Universidad Nacional Autonoma de México

Sistemas Operativos

Profesor: Dr. Gunnar Eyal Wolf Iszaevich

PROYECTO 02

(Micro) sistema de archivos multihilos

Sierra García Mariana
Tapia García Andrés

Grupo: 08

Facultad de Ingeniería

Ciudad Universitaria, 18 de Noviembre del 2025

Índice

1. Descripción

El proyecto consiste en desarrollar un programa micro-sistema de archivos multihilos basado en las especificaciones de la imagen FiUnamFS, proporcionada por el profesor Gunnar Wolf, es un sistema de archivos plano diseñado para simular un volumen tipo *diskete* dentro de un archivo de 1.44 MB.

1.1. ¿Qué caracteristicas debe de tener?

1. **Listar contenido**: Ver directorio de la imagen con sus metadatos
2. **Copiar desde el FS**: El programa debe de tener la capacidad de copiar archivos de la imagen hacia el equipo
3. **Copiar hacia el FS**: El programa debe tener la capacidad de copar archivos de tu equipo a la imagen
4. **Eliminar archivos**: Eliminar de forma segura archivos con limpieza de clusters
5. **Sincronización multihilo**: Comunicación entre GUI y backend mediante mecanismos de sincronización (Lock y Condition)

2. Requisitos y Entorno

2.1. Entorno de Desarrollo

- **Lenguaje**: Python 3.8 o superior
- **Sistema Operativo**: Linux, macOS o Windows
- **Módulos requeridos**: Todos son parte de la biblioteca estándar de Python
  - `tkinter` - Interfaz gráfica (incluido en Python)
  - `threading` - Manejo de hilos
  - `struct` - Manipulación de datos binarios
  - `os`, `math`, `datetime` - Utilidades estándar

2.2. Instalación y Ejecución

2.2.1. Requisitos previos

Comprobar que tkinter esté instalado:

```
sudo dnf install python3-tkinter # Repositorio de fedora
```

2.2.2. Ejecutar el programa

```
# Metodo 1: Ejecucion directa
$ python3 fiunamfs.py

# Metodo 2: Como ejecutable
$ chmod +x fiunamfs.py
./fiunamfs.py
```

Nota: La imagen `fiunamfs.img` debe estar en el mismo directorio que el script, o se puede seleccionar manualmente al iniciar.

3. Especificaciones de FiUnamFS

3.1. Estructura General del Sistema

- **Tamaño total**: 1440 KB (1,474,560 bytes)
- **Cluster**: 1024 bytes (2 sectores de 512 bytes)
- **Superbloque**: Cluster 0
- **Directorio**: Clusters 1-4 (máximo 64 entradas)
- **Área de datos**: Clusters 5-1439
- **Asignación**: Contigua (archivos en clusters consecutivos)

3.2. Superbloque (Cluster 0)

Bytes	Campo	Valor
0-8	Nombre del FS	`FiUnamFS` (validación obligatoria)
10-14	Versión	`26-1` (advertencia si difiere)
20-35	Etiqueta del volumen	16 bytes ASCII
40-43	Tamaño de cluster	1024 (little-endian 32-bit)
45-48	Clusters de directorio	4 (little-endian 32-bit)
50-53	Total de clusters	1440 (little-endian 32-bit)

3.3. Entrada de Directorio (64 bytes)

La estructura utilizada es: `ENTRY_STRUCT = struct.Struct(''<c15sII14s14s12x'')`

Bytes	Campo	Descripción
0	Tipo	'.' = ocupado, '-' = libre
1-15	Nombre	ASCII, máximo 14 caracteres útiles
16-19	Cluster inicial	Entero 32-bit little-endian
20-23	Tamaño	Bytes (32-bit little-endian)
24-37	Fecha creación	AAAAMMDDHHMMSS (14 bytes)
38-51	Fecha modificación	AAAAMMDDHHMMSS (14 bytes)
52-63	Reservado	12 bytes padding

4. Arquitectura e Implementación

4.1. Estrategia General

El proyecto se divide en dos clases:

1. **FiUnamFS**: Tiene las funciones que manejan el sistema de archivos, operaciones y comprobaciones
2. **FiUnamFSGUI**: Tiene las funciones que generan la interfaz gráfica con Tkinter, botones y coordinación de hilos

4.2. Definición de Constantes

Son constantes que nos van a servir para el manejo de la imagen (Valores que son proporcionados en las especificaciones del proyecto dentro del README.md)

```
CLUSTER_SIZE = 1024              # Tamano de cada cluster
TOTAL_CLUSTERS = 1440            # Total de clusters en imagen
DIR_CLUSTER_START = 1            # Inicio del directorio
DIR_CLUSTER_END = 4              # Fin del directorio
ENTRY_SIZE = 64                  # Tamano de entrada
ENTRIES_PER_CLUSTER = 16         # Entradas por cluster

# Estructura binaria para entradas del directorio
ENTRY_STRUCT = struct.Struct("<c15sII14s14s12x")
```

4.3. Clase FiUnamFS ¿Qué hace cada método?

4.3.1. Constructor

```
def __init__(self, disk_path):
    self.disk = disk_path
    self.lock = threading.Lock()     # Proteccion de concurrencia
    self.archivos = None             # Cache del directorio
    if not os.path.exists(self.disk): # Verifica que el disco este en la
        misma carpeta
        raise FileNotFoundError(f"No se encontró: {self.disk}")
```

4.3.2. leer_superbloque()

Lee y valida el superbloque desde los primeros 54 bytes de la imagen. Verifica que el nombre sea `FiUnamFS` y advierte si la versión difiere de `26-1`, aqui pusimos una advertencia en vez de un error, por que el disco realmente **tiene la version 26-1, pero, en los requerimentos dice que es la 26-2**.

4.3.3. enlistar_directorio(mostrar_vacias=False)

Recorre clusters 1-4 leyendo entradas de 64 bytes. Decodifica campos binarios usando `ENTRY_STRUCT.unpack()`. Una entrada está vacía si su tipo es '-' o el nombre es cadena vacía/-puntos. Retorna lista de diccionarios con metadatos y mantiene caché en `self.archivos`.

4.3.4. hay_hueco_contiguo(tamaño)

Busca secuencia de clusters contiguos completamente en ceros. Calcula clusters necesarios con `math.ceil(tamaño/CLUSTER_SIZE)`. Recorre desde cluster 5 hasta el final validando que cada cluster sea todo ceros. Retorna índice inicial o `None`.

4.3.5. copiar_desde(nombre_archivo, destino)

Localiza archivo en caché, calcula offset (`cluster * CLUSTER_SIZE`), lee `tamaño` bytes y escribe en archivo de destino. Si destino es directorio, construye ruta completa con nombre original.

4.3.6. copiar_hacia(ruta_local)

1. Valida existencia, nombre ASCII y longitud ≤ 14 caracteres
2. Verifica que no exista archivo con mismo nombre
3. Busca espacio contiguo con `hay_hueco_contiguo()`
4. Localiza entrada libre en directorio (tipo '-')
5. Crea timestamps AAAAMMDDHHMMSS con `datetime.now()`
6. Escribe entrada con `ENTRY_STRUCT.pack()`
7. Copia contenido chunk por chunk, rellenando último cluster con zeros
8. Invalida caché y notifica GUI

4.3.7. eliminar(nombre)

Marca entrada como vacía (tipo='-', nombre='——...', tamaño=0, cluster=0, fechas='00000000000000'). Sobrescribe clusters ocupados con `b'\x00' * CLUSTER_SIZE`. Invalida caché y notifica cambios.

4.4. Clase FiUnamFSGUI ¿Qué hace cada método?

4.4.1. Constructor

```
def __init__(self, root):
    self.root = root
    self.root.title("FiUnamFS v26-1 - Sistema de Archivos")
    self.root.geometry("900x600")
    self.fs = None
    self.monitor_thread = None

    # Construir widgets y abrir el disco por defecto
    self.crear_widgets()
    self.abrir_disco()
```

Inicializa la logica para mostrar la interfaz y abre la información del disco para mostrarla en la interfaz

4.4.2. crear_widgets()

```
# Marco con info del sistema
    info_frame = ttk.LabelFrame(
        self.root, text="Informacion del Sistema", padding=10
    )
    info_frame.pack(fill=tk.X, padx=10, pady=5)
    ...
```

Empieza a crear la interfaz grafica sección por seccion

```
 # Configurar encabezados y anchos
    self.tree.heading("nombre", text="Nombre")
    self.tree.heading("tamaño", text="Tamaño")
    self.tree.heading("cluster", text="Cluster")
    self.tree.heading("creado", text="Creado")
    self.tree.heading("modificado", text="Modificado")

    self.tree.column("nombre", width=150)
    self.tree.column("tamaño", width=100, anchor=tk.E)
    self.tree.column("cluster", width=80, anchor=tk.E)
    self.tree.column("creado", width=180)
    self.tree.column("modificado", width=180)

    # Empaquetar treeview y conectar scrollbar
    self.tree.pack(fill=tk.BOTH, expand=True)
    scrollbar.config(command=self.tree.yview)

    # Botonera con acciones
    btn_frame = ttk.Frame(self.root, padding=10)
    btn_frame.pack(fill=tk.X, padx=10, pady=5)
```

aqui crea los encabezados del directorio, mete la información a la tabla, y crea los botones

```
ttk.Button(btn_frame, text="Actualizar", command=self.actualizar_lista).
    pack(
        side=tk.LEFT, padx=5
    )
    ttk.Button(
        btn_frame, text="Copiar desde FS", command=self.copiar_desde_fs
    ).pack(side=tk.LEFT, padx=5)
    ttk.Button(
        btn_frame, text="Copiar hacia FS", command=self.copiar_hacia_fs
    ).pack(side=tk.LEFT, padx=5)
    ttk.Button(btn_frame, text="Eliminar", command=self.eliminar_archivo
        ).pack(
        side=tk.LEFT, padx=5
    )
    ttk.Button(
        btn_frame, text="Ver Superbloque", command=self.
            mostrar_superbloque
    ).pack(side=tk.LEFT, padx=5)
```

Esta es una de las partes mas fundamentales de la clase gui, ya que es la que conecta las funciones de la clase `FiUnamFSGUI` que a su vez llaman a las funciones de la clase `FiUnamFS` para realizar las operaciones que el usuario quiere.

4.4.3. abrir_disco()

Primero corrobora si es que el disco esta en la misma carpeta que el codigo, si no es asi, le sale una ventana en la que le preguntara si quiere ubicar el disco

```
def abrir_disco(self):
    # Ruta por defecto: fiunamfs.img junto al script
    disk_path = os.path.join(os.path.dirname(__file__), "fiunamfs.img")

    # Si no existe el archivo por defecto, preguntamos al usuario
    if not os.path.exists(disk_path):
        respuesta = messagebox.askyesno(
            "Disco no encontrado",
            "No se encontro 'fiunamfs.img' en el directorio actual.\n\n"
            "Desea seleccionar una imagen manualmente?",
        )
        if respuesta:
            disk_path = filedialog.askopenfilename(
                title="Seleccionar imagen FiUnamFS",
                filetypes=[("Imagenes de disco", "*.img"), ("Todos", "
                    *.*")],
            )
            if not disk_path:
                messagebox.showerror("Error", "No se selecciono ningun
                    archivo")
                self.root.quit()
                return
        else:
            self.root.quit()
            return
```

Si lo encuentra de forma satisfactoria creara el objeto fs que es una instancia de la clase FiUnamFS, con la cual ocupara el metodo de extttleer_superbloque() para guardar la iformación en un diccionario

```
self.fs = FiUnamFS(disk_path)
        info = self.fs.leer_superbloque()

        # Mostrar información condensada en la etiqueta superior
        self.info_label.config(
            text=f"{info['Nombre']} v{info['Version']} | "
            f"Etiqueta: {info['Etiqueta']} | "
            f"Clusters: {info['Total_Clusters']} | "
            f"Tamano Cluster: {info['Tamano_Cluster']} bytes"
        )

        # Arrancar hilo que espera notificaciones de cambios
        self.monitor_thread = threading.Thread(
            target=self.monitor_cambios, daemon=True
        )
        self.monitor_thread.start()

        # Lista inicial de archivos
        self.actualizar_lista()
```

4.4.4. actualizar_lista()

Limpia la tabla que ya teniamos en la interfaz para después volver a llenarla volviendo a llamar al metodo `enlistar_directorio()` del objeto self.fs (instancia de la clase FiUnamFS)

4.4.5. copiar_desde_fs()

1. Seleccionar archivo en tabla
2. Clic en `Copiar desde FS`
3. Elegir carpeta destino
4. Manda esa esa información a el objeto self.fs con la función `copiar_desde()`
5. Si existe algun error le salen ventanas de error
6. Archivo se extrae con nombre original

4.4.6. copiar_hacia_fs()

1. Clic en `Copiar hacia FS`
2. Selecciónas el archivo
3. llama al metodo `copiar_hacia()` del objeto self.fs (instancia de FiUnamFS) y le manda el archivo que seleccionaste
4. Se realizan todas las validaciones dentro de esta funcion, como:
   a) Nombre valido de 14 o menos caracteres ASCII
   b) Buscar espacio contiguo de la funcion `hay_hueco_contiguo()`
   c) copia los archivos con nuevos metadatos
5. Actualiza la tabla con `self.actualizar_lista()`

4.4.7. eliminar()

1. Seleccionar archivo en tabla
2. Clic en `Eliminar`
3. Confirmar operación
4. Se llama al metodo `eliminar()` del objeto self.fs (instancia de FiUnamFS)
5. Actualiza la tabla con `self.actualizar_lista()`

4.5. mostrar_superbloque()

```
info = self.fs.leer_superbloque()
texto = "\n".join([f"{k}: {v}" for k, v in info.items()])
messagebox.showinfo("Informacion del Superbloque", texto)
```

Guarda la información en un diccionario y lo muestra

5. Sincronización Multihilo

5.1. Variable de Condición Global

```
# Variable global para notificacion entre hilos
VCListFiles = threading.Condition()
```

Esta `Condition` nos permite hacer que el hilo monitor se bloquee hasta recibir notificación de cambios en el sistema de archivos.

5.2. Mutex

```
self.lock = threading.Lock()
```

Esta linea nos ayuda a evitar condiciones de carrera, para proteger el disco cuando realizamos operaciones criticas.

5.3. Hilos Principales

1. **Hilo Principal (GUI)**: Maneja eventos de interfaz y ejecuta operaciones solicitadas por el usuario
2. **Hilo Monitor (daemon)**: Detecta cambios y actualiza la tabla automáticamente sin intervención del usuario

5.4. Flujo

1. La interfaz detecta que el usuario apreciono un boton (copiar/eliminar)
2. Se evitan las condiciones de carrera en los hilos

```
with self.lock:
    # Operacion critica sobre disco
    # ...
    self.archivos = None  # Invalidar cache
```

3. Al finalizar, notifica hilo monitor:

```
with VCListFiles:
    VCListFiles.notify_all()
```

4. Hilo monitor despierta y programa actualización en hilo principal:

```
def monitor_cambios(self):
    while True:
        with VCListFiles:
            VCListFiles.wait()  # Espera bloqueante
            # Actualizar en hilo GUI (thread-safe)
            self.root.after(100, self.actualizar_lista)
```

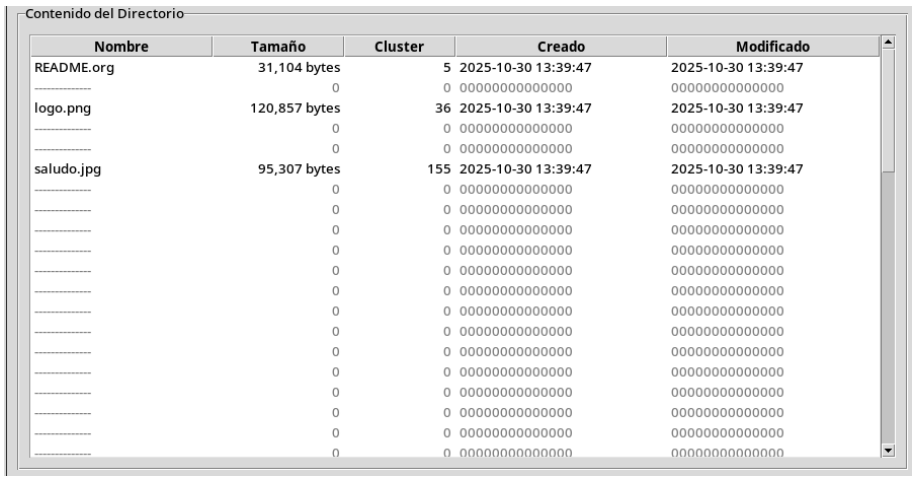
5. La tabla se refresca automáticamente mostrando cambios

6. Interfaz Gráfica

Los componentes principales de la interfaz Gráfica son:

- **Panel de información que esta en la parte superior**: Muestra nombre, versión, etiqueta y tamaño de cluster

- **Tabla de archivos**: `ttk.Treeview` con columnas (Nombre, Tamaño, Cluster, Creado, Modificado)



- **Botones**: Actualizar, Copiar desde FS, Copiar hacia FS, Eliminar, Ver Superbloque

7. Ejemplo de Uso

7.1. Contenido Inicial de la Imagen

La imagen `fiunamfs.img` proporcionada contiene:

Figura 1: Listado inicial de archivos en la imagen

El archivo README.org contiene las instrucciones originales del proyecto.

7.2. Visualizar Superbloque

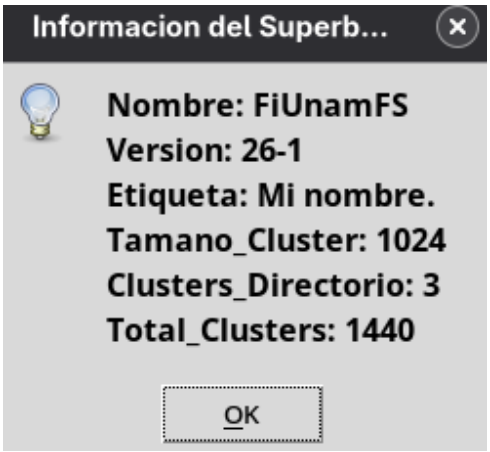
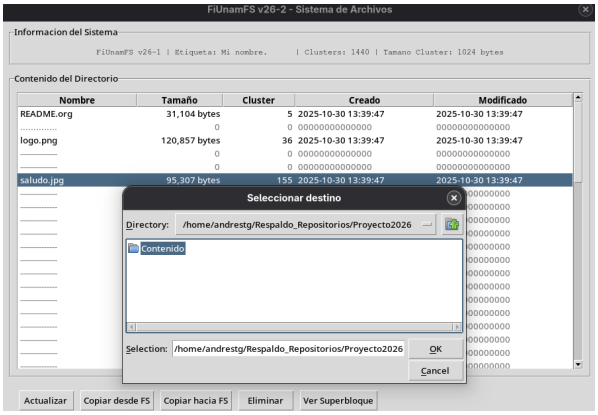


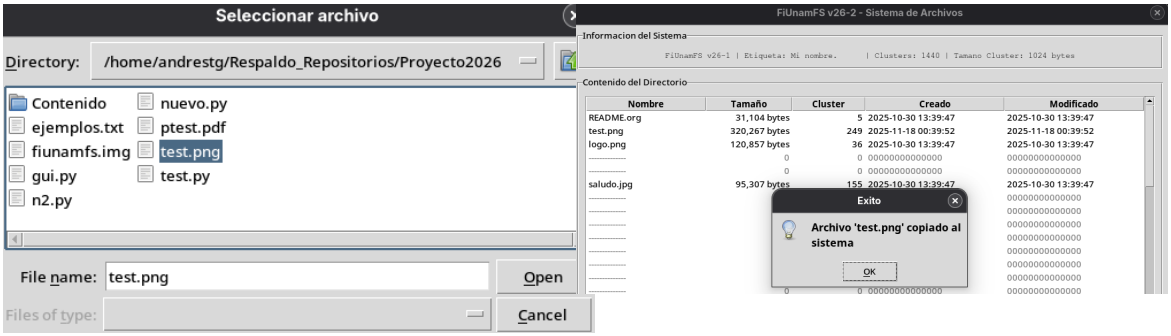
Figura 2: Información del superbloque

7.3. Copiar Archivo del File System FiUNAMFS

1. Seleccionar archivo en lista
2. Clic en “Copiar desde FS”
3. Elegir carpeta destino
4. Confirmar con [OK]

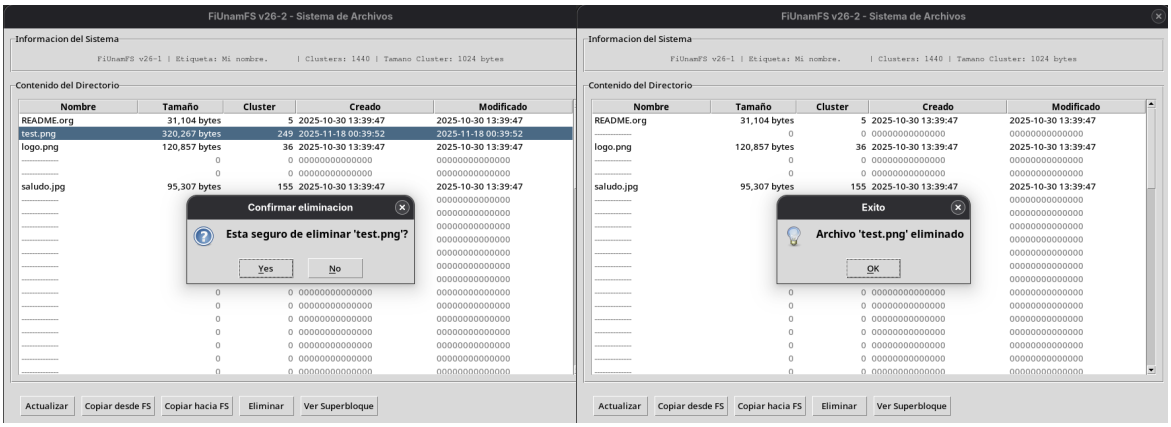


7.4. Copiar archivo del equipo a FiUNAMFS



Proceso: Clic en “Copiar hacia FS” → Seleccionar archivo → Verificar en listado

7.5. Eliminar Archivo de la imagen



Seleccionar archivo → Clic en “Eliminar” → Confirmar → Verificar remoción

8. Errores y el manejo de la implementación

Error	**Manejo**
Imagen no encontrada	Diálogo para seleccionar manualmente o salir
Superbloque inválido	`ValueError` si nombre ≠ “FiUnamFS”
Versión incorrecta	Advertencia en consola, continúa ejecución
Nombre no-ASCII	`ValueError` al intentar copiar hacia FS
Nombre muy largo (>14)	`ValueError` rechaza operación
Archivo ya existe	`ValueError` antes de buscar espacio
Espacio insuficiente	`RuntimeError` si no hay clusters contiguos
Directorio lleno	`RuntimeError` si no hay entradas libres
Archivo no encontrado	`FileNotFoundError` al copiar/eliminar

8.1. Mensajes al Usuario

Todos los errores se comunican mediante `messagebox.showerror()` con descripción del problema. El resto de operaciones se manejan con `messagebox.showinfo()`.