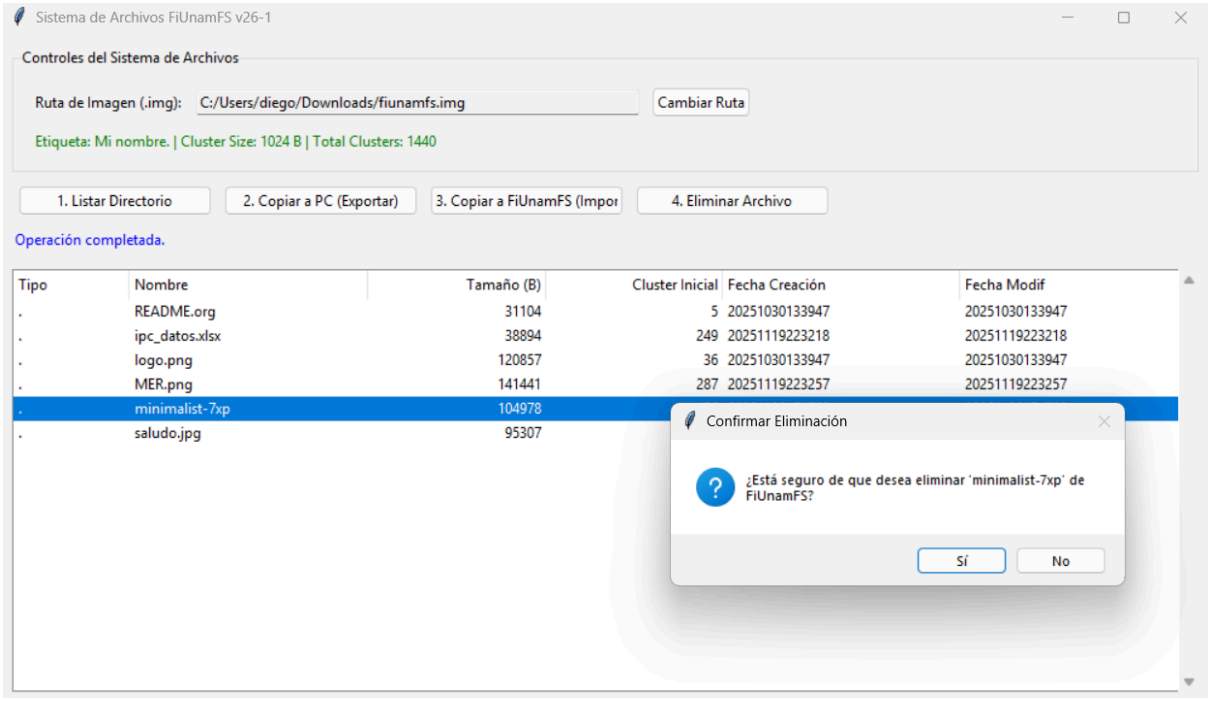


Damos clic en abrir y se carga automáticamente



5.4 Eliminación

Seleccionar archivo → “Eliminar Archivo” → confirmar.



comprobamos la eliminación del archivo