UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Faculta de ingeniería

Sistemas Operativos

Grupo: 06

Semestre 2026-1

-Tejeda Vaca Abraham: 321328456

Proyecto: (Micro) sistema de archivos multihilos

Fecha de entrega: 20/Noviembre/2025

1. Introducción

En este proyecto de la materia Sistemas Operativos implemento un micro–sistema de archivos multihilos que trabaja sobre fiunamfs.img, basada en el formato FiUnamFS desarrollado para la Facultad de Ingeniería.

FiUnamFS simula:

- Un superbloque en el *cluster* 0, que guarda el nombre del sistema (FiUnamFS), la etiqueta del volumen, el tamaño de cluster, los clusters de directorio y el total de clusters.
- Un directorio plano ocupando varios clusters, donde cada entrada tiene tamaño fijo (64 bytes) y almacena tipo, nombre, cluster inicial, tamaño y marcas de tiempo de creación/modificación.
- Una zona de datos contiguos, a partir de la cual se guardan los contenidos reales de cada archivo.

El objetivo del proyecto es ofrecer un programa que permita interactuar con esta imagen de disco sin montarla como sistema de archivos del sistema operativo, y al mismo tiempo practicar conceptos de concurrencia y sincronización vistos en clase. Para ello, la solución se compone de dos bloques principales:

- Una clase FiUnamFS, que encapsula la lógica de bajo nivel para leer y modificar la imagen: lectura del superbloque, recorrido del directorio, búsqueda de espacio contiguo, copia de datos y eliminación de archivos.
- Una clase FiUnamFSApp, que implementa una interfaz gráfica con Tkinter y utiliza múltiples hilos para que las operaciones de entrada y salida.

1.1 Objetivos

1.1.1 Objetivo general

Diseñar e implementar un programa en Python que pueda leer, crear, copiar y eliminar archivos dentro de una imagen FiUnamFS, utilizando hilos de ejecución y mecanismos de sincronización, y ofreciendo una interfaz gráfica sencilla que no requiera conocer los detalles internos del sistema de archivos.

1.1.2 Objetivos específicos

- Interpretar la estructura de FiUnamFS:
    - Leer y validar el superbloque.
- Implementar operaciones básicas sobre el sistema de archivos:

-Listar los contenidos del directorio.

-Copiar un archivo desde FiUnamFS hacia el sistema de mi pc.

-Copiar un archivo desde el sistema anfitrión hacia FiUnamFS.

-Eliminar un archivo de FiUnamFS.

- Aplicar conceptos de concurrencia y sincronización:
    - Utilizar múltiples hilos para ejecutar en paralelo las operaciones de copia y de actualización del listado, manteniendo receptiva la interfaz gráfica.
    - Proteger el acceso a la imagen con un mutex para evitar condiciones de carrera.
    - Coordinar el hilo que lista archivos con las otras operaciones de forma que se actualice la tabla sólo cuando haya cambios.

- Cumplir con los criterios de la rúbrica:
    - Entregar un proyecto con código legible, comentado donde aporta a la comprensión de la lógica, y organizado por bloques (FiUnamFS, FiUnamFSApp, main).
    - Proveer documentación externa con instrucciones claras de ejecución y ejemplos de uso, de modo que el programa pueda probarse al primer intento sin modificar el código.

2. Desarrollo

El código del proyecto se organiza en tres bloques principales:

- La clase FiUnamFS, que se encarga de toda la lógica del sistema de archivos.
- La clase FiUnamFSApp, que implementa la interfaz gráfica y coordina los hilos.
- El bloque main, que inicializa todo y arranca la aplicación.

2.1 Clase FiUnamFS

```python
import os
import struct
import threading
import math
import tkinter as tk
import sys
from datetime import datetime
from tkinter import ttk, messagebox, filedialog

FS_NAME = "FiUnamFS"
FS_VERSION = "26-1"


FILE_TYPE_USED = b'.'
FILE_TYPE_FREE = b'-'
EMPTY_NAME_PATTERN = b'............'

VCListFiles = threading.Condition()

class FiUnamFS:
    """
    Implementación del micro-sistema de archivos FiUnamFS.
    Se encarga de operar directamente sobre la imagen de disco.
    """

    SUPERBLOCK_STRUCT = struct.Struct('<9s1x5s5x16s4xI1xI1xI')
    DIR_ENTRY_STRUCT = struct.Struct('<c15sII14s14s12x')

    def __init__(self, disk_path: str) -> None:
        self.disk_path = disk_path
        self.lock = threading.Lock()
        self.archivos = []
        self.fs_name = None
        self.version = None
        self.volume_label = None
        self.cluster_size = None
        self.dir_clusters = None
        self.total_clusters = None
        self.dir_start_cluster = 1
        self.data_start_cluster = None
        self._leer_superbloque()
```

2.1.1 Superbloque

```python
    def _leer_superbloque(self) -> None:
        """Lee y valida el superbloque de FiUnamFS."""
        with open(self.disk_path, 'rb') as f:
            data = f.read(self.SUPERBLOCK_STRUCT.size)

        nombre_raw, version_raw, etiqueta_raw, tam_cluster, dir_clusters, total_clusters = \
            self.SUPERBLOCK_STRUCT.unpack(data)

        nombre = nombre_raw.decode('ascii', errors='ignore').strip('\x00')
        version = version_raw.decode('ascii', errors='ignore').strip('\x00')
        etiqueta = etiqueta_raw.decode('ascii', errors='ignore').strip('\x00')

        if nombre != FS_NAME:
            raise ValueError(
                f"Sistema de archivos inválido: se esperaba nombre '{FS_NAME}', se encontró '{nombre}'"
            )
        if version != FS_VERSION:
            raise ValueError(
                f"Versión inválida de FiUnamFS: se esperaba '{FS_VERSION}', se encontró '{version}'"
            )

        self.fs_name = nombre
        self.version = version
        self.volume_label = etiqueta
        self.cluster_size = tam_cluster
        self.dir_clusters = dir_clusters
        self.total_clusters = total_clusters
        self.data_start_cluster = self.dir_start_cluster + self.dir_clusters

    def get_superblock_info(self) -> dict:
        """Devuelve la información del superbloque en un diccionario amigable."""
        return {
            "Nombre": self.fs_name,
            "Versión": self.version,
            "Etiqueta de Volumen": self.volume_label,
            "Tamaño de Cluster": self.cluster_size,
            "Número de Clusters de Directorio": self.dir_clusters,
            "Total de Clusters": self.total_clusters,
        }

```

Al crear un objeto FiUnamFS, el constructor abre la imagen fiunamfs.img y llama a _leer_superbloque().
Este método:

- Lee los primeros bytes del archivo con una estructura struct fija.
- Extrae nombre del sistema, versión, etiqueta de volumen, tamaño de clúster, número de clústeres de directorio y total de clústeres.
- Valida que el nombre sea FiUnamFS y que la versión coincida con la de la imagen.
- Calcula a partir de ahí dónde empiezan los clústeres de datos.

2.1.2 Directorio

```python
    def listar_directorio(self):
        """Lee todas las entradas válidas del directorio y las guarda en self.archivos."""
        archivos = []
        entries_per_cluster = self.cluster_size // self.DIR_ENTRY_STRUCT.size

        with open(self.disk_path, 'rb') as f:
            for cluster in range(self.dir_start_cluster,
                                 self.dir_start_cluster + self.dir_clusters):
                base_offset = cluster * self.cluster_size
                f.seek(base_offset)

                for entry_index in range(entries_per_cluster):
                    entry_data = f.read(self.DIR_ENTRY_STRUCT.size)
                    if len(entry_data) < self.DIR_ENTRY_STRUCT.size:
                        break

                    tipo, nombre_raw, cluster_inicial, tamaño, creado_raw, modificado_raw = \
                        self.DIR_ENTRY_STRUCT.unpack(entry_data)
                    nombre_decoded = nombre_raw.decode('ascii', errors='ignore').rstrip('\x00')
                    if tipo == FILE_TYPE_FREE:
                        continue
                    if not nombre_decoded:
                        continue
                    if nombre_decoded.startswith(EMPTY_NAME_PATTERN.decode('ascii')):
                        continue
                    if tipo != FILE_TYPE_USED:
                        continue

                    creado_str = creado_raw.decode('ascii', errors='ignore').rstrip('\x00')
                    mod_str = modificado_raw.decode('ascii', errors='ignore').rstrip('\x00')

                    archivos.append({
                        "Nombre": nombre_decoded,
                        "Tamaño": tamaño,
                        "Creado": creado_str,
                        "Modificado": mod_str,
                        "Cluster Inicial": cluster_inicial,
                        })

        self.archivos = archivos
        return archivos

    def _buscar_entrada_directorio_libre(self):
        """
        Busca una entrada libre en el directorio y devuelve el offset absoluto
        dentro del archivo de imagen donde se puede escribir.
        """
        entries_per_cluster = self.cluster_size // self.DIR_ENTRY_STRUCT.size
        empty_name_str = EMPTY_NAME_PATTERN.decode('ascii')

        with open(self.disk_path, 'rb') as f:
            for cluster in range(self.dir_start_cluster,
                                 self.dir_start_cluster + self.dir_clusters):
                base_offset = cluster * self.cluster_size

                for entry_index in range(entries_per_cluster):
                    entry_offset = base_offset + entry_index * self.DIR_ENTRY_STRUCT.size
                    f.seek(entry_offset)
                    entry_data = f.read(self.DIR_ENTRY_STRUCT.size)
                    if len(entry_data) < self.DIR_ENTRY_STRUCT.size:
                        continue

                    tipo, nombre_raw, _, _, _, _ = self.DIR_ENTRY_STRUCT.unpack(entry_data)
                    nombre_decoded = nombre_raw.decode('ascii', errors='ignore').rstrip('\x00')

                    if tipo == FILE_TYPE_FREE or nombre_decoded.startswith(empty_name_str):
                        return entry_offset

        return None
```

El directorio se maneja con el método listar_directorio():

- Recorre los clústeres reservados al directorio, leyendo entradas de 64 bytes.
- Usa otra estructura struct para desempaquetar tipo, nombre, clúster inicial, tamaño y fechas.
- Ignora las entradas marcadas como vacías y guarda las válidas en una lista self.archivos, que después usa la GUI.

2.1.3 Gestión de espacio en datos

```
162     def _buscar_espacio_contiguo(self, tamaño_bytes: int):
163         """
164         Busca espacio contiguo suficiente en la zona de datos para almacenar
165         'tamaño_bytes'. Devuelve el cluster inicial o None si no hay espacio.
166         """
167         clusters_necesarios = math.ceil(tamaño_bytes / self.cluster_size)
168 
169         with open(self.disk_path, 'rb') as f:
170             run_start = None
171             run_length = 0
172 
173             for cluster in range(self.data_start_cluster, self.total_clusters):
174                 f.seek(cluster * self.cluster_size)
175                 data = f.read(self.cluster_size)
176                 if len(data) < self.cluster_size:
177                     break
178 
179                 if all(b == 0 for b in data):
180                     if run_length == 0:
181                         run_start = cluster
182                     run_length += 1
183 
184                     if run_length >= clusters_necesarios:
185                         return run_start
186                 else:
187                     run_start = None
188                     run_length = 0
189 
190         return None
191 
```

Para escribir archivos nuevos se usa _buscar_espacio_contiguo(tamaño):

- Calcula cuántos clústeres se necesitan para el tamaño solicitado.
- Recorre la zona de datos buscando una secuencia de clústeres llenos de ceros.

- Si encuentra un bloque contiguo lo devuelve como clúster inicial; si no, indica falta de espacio.

2.1.4 Operaciones de alto nivel

Sobre esa base se implementan tres operaciones públicas:

- copiar_desde_fs(nombre, destino_pc): lee los bytes de un archivo de FiUnamFS y los guarda en el sistema anfitrión.
- copiar_a_fs(ruta_origen): toma un archivo del host, le asigna espacio contiguo en la imagen, crea la entrada de directorio y copia los datos.
- eliminar_archivo(nombre): localiza la entrada del archivo, la marca como libre, pone a cero el espacio de datos y actualiza la lista interna.

Todas estas funciones se ejecutan dentro de un Lock para que nunca haya dos hilos escribiendo en la imagen a la vez.

2.2 Clase FiUnamFSApp

2.2.1 Interfaz gráfica

```
377  class FiUnamFSApp:
378      def __init__(self, root, fs: FiUnamFS):
379          self.fs = fs
380          self.root = root
381          self.root.title("FiUnamFS")
382
383          self.superblock_info = tk.Label(root, text="", justify='left')
384          self.superblock_info.grid(row=0, column=0, columnspan=2, padx=10, pady=10)
385
386          self.tree = ttk.Treeview(
387              root,
388              columns=("Nombre", "Tamaño", "Creado", "Cluster Inicial", "Modificado"),
389              show='headings'
390          )
391          self.tree.heading("Nombre", text="Nombre")
392          self.tree.heading("Tamaño", text="Tamaño (bytes)")
393          self.tree.heading("Creado", text="Creado")
394          self.tree.heading("Cluster Inicial", text="Cluster Inicial")
395          self.tree.heading("Modificado", text="Modificado")
396
397          self.tree.column("Nombre", width=200)
398          self.tree.column("Tamaño", width=120, anchor='e')
399          self.tree.column("Creado", width=150)
400          self.tree.column("Cluster Inicial", width=120, anchor='e')
401          self.tree.column("Modificado", width=150)
402
403          self.tree_scroll = ttk.Scrollbar(root, orient="vertical", command=self.tree.yview)
404          self.tree.configure(yscrollcommand=self.tree_scroll.set)
405
406          self.tree_scroll.grid(row=1, column=2, sticky='ns')
407          self.tree.grid(row=1, column=0, columnspan=2, padx=10, pady=10)
408
409          self.list_button = tk.Button(root, text="Listar archivos", command=self.notify_list_files)
410          self.list_button.grid(row=2, column=0, padx=10, pady=5)
411
412          self.copy_to_pc_button = tk.Button(root, text="Copiar a PC", command=self.copy_to_pc)
413          self.copy_to_pc_button.grid(row=2, column=1, padx=10, pady=5)
414
415          self.copy_to_fs_button = tk.Button(root, text="Copiar a FiUnamFS", command=self.copy_to_fs)
416          self.copy_to_fs_button.grid(row=3, column=0, padx=10, pady=5)
417
418          self.delete_button = tk.Button(root, text="Eliminar archivo", command=self.delete_file)
419          self.delete_button.grid(row=3, column=1, padx=10, pady=5)
420
421          self.show_superblock_info()
422
423          self.list_thread = threading.Thread(target=self.list_files_loop, daemon=True)
424          self.list_thread.start()
425
```

FiUnamFSApp recibe una instancia de FiUnamFS y construye la ventana principal con Tkinter:

- Arriba muestra una etiqueta con la información del superbloque.
- En medio un Treeview con columnas: Nombre, Tamaño, Creado, Clúster inicial, Modificado.
- Abajo cuatro botones: Listar archivos, Copiar a PC, Copiar a FiUnamFS y Eliminar archivo.

2.2.2 SuperbLoque en GUI

```
429     def show_superblock_info(self):
430         info = self.fs.get_superblock_info()
431         texto = (
432             f"Nombre: {info['Nombre']}\n"
433             f"Versión: {info['Versión']}\n"
434             f"Etiqueta de Volumen: {info['Etiqueta de Volumen']}\n"
435             f"Tamaño de Cluster: {info['Tamaño de Cluster']} bytes\n"
436             f"Número de Clusters de Directorio: {info['Número de Clusters de Directorio']}\n"
437             f"Total de Clusters: {info['Total de Clusters']}"
438         )
439         self.superblock_info.config(text=texto)
440
```

El método show_superblock_info() llama a fs.get_superblock_info() y formatea esos datos en texto: nombre, versión, etiqueta, tamaño de clúster, clústeres de directorio y total.
Con eso el usuario ve de inmediato con qué imagen está trabajando.

2.2.3 Listado de archivos (concurrente)

```
444     def notify_list_files(self):
445         """Despierta al hilo enlistador para refrescar el listado."""
446         with VCListFiles:
447             VCListFiles.notify_all()
448 
449     def list_files_loop(self):
450         """Hilo que espera notificaciones y actualiza la tabla de archivos."""
451         while True:
452             with VCListFiles:
453                 VCListFiles.wait()
454 
455             with self.fs.lock:
456                 files = self.fs.listar_directorio()
457             def actualizar_tree():
458                 self.tree.delete(*self.tree.get_children())
459                 for file in files:
460                     creado_fmt = self._formatear_fecha(file["Creado"])
461                     mod_fmt = self._formatear_fecha(file["Modificado"])
462                     self.tree.insert(
463                         "",
464                         "end",
465                         values=(
466                             file["Nombre"],
467                             file["Tamaño"],
468                             creado_fmt,
469                             file["Cluster Inicial"],
470                             mod_fmt,
471                         ),
472                     )
473 
474             self.root.after(0, actualizar_tree)
475 
476     @staticmethod
477     def _formatear_fecha(fecha_raw: str) -> str:
478         """Convierte AAAAMMDDHHMMSS a 'YYYY-MM-DD HH:MM:SS'."""
479         try:
480             dt = datetime.strptime(fecha_raw, "%Y%m%d%H%M%S")
481             return dt.strftime("%Y-%m-%d %H:%M:%S")
482         except Exception:
483             return fecha_raw or "-"
```

La app crea un hilo separado list_thread que ejecuta list_files_loop():

- Este hilo espera en una Condition global (VCListFiles).
- Cuando el usuario presiona “Listar archivos” o se termina una operación que modifica el directorio, se hace notify_all().
- El hilo despierta, llama a fs.listar_directorio() (bajo el Lock) y luego actualiza el Treeview desde el hilo principal con root.after(0, ...).

Así el listado se refresca de manera concurrente sin congelar la interfaz.

2.2.4 Operaciones GUI

```
488     def copy_to_pc(self):
489         selected_items = self.tree.selection()
490         if not selected_items:
491             messagebox.showwarning("Atención", "Selecciona al menos un archivo para copiar.")
492             return
493 
494         def worker(nombre_archivo, destino):
495             self.fs.copiar_desde_fs(nombre_archivo, destino)
496 
497         for item in selected_items:
498             nombre_archivo = self.tree.item(item)["values"][0]
499             destino = filedialog.asksaveasfilename(initialfile=nombre_archivo)
500             if not destino:
501                 continue
502             hilo = threading.Thread(target=worker, args=(nombre_archivo, destino), daemon=True)
503             hilo.start()
504 
505     def copy_to_fs(self):
506         rutas = filedialog.askopenfilenames()
507         if not rutas:
508             return
509 
510         def worker(ruta):
511             self.fs.copiar_a_fs(ruta)
512 
513         hilos = []
514         for ruta in rutas:
515             hilo = threading.Thread(target=worker, args=(ruta,), daemon=True)
516             hilos.append(hilo)
517             hilo.start()
518 
519     def delete_file(self):
520         selected_items = self.tree.selection()
521         if not selected_items:
522             messagebox.showwarning("Atención", "Selecciona al menos un archivo para eliminar.")
523             return
524 
525         def worker(nombre_archivo):
526             self.fs.eliminar_archivo(nombre_archivo)
527 
528         hilos = []
529         for item in selected_items:
530             nombre_archivo = self.tree.item(item)["values"][0]
531             confirmar = messagebox.askyesno(
532                 "Confirmar eliminación",
533                 f"¿Seguro que deseas eliminar '{nombre_archivo}' de FiUnamFS?"
534             )
535             if confirmar:
536                 hilo = threading.Thread(target=worker, args=(nombre_archivo,), daemon=True)
537                 hilos.append(hilo)
538                 hilo.start()
```

Cada botón de la GUI dispara hilos “worker”:

- Copiar a PC: por cada archivo seleccionado abre un diálogo de guardado y lanza un hilo que llama a fs.copiar_desde_fs.

- Copiar a FiUnamFS: abre un diálogo para seleccionar uno o varios archivos y crea un hilo por cada uno, que ejecuta fs.copiar_a_fs.
- Eliminar archivo: pide confirmación y, si el usuario acepta, lanza un hilo que ejecuta fs.eliminar_archivo.

En todos los casos, las operaciones pesadas corren en segundo plano y al terminar notifican al hilo enlistador para que actualice la tabla.

2.3 Bloque main

El bloque if __name__ == "__main__": es el punto de entrada:

- Crea la ventana principal de Tkinter.
- Determina la ruta de la imagen fiunamfs.img (y si no se encuentra, puede pedirse al usuario).
- Inicializa un objeto FiUnamFS con esa ruta; si hay algún error (versión incorrecta, archivo inexistente), muestra un mensaje y termina.
- Si la carga es correcta, crea FiUnamFSApp con esa instancia y llama a root.mainloop() para iniciar el ciclo de eventos de la GUI.

```
544  if __name__ == "__main__":
545      root = tk.Tk()
546      root.withdraw()
547
548      script_dir = os.path.dirname(os.path.abspath(__file__))
549      default_img = os.path.join(script_dir, "fiunamfs.img")
550
551      if os.path.exists(default_img):
552          disk_image_path = default_img
553      else:
554
555          messagebox.showinfo(
556              "Seleccionar imagen de FiUnamFS",
557              "No se encontró 'fiunamfs.img' junto a PFSO.py.\n\n"
558              "Selecciona el archivo de imagen de FiUnamFS."
559          )
560          disk_image_path = filedialog.askopenfilename(
561              title="Selecciona fiunamfs.img",
562              initialdir=script_dir,
563              filetypes=[("Imagen FiUnamFS", "*.img"), ("Todos los archivos", "*.*")]
564          )
565
566          if not disk_image_path:
567              messagebox.showerror(
568                  "Error",
569                  "No se seleccionó ninguna imagen de FiUnamFS. El programa se cerrará."
570              )
571              root.destroy()
572              sys.exit(1)
573
574      try:
575          fs = FiUnamFS(disk_image_path)
576      except Exception as e:
577          messagebox.showerror("Error inicializando FiUnamFS", str(e))
578          root.destroy()
579          sys.exit(1)
580
581      root.deiconify()
582      app = FiUnamFSApp(root, fs)
583      root.mainloop()
```

3. Ejecución y uso del programa

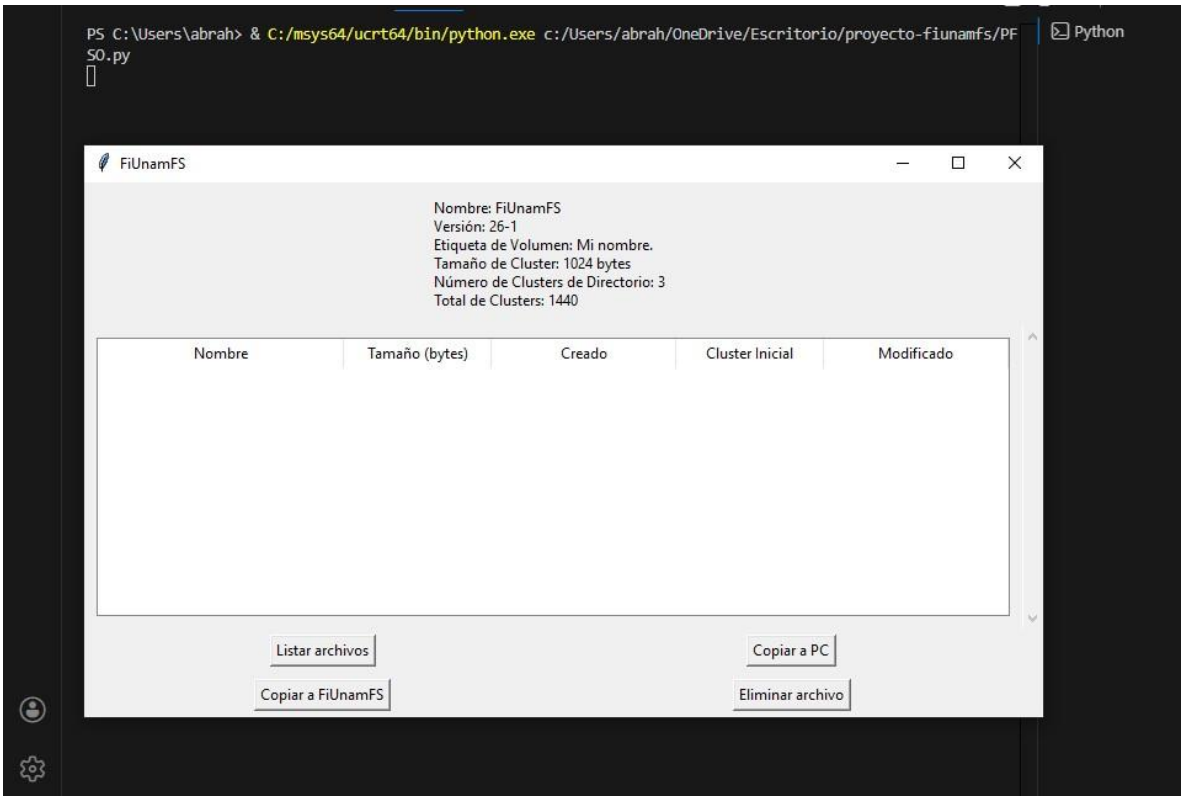
3.1 Requisitos previos

- Python 3.
- Archivo de imagen del sistema de archivos: fiunamfs.img.
- Archivo del programa: PFSO.py (código del proyecto).

Se recomienda colocar PFSO.py y fiunamfs.img en el mismo directorio de trabajo. Ya que yo tuve problemas al encontrar el archivo.

3.2 Ejecución del programa

1. Abrir una terminal en la carpeta del proyecto.
2. Ejecutar:
3. python PFSO.py
4. Si la imagen es válida, se abre una ventana con:
    - Datos del superbloque
    - Una tabla inicialmente vacía.
    - Cuatro botones: Listar archivos, Copiar a PC, Copiar a FiUnamFS, Eliminar archivo.

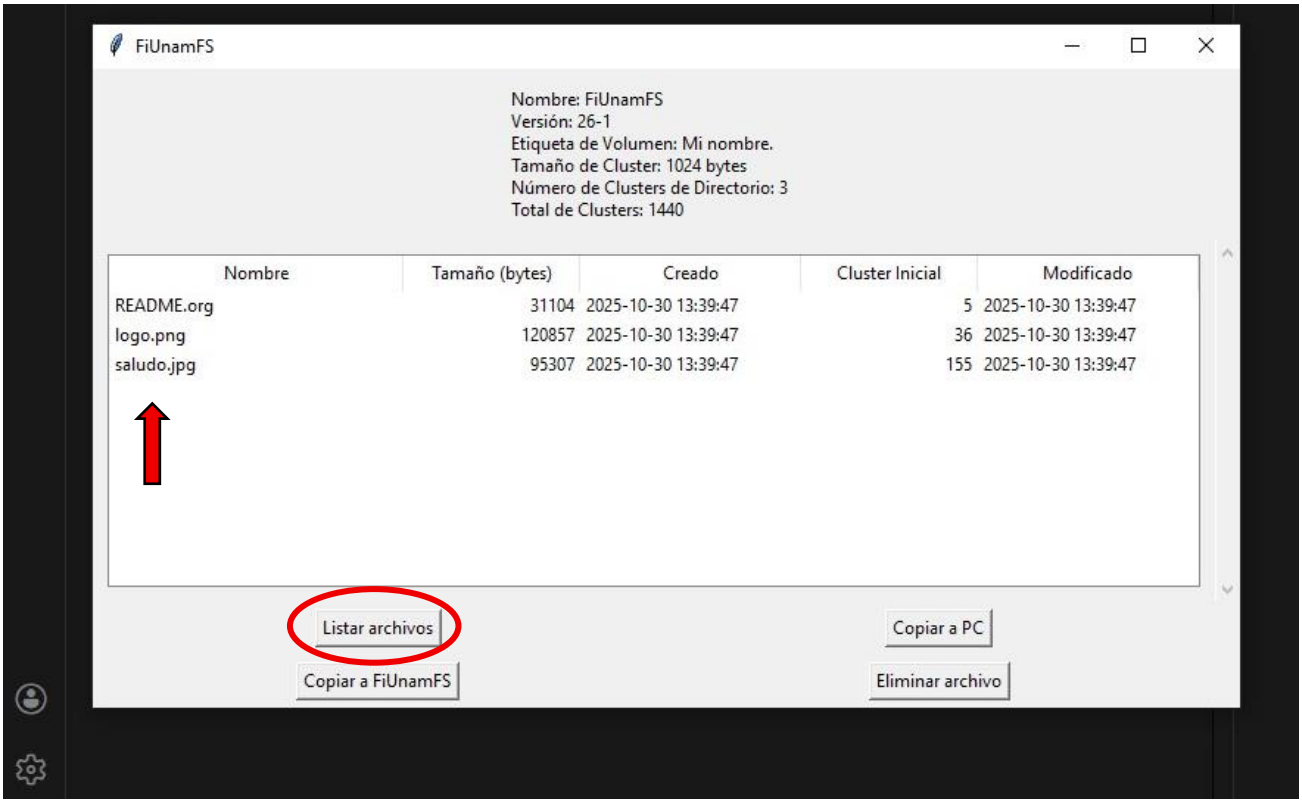


3.3 Listar los archivos de FiUnamFS

Esta operación permite ver el contenido actual del directorio plano dentro de la imagen.

Pasos:

1. En la ventana principal, pulsar el botón “Listar archivos”.
2. El programa despierta el hilo enlistador, que recorre el directorio de FiUnamFS y actualiza la tabla.
3. En la tabla aparecerá una fila por cada archivo, con:
    - Nombre
    - Tamaño (bytes)
    - Fecha de creación
    - Clúster inicial
    - Fecha de última modificación



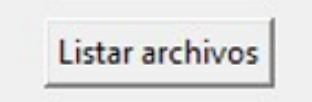
Cada vez que se copia o elimina un archivo, se puede volver a pulsar “Listar archivos” para comprobar los cambios.

3.4 Copiar archivos desde FiUnamFS al dispositivo (PC)

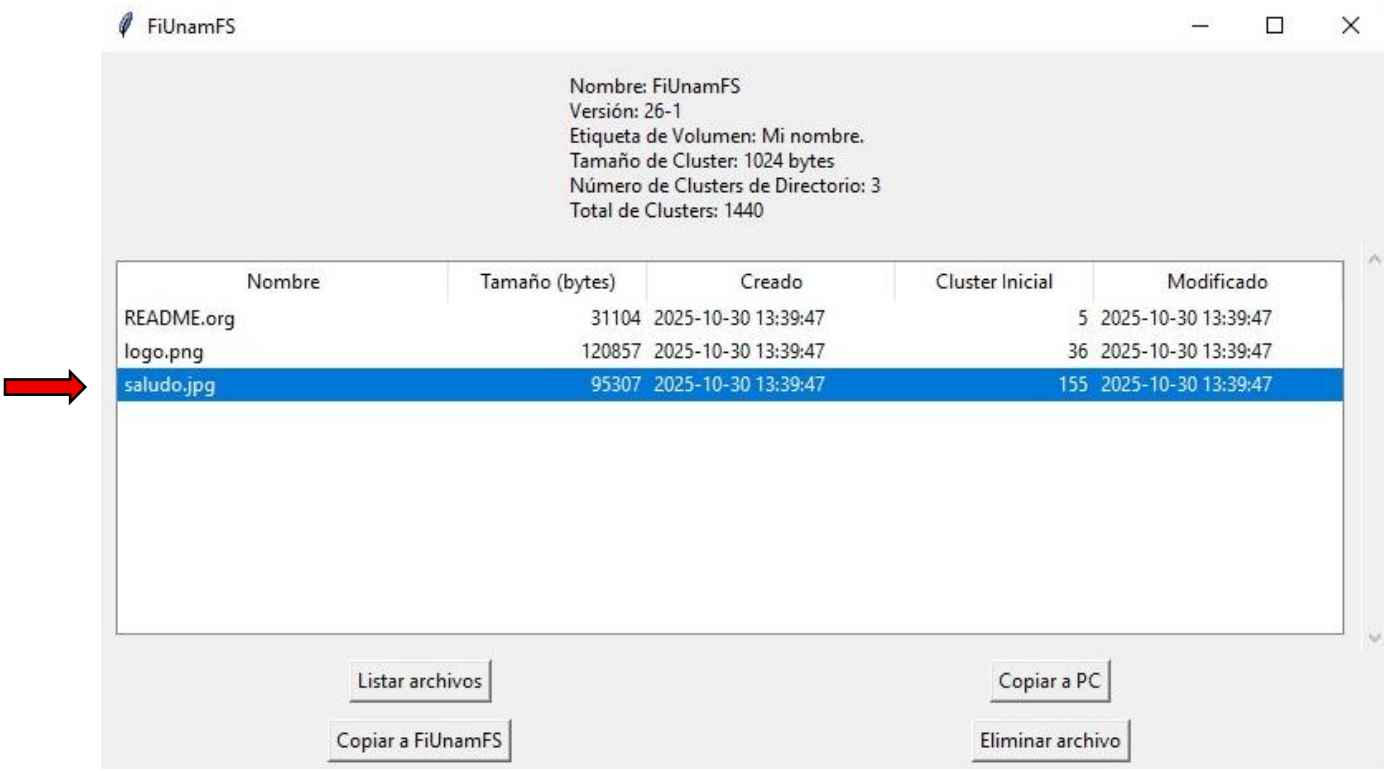
Esta opción copia uno o varios archivos de la imagen fiunamfs.img hacia el sistema de archivos del host.

Pasos:

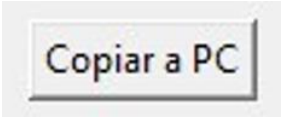
1. Pulsar “Listar archivos” para tener el directorio actualizado.



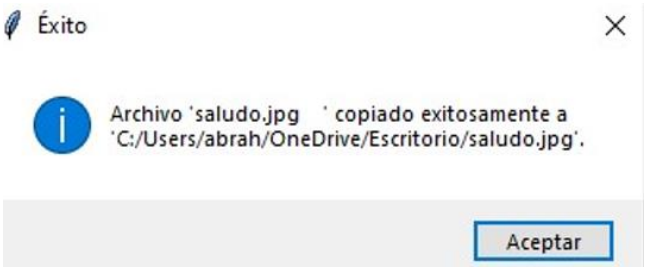
2. En la tabla, seleccionar el archivo (o archivos) a copiar.



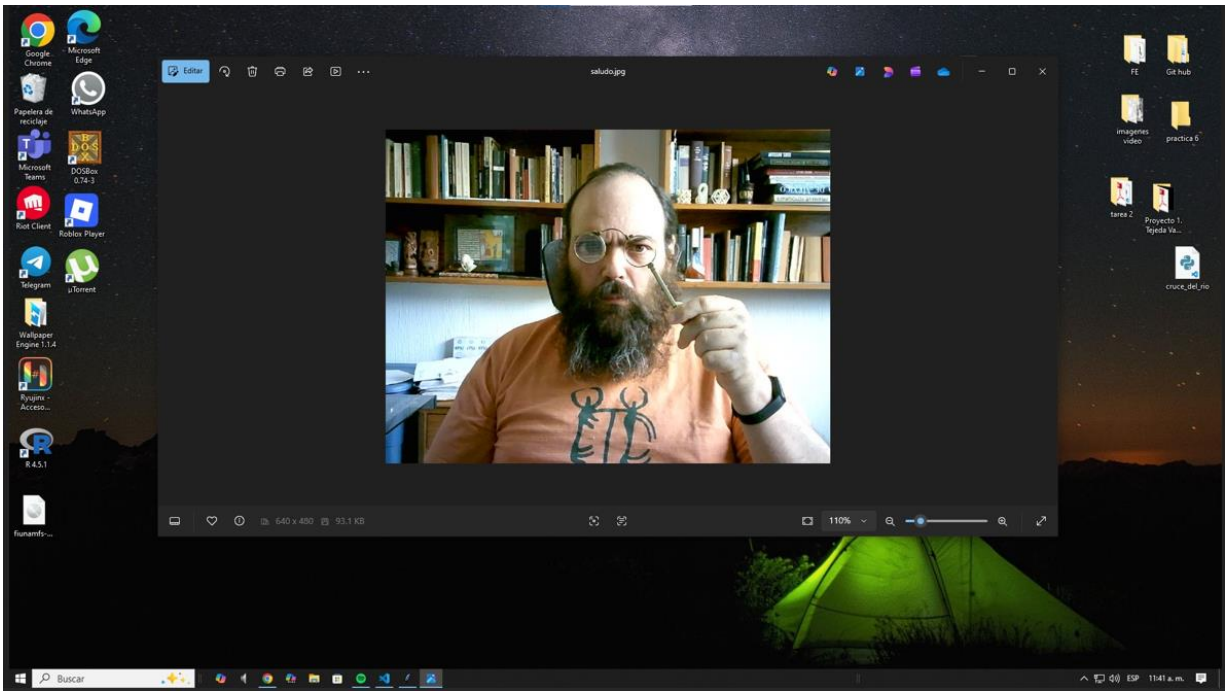
3. Hacer clic en el botón “Copiar a PC”.



4. Para cada archivo seleccionado se abre un cuadro de diálogo “Guardar como”:
    - Elegir la carpeta de destino en el equipo.
    - Confirmar el nombre del archivo y pulsar Guardar.
5. El programa crea un hilo por cada archivo y realiza la copia en segundo plano.
6. Al terminar, se muestra un mensaje indicando que la operación fue exitosa.



El archivo copiado queda disponible como un archivo normal del sistema operativo.



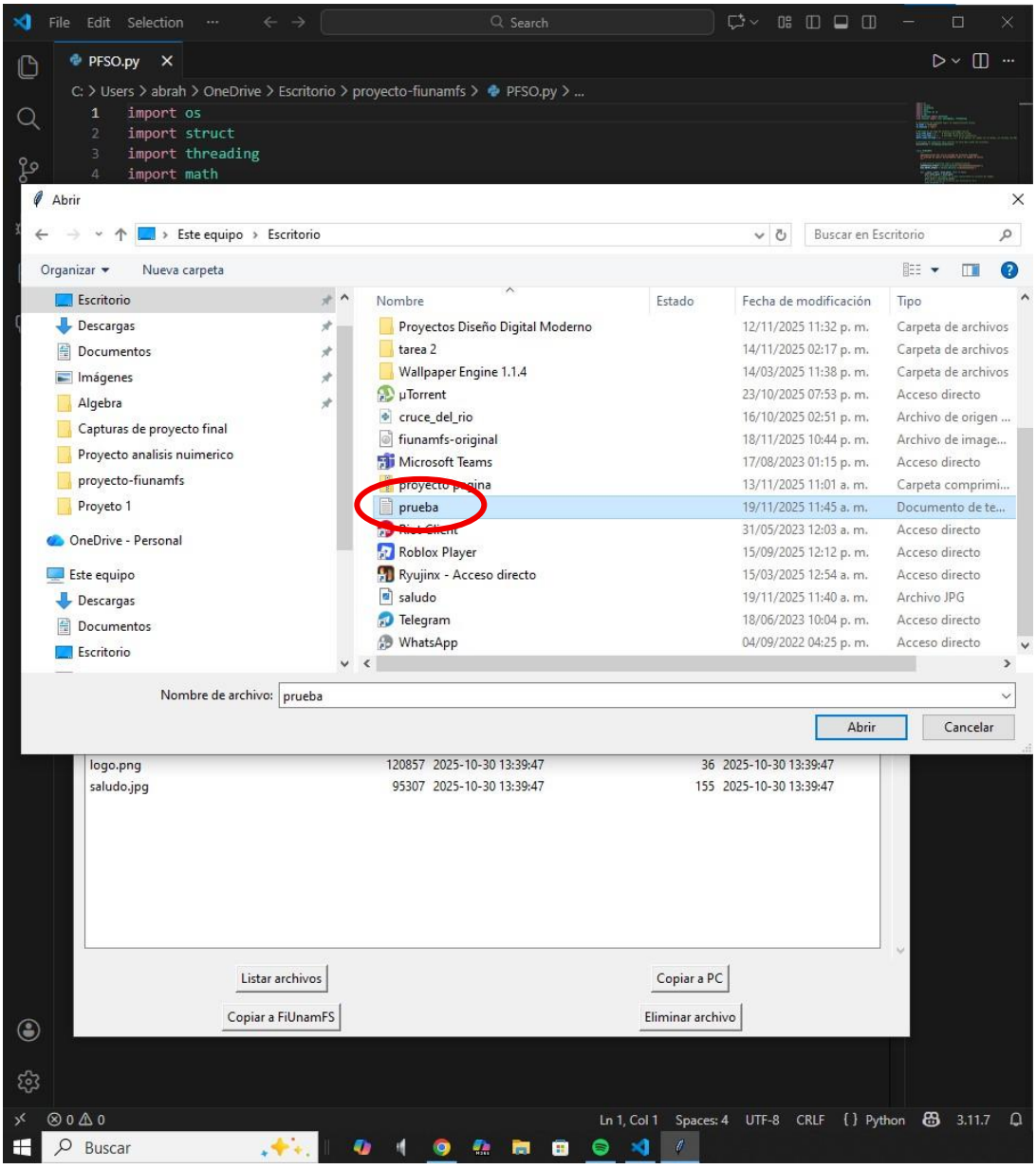
3.5 Copiar archivos desde el dispositivo (PC) hacia FiUnamFS

Esta opción permite agregar nuevos archivos dentro de la imagen FiUnamFS, siempre que haya espacio contiguo y una entrada libre en el directorio.

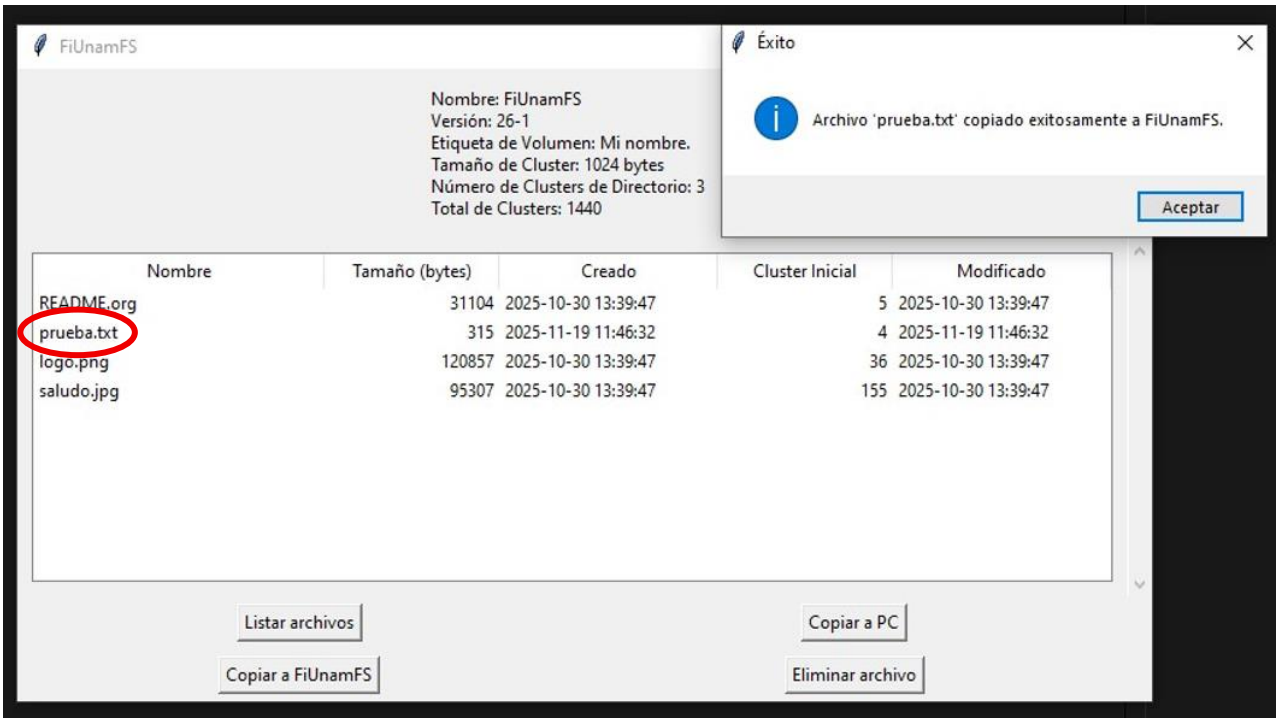
Pasos:

1. Pulsar el botón “Copiar a FiUnamFS”.

2. En el cuadro de diálogo, seleccionar uno o varios archivos del sistema anfitrión.



3. Aceptar la selección.
4. El programa lanza un hilo por cada archivo seleccionado y, para cada uno:
    - Comprueba que no exista ya un archivo con el mismo nombre en FiUnamFS.
    - Busca espacio contiguo suficiente en la zona de datos.
    - Busca una entrada libre en el directorio.
    - Escribe la entrada con nombre, tamaño, clúster inicial y marcas de tiempo.
    - Copia los datos al área de datos.
5. Al finalizar cada copia se muestra un mensaje de éxito.



6. Para verificar los cambios, pulsar de nuevo “Listar archivos” y comprobar que los nuevos archivos aparecen en la tabla.

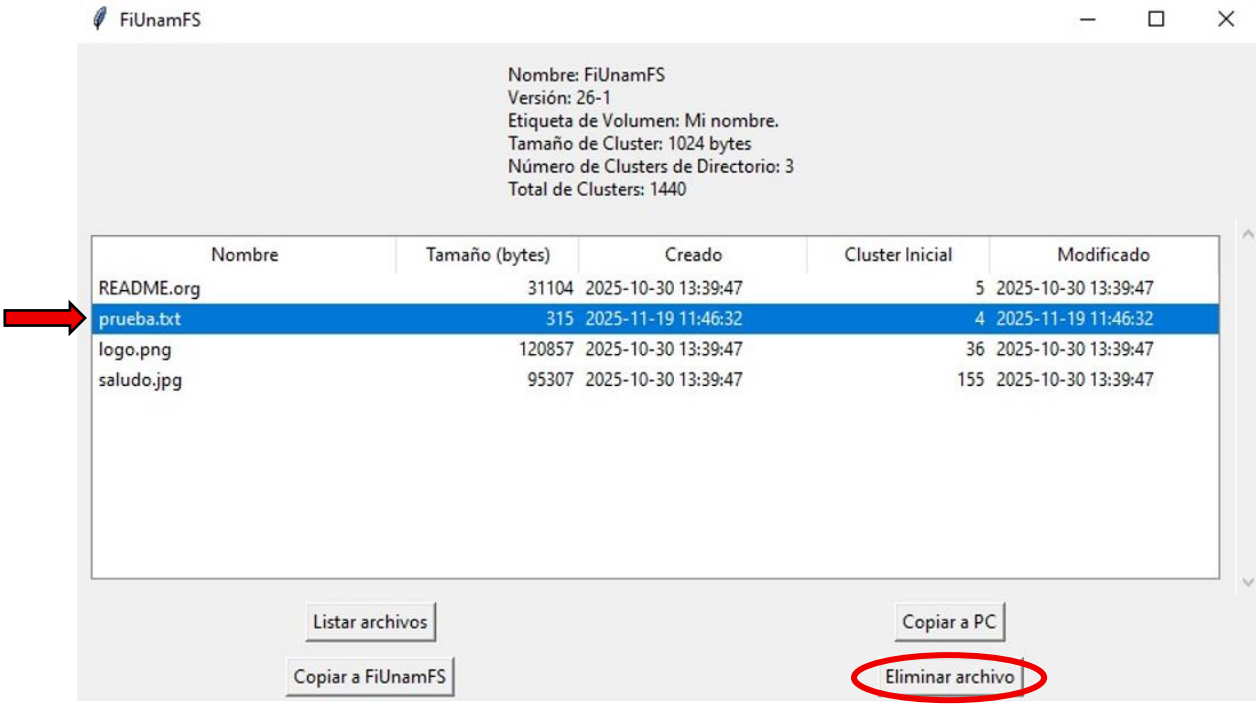
Si no hay espacio contiguo o ya no hay entradas de directorio libres, el programa muestra un mensaje de error explicando el motivo.

3.6 Eliminar archivos dentro de FiUnamFS

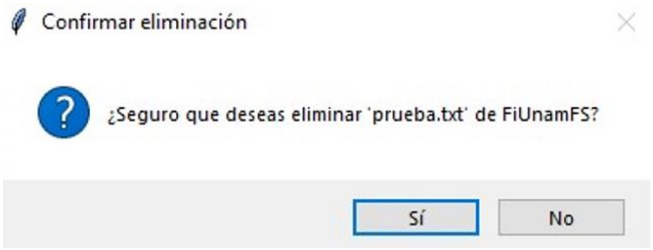
Esta operación borra archivos de la imagen, liberando su entrada en el directorio y sus clústeres de datos.

Pasos:

1. Pulsar “Listar archivos” para ver el estado actual.
2. En la tabla, seleccionar el archivo que se desea eliminar.
3. Hacer clic en “Eliminar archivo”.



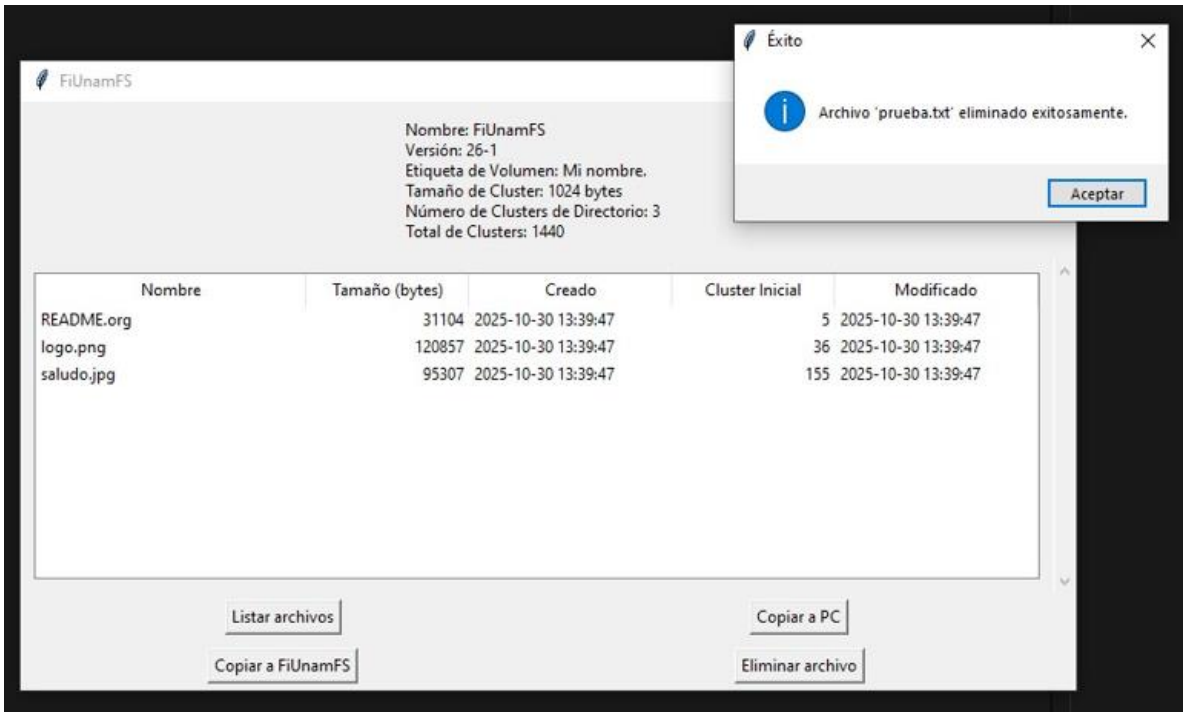
4. Aparece una ventana de confirmación preguntando si realmente se desea borrar el archivo.



5. Si se confirma, el programa lanza un hilo que:
   - Localiza la entrada de directorio correspondiente.
   - La marca como vacía (tipo -, nombre de relleno) y pone a cero tamaño y clúster inicial.

- Rellena con ceros los clústeres de datos que ocupaba ese archivo.

6. Al terminar, se muestra un mensaje indicando que la eliminación fue exitosa.



7. Volver a pulsar “Listar archivos” para confirmar que el archivo ya no aparece en el directorio.

4. Conclusión

En este proyecto implementé un micro–sistema de archivos multihilos que trabaja sobre la imagen FiUnamFS, capaz de listar, copiar en ambos sentidos y eliminar archivos desde una interfaz gráfica sencilla. Al diseñar la lógica por bloques y combinarla con hilos, locks y condiciones, pude aplicar en la práctica los temas de sistemas de archivos y sincronización de procesos vistos en clase. El resultado es una herramienta funcional y comprensible que sirve tanto para manipular la imagen de disco como para ilustrar el funcionamiento interno de un sistema de archivos contiguo bajo un entorno concurrente.