UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMAS OPERATIVOS

PROYECTO: (MICRO) SISTEMA DE ARCHIVOS MULTIHILOS

MAESTRO: ING. GUNNAR EYAL WOLF ISZAEVICH

GRUPO: 06

INTEGRANTES:

GONZÁLEZ LÓPEZ DAVID

REYES ROMERO LUIS FERNANDO

SEMESTRE: 2024-2

FECHA DE ENTREGA: 19 DE MAYO DE 2024

Índice

1. Introducción

En el presente escrito se documentará y ahondará la implementación de todos los aspectos requeridos para la realización del proyecto final de la materia de Sistemas Operativos denominado "(Micro) sistema de archivos multihilos". De igual manera, se fundamentan las decisiones adoptadas sobre soluciones presentadas para cada aspecto relevante del proyecto.

2. Implementación

Para el desarrollo e implementación del presente proyecto se tomaron en cuenta los siguientes 4 aspectos planteados por el profesor:

- Listar los contenidos del directorio
- Copiar uno de los archivos de dentro del FiUnamFS hacia tu sistema
- Copiar un archivo de tu computadora hacia tu FiUnamFS
- Eliminar un archivo del FiUnamFS

Aunado a los aspectos anteriores, se añadió la inclusión de una interfaz de usuario para facilitar el manejo del propio programa, además de que, su uso permite visualizar de mejor manera el manejo de hilos que será explicado posteriormente en el documento.
Previo a abordar los aspectos relevantes del proyecto, hay que mencionar un par de aspectos, primeramente, el programa está realizado en el lenguaje de programación *Python,* esto dadas las facilidades que el mismo proporciona tanto apra programar como para poder manejar hilos de operación, de igual manera, se debe saber que el micro sistema de archivos a emplear es el proporcionado por el profesor, siendo llamado "fiunamfs.img", el cual poseé 3 archivos; un .png, un .jpg y un .org.
Así pues, a continuación se explicarán a detalle todas las implementaciones realizadas para la realización del proyecto.

2.1 Listado de contenidos del directorio

Para la implementación de este aspecto en específico, se tuvo que definir un manejo de los contenidos del directorio en base a las especificaciones proporcionadas por el profesor, resultando primero en la definición de las siguientes constantes a emplear a lo largo del programa.

```
# Definiendo constantes a emplear para el manejo correcto del sistema
SUPERBLOQUE_SIZE = 64
ENTRADA_DIRECTORIO_SIZE = 64
NUM_SECTORES_ENTRADA_DIRECTORIO = 1
CLUSTER_SIZE = 512
NUM_SECTORES_POR_CLUSTER = 4
DIRECTORIO_CLUSTER_INICIAL = 1
NUMERO_SECTORES_SUPERBLOQUE = 1
NUMERO_SECTORES_ENTRADA_DIRECTORIO = 4
NUMERO_ENTRADAS_DIRECTORIO = 16
FIUNAMFS_SIGNATURE = b"FiUnamFS"
VERSION_IMPLEMENTACION = "24-2"
NOMBRE_FIUNAMFS_IMG = "fiunamfs.img"
NOMBRE_ARCHIVO_DIRECTORIO = "directorio"
```

Como puede notarse, se cubren los tamaños del cluster, el superbloque, el número de sectores, las definiciones de las entradas del directorio y el número de sectores.
Entonces, se hace uso de una clase llamada “EntradaDirectorio” con el fin de definir la información propia de cada archivo del directorio, además de hacer uso de métodos para el manejo de los bytes de dichos archivos, quedando el constructor de la clase de la siguiente manera.

```python
#Constructor de la entrada para definir la información del archivo
def __init__(self, tipo_archivo, nombre_archivo, tamano_archivo, cluster_inicial, fecha_creacion, fecha_modificacion):
    self.tipo_archivo = tipo_archivo
    self.nombre_archivo = nombre_archivo
    self.tamano_archivo = tamano_archivo
    self.cluster_inicial = cluster_inicial
    self.fecha_creacion = fecha_creacion
    self.fecha_modificacion = fecha_modificacion
```

Posteriormente, el programa hace uso de dos métodos para el manejo de los datos pertenecientes a cada archivo en base a las especificaciones planteadas por el profesor, siendo así los métodos:

- “to_bytes”: El método adecua tanto el tipo de archivo, su fecha de creación, fecha de modificación y su nombre en “ascii”, esto dadas las especificaciones ya mencionadas, siendo conseguido esto por medio del método “encode”. Posteriormente, haciendo uso de “struct.pack” para convertir valores enteros a su representación en bytes, esto siguiendo un cierto formato en el que, la cadena de formato “<I” le dice a struct.pack cómo debe convertir los datos. Así pues, el símbolo de menor que “<”, indica que los datos deben ser empaquetados en "little-endian" (orden de byte pequeño), mientras que, la “I” mayúscula indica que el valor a empaquetar es un entero sin signo de 4 bytes (32 bits). Finalmente, se tiene el segundo argumento de struct.pack, el cual es el valor que se desea convertir a bytes, siendo en este caso, “self.tamano_archivo” y “self.cluster_inicial”, mismos que son enteros que se están convirtiendo a una secuencia de 4 bytes cada uno.

```python
def to_bytes(self):
    # Convertir la entrada del directorio a bytes
    tipo_archivo_bytes = self.tipo_archivo.encode("ascii")
    nombre_archivo_bytes = self.nombre_archivo.encode("ascii").ljust(15, b"\0")
    tamano_archivo_bytes = struct.pack("<I", self.tamano_archivo)
    cluster_inicial_bytes = struct.pack("<I", self.cluster_inicial)
    fecha_creacion_bytes = self.fecha_creacion.encode("ascii")
    fecha_modificacion_bytes = self.fecha_modificacion.encode("ascii")
    return tipo_archivo_bytes + nombre_archivo_bytes + tamano_archivo_bytes + cluster_inicial_bytes + fecha_creacion_bytes + fecha_modificacion_bytes
```

- “from_bytes”: El método hace uso de un “@classmethod” para así poder trabajar en la misma instancia de la declaración de la entrada del directorio, usando igualmente “cls”, para que, al mismo tiempo que se realizan las conversiones anteriormente aclaradas, se haga el análisis del tamaño de los datos de entrada, y las validaciones pertinentes sobre las longitudes de cada uno de los elementos que conforman al propio archivo de entrada en el directorio, esto siguiendo nuevamente el manejo de “little-endian”.

```
@classmethod
def from_bytes(cls, data):
    if len(data) < 52:
        print(f"Longitud de los datos: {len(data)}")
        raise ValueError("Los datos de entrada no tienen la longitud esperada.")
    tipo_archivo = data[0:1]
    nombre_archivo = data[1:16].rstrip(b"\0").decode("ascii")
    tamano_archivo = struct.unpack("<I", data[16:20])[0]
    cluster_inicial = struct.unpack("<I", data[20:24])[0]
    fecha_creacion = data[24:38].decode("ascii")
    fecha_modificacion = data[38:52].decode("ascii", errors="ignore")
    return cls(tipo_archivo, nombre_archivo, tamano_archivo, cluster_inicial, fecha_creacion, fecha_modificacion)
```

- “leer_superbloque”: Se define finalmente como método adicional, la lectura del superbloque, esto respetando el tamaño de los bytes y su correspondencia con respecto a las especificaciones planteadas por el profesor.

```
# Definiendo métodos auxiliares
def leer_superbloque():
    with open(NOMBRE_FIUNAMFS_IMG, "rb") as img_file:
        superbloque_data = img_file.read(SUPERBLOQUE_SIZE)
        if superbloque_data[:8] != FIUNAMFS_SIGNATURE:
            raise Exception("No es un sistema de archivos FiUnamFS válido.")
        if superbloque_data[10:14].decode("ascii") != VERSION_IMPLEMENTACION:
            raise Exception("Versión de implementación incorrecta.")
        etiqueta_volumen = superbloque_data[20:36].decode("ascii").strip()
        tamano_cluster = struct.unpack("<I", superbloque_data[40:44])[0]
        num_clusters_directorio = struct.unpack("<I", superbloque_data[45:49])[0]
        num_clusters_total = struct.unpack("<I", superbloque_data[50:54])[0]
        return etiqueta_volumen, tamano_cluster, num_clusters_directorio, num_clusters_total
```

Antes de seguir con el método implementado para realizar el listado de los elementos en el directorio, hay que mencionar que el programa funciona en base a hilos, los cuales se encargan cada uno de hacer una operación individual del sistema, siendo éstas el listado, copiado y eliminación de archivos del propio micro sistema de archivos. El manejo de hilos se hace por medio de la biblioteca “threading” de Python.

Así pues, como puede observarse el manejo de los hilos se realiza por medio de un aclase “HiloOperacion” la cual se crea en base al sistema de archivos .img utilizado, un indicador de qué operación realizar en forma de cadena, y los argumentos de dicha operación, siendo para este caso el de la operación “listar”. El manejo de los hilos del programa será esclarecido en su totalidad en el apartado de la “Interfaz de Usuario” del presente texto.

```
class HiloOperacion(threading.Thread):
    def __init__(self, operacion, img_file, *args):
        threading.Thread.__init__(self)
        self.operacion = operacion
        self.img_file = img_file
        self.args = args

    #Elección de operación a realizar por el hilo
    def run(self):
        if self.operacion == "listar":
            listar_contenido_directorio(self.img_file)
        elif self.operacion == "copiar_desde":
            if len(self.args) == 1:
                copiar_desde_fiunamfs(self.img_file, *self.args)
            else:
                print("Número incorrecto de argumentos para la operación 'copiar_desde'.")
        elif self.operacion == "copiar_hacia":
            if len(self.args) == 2:
                copiar_hacia_fiunamfs(self.img_file, *self.args)
            else:
                print("Número incorrecto de argumentos para la operación 'copiar_hacia'.")

        elif self.operacion == "eliminar":
            nombre_archivo_fiunamfs = self.args[0]
            eliminar_archivo_fiunamfs(self.img_file, nombre_archivo_fiunamfs)
            print(f"El archivo {nombre_archivo_fiunamfs} ha sido eliminado.")
```

Teniendo así, el método “listar_contenido_directorio” el cual se encarga de realizar la búsqueda y lectura de cada archivo dependiendo de su clúster. Igualmente, se asegura que el espacio a buscar no sea vacío como puede observarse. Durante este proceso, toda la información del archivo se va almacenando en una variable “entrada_data” la cual después es procesada de bytes a chars por medio del método “from_bytes” para así poder hacer el listado del contenido del sistema de archivos en lenguaje humano.

```
def listar_contenido_directorio(img_file):
    contenido = ""
    img_file.seek(CLUSTER_SIZE * NUM_SECTORES_POR_CLUSTER * DIRECTORIO_CLUSTER_INICIAL)
    directorio_data = img_file.read(CLUSTER_SIZE * NUM_SECTORES_ENTRADA_DIRECTORIO * NUMERO_ENTRADAS_DIRECTORIO)
    for i in range(NUMERO_ENTRADAS_DIRECTORIO):
        entrada_data = directorio_data[i * ENTRADA_DIRECTORIO_SIZE: (i + 1) * ENTRADA_DIRECTORIO_SIZE]
        # Verificar si la línea contiene caracteres '#' o está vacía
        if b'#' in entrada_data or not entrada_data.strip():
            continue  # Saltar esta entrada si contiene '#' o está vacía
        if len(entrada_data) < 52:
            continue  # Saltar esta entrada si la longitud no es la esperada
        entrada = EntradaDirectorio.from_bytes(entrada_data)
        if entrada.tipo_archivo != "-" and entrada.tipo_archivo != "/":
            contenido += f"Nombre: {entrada.nombre_archivo}, Tamaño: {entrada.tamano_archivo}, Cluster inicial: {entrada.cluster_inicial}\n"
    return contenido
```

2.2 Copia de archivos desde el micro-sistema de archivos al sistema local

Para la implementación de este requerimiento, primeramente se hace uso de la opción "copiar_hacia" de la clase "HiloOperacion".

```
class HiloOperacion(threading.Thread):
    def __init__(self, operacion, img_file, *args):
        threading.Thread.__init__(self)
        self.operacion = operacion
        self.img_file = img_file
        self.args = args

    #Elección de operación a realizar por el hilo
    def run(self):
        if self.operacion == "listar":
            listar_contenido_directorio(self.img_file)
        elif self.operacion == "copiar_desde":
            if len(self.args) == 1:
                copiar_desde_fiunamfs(self.img_file, *self.args)
            else:
                print("Número incorrecto de argumentos para la operación 'copiar_desde'.")
        elif self.operacion == "copiar_hacia":
            if len(self.args) == 2:
                copiar_hacia_fiunamfs(self.img_file, *self.args)
            else:
                print("Número incorrecto de argumentos para la operación 'copiar_hacia'.")

        elif self.operacion == "eliminar":
            nombre_archivo_fiunamfs = self.args[0]
            eliminar_archivo_fiunamfs(self.img_file, nombre_archivo_fiunamfs)
            print(f"El archivo {nombre_archivo_fiunamfs} ha sido eliminado.")
```

Entonces, con el método "copiar_desde_fiunamfs" se hace uso de una carpeta adicional llamada "LocalSO", la cual puede ser creada previamente, y en caso de que no se encuentre, el programa se encarga de crearla, esto debido a que el sistema local suele tener problemas con los permisos de acceso y creación de archivos con respecto a la carpeta propia dónde se ubique el proyecto, caso que se soluciona con esta solución. Con lo anterior esclarecido, el programa procede a definir la ruta donde se ubicará el archivo a copiar, siendo el caso de la carpeta "LocalSO".

```
# Método para realizar el copiado de un archivo del sistema de archivos a una carpeta local del sistema propio
def copiar_desde_fiunamfs(img_file, nombre_archivo_fiunamfs):
    # Ruta de la carpeta "LocalSO" dentro de la carpeta actual del proyecto
    carpeta_localSO = os.path.join(os.getcwd(), "LocalSO")

    # Crear la carpeta "LocalSO" si no existe
    if not os.path.exists(carpeta_localSO):
        os.makedirs(carpeta_localSO)

    # Ruta del archivo local en la carpeta "LocalSO"
    nombre_archivo_local = os.path.join(carpeta_localSO, nombre_archivo_fiunamfs)
```

Tras lo anterior, al programa solo le queda realizar la búsqueda del archivo a copiar desde el micro sistema de archivos, esto respetando los tamaño en bytes de los clúster, sectores y los números de entradas del directorio. La identificación de que el archivo coincida en nombre

dentro de la búsqueda se hace gracias al método “from_bytes”, el cual como ya se ha visto,

```
with open(nombre_archivo_local, "wb") as local_file:
    img_file.seek(CLUSTER_SIZE * NUM_SECTORES_POR_CLUSTER * DIRECTORIO_CLUSTER_INICIAL)
    for i in range(NUMERO_ENTRADAS_DIRECTORIO):
        entrada_data = img_file.read(ENTRADA_DIRECTORIO_SIZE)
        entrada = EntradaDirectorio.from_bytes(entrada_data)
        if os.path.basename(entrada.nombre_archivo).strip() == nombre_archivo_fiunamfs.strip():
            img_file.seek(entrada.cluster_inicial * CLUSTER_SIZE * NUM_SECTORES_POR_CLUSTER)
            while True:
                data = img_file.read(CLUSTER_SIZE)
                if not data:
                    break
                local_file.write(data)
            print(f"Se ha copiado {nombre_archivo_fiunamfs} a la carpeta 'LocalSO'.")
            return  # Salir del bucle una vez que se haya copiado el archivo
    print(f"No se encontró el archivo {nombre_archivo_fiunamfs} en el FiUnamFS.")
```

permite el análisis desde bytes a chars.

Finalmente, tras encontrar al archivo se hace el copiado de los datos del archivo hacia la carpeta “LocalSO” hasta que se encuentre una dirección vacía.

2.3 Eliminado de archivo del micro-sistema de archivos

Para la implementación de este requerimiento, se hace uso de la opción “eliminar” de la clase “HiloOperacion”.

```
class HiloOperacion(threading.Thread):
    def __init__(self, operacion, img_file, *args):
        threading.Thread.__init__(self)
        self.operacion = operacion
        self.img_file = img_file
        self.args = args

    #Elección de operación a realizar por el hilo
    def run(self):
        if self.operacion == "listar":
            listar_contenido_directorio(self.img_file)
        elif self.operacion == "copiar_desde":
            if len(self.args) == 1:
                copiar_desde_fiunamfs(self.img_file, *self.args)
            else:
                print("Número incorrecto de argumentos para la operación 'copiar_desde'.")
        elif self.operacion == "copiar_hacia":
            if len(self.args) == 2:
                copiar_hacia_fiunamfs(self.img_file, *self.args)
            else:
                print("Número incorrecto de argumentos para la operación 'copiar_hacia'.")

        elif self.operacion == "eliminar":
            nombre_archivo_fiunamfs = self.args[0]
            eliminar_archivo_fiunamfs(self.img_file, nombre_archivo_fiunamfs)
            print(f"El archivo {nombre_archivo_fiunamfs} ha sido eliminado.")
```

Con esto, el método “eliminar_archivo_fiunamfs”, de manera similar a la metodología anterior, hace una búsqueda del archivo por medio de su nombre y respetando las especificaciones establecidas por el profesor sobre el sistema de archivos.

```
def eliminar_archivo_fiunamfs(img_file, nombre_archivo_fiunamfs):
    img_file.seek(CLUSTER_SIZE * NUM_SECTORES_POR_CLUSTER * DIRECTORIO_CLUSTER_INICIAL)
    directorio_data = img_file.read(CLUSTER_SIZE * NUM_SECTORES_ENTRADA_DIRECTORIO * NUMERO_ENTRADAS_DIRECTORIO)

    for i in range(NUMERO_ENTRADAS_DIRECTORIO):
        entrada_data = directorio_data[i * ENTRADA_DIRECTORIO_SIZE: (i + 1) * ENTRADA_DIRECTORIO_SIZE]
        entrada = EntradaDirectorio.from_bytes(entrada_data)
        if entrada.nombre_archivo.strip() == nombre_archivo_fiunamfs:
```

Posteriormente, tras hacer la coincidencia de la entrada en el sistema de archivos por medio de su nombre gracias al método “from_bytes”, se procede a realizar el “borrado” del archivo, constando éste borrado del uso de 2 variables.

- “nueva_entrada_data”: Se encarga de construir una nueva entrada de directorio en formato binario, que representará una entrada vacía o eliminada, esto haciéndo uso inicialmente de b"/", el cual representa el tipo de archivo, además de que se está utilizando el carácter / para indicar que esta entrada ha sido ya eliminada. Tras ello se le añade b"-" * 15, constando esta parte en crear una cadena de 15 guiones (-) en formato de bytes, par posteriormente, por medio de struct.pack("<I", 0), empaquetar el valor 0 como un entero sin signo de 4 bytes, esto para representar el tamaño del archivo en la entrada eliminada.: Finalmente, se añade una cadena de 26 ceros en formato de bytes por medio de b"0" * 26, representando así la eliminación de las fechas de creación y modificación en la entrada eliminada. El resultado de este manejo de eliminación es una secuencia de 52 bytes, correspondiendo ésta a una entrada completa en el directorio.

```
# Modificar y borrar así la entrada en memoria
nueva_entrada_data = b"/" + b"-" * 15 + struct.pack("<I", 0) + struct.pack("<I", 0) + b"0" * 26
```

- “directorio_data”: Se encarga de reemplazar la entrada correspondiente en el directorio con la nueva entrada "vacía" creada en la línea anterior, esto respetando el tamaño en bytes de cada entrada de directorio.

```
directorio_data = directorio_data[:i * ENTRADA_DIRECTORIO_SIZE] + nueva_entrada_data
+ directorio_data[(i + 1) * ENTRADA_DIRECTORIO_SIZE:]
```

Tras lo anterior, y con la nueva información de “borrado” establecida, se hace una reescritura del archivo borrado en el sistema de archivos, finalizando así el proceso de eliminación.

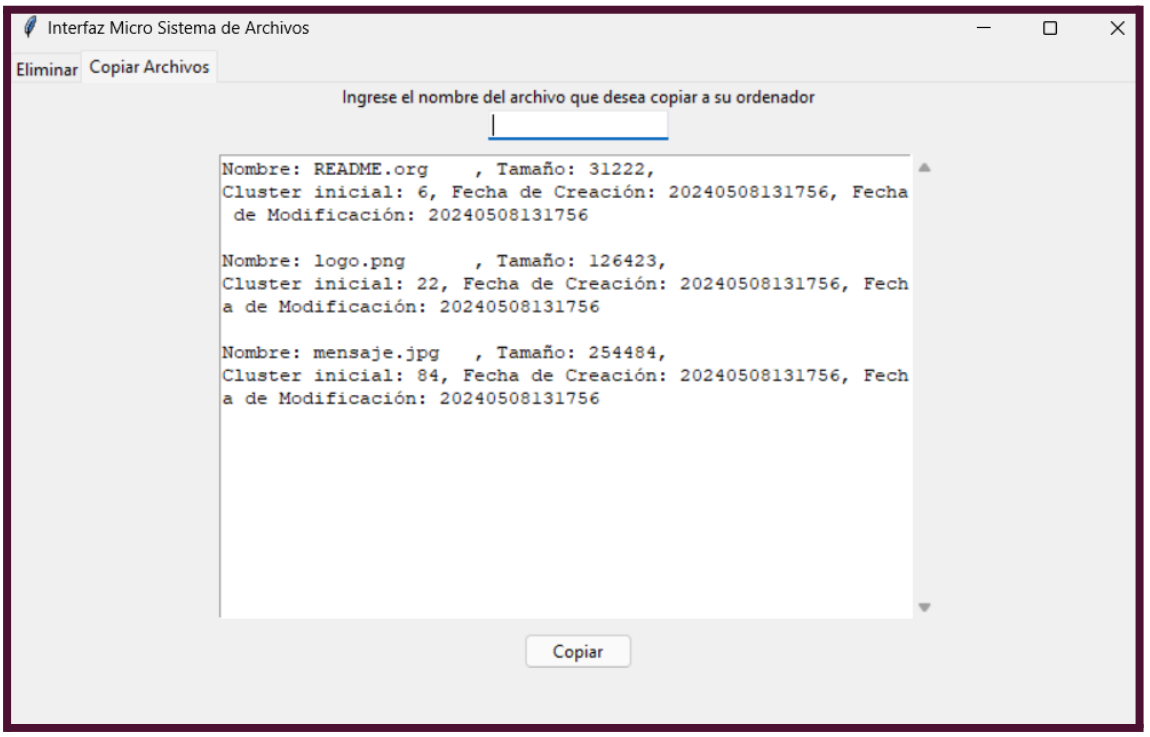
```
# Escribir el directorio modificado de nuevo en el archivo
img_file.seek(CLUSTER_SIZE * NUM_SECTORES_POR_CLUSTER * DIRECTORIO_CLUSTER_INICIAL)
img_file.write(directorio_data)

print(f"El archivo {nombre_archivo_fiunamfs} ha sido eliminado.")
return
```

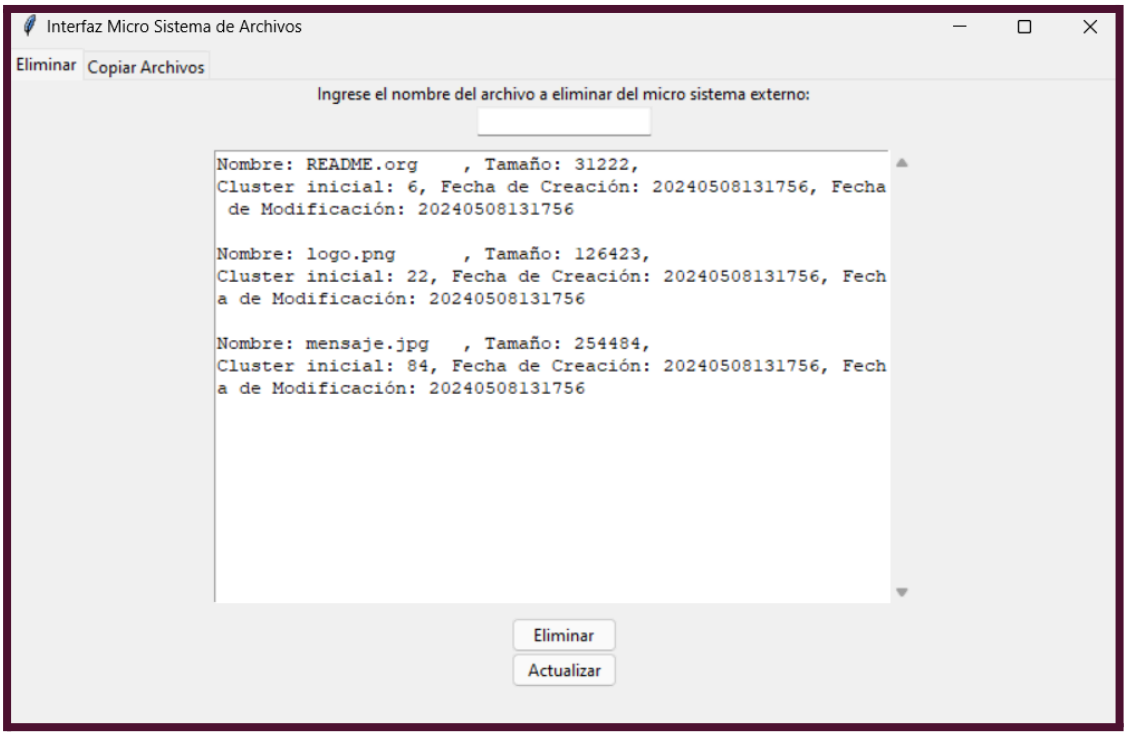
2.4 Interfaz de Usuario y Manejo de Hilos

Como ya se ha mencionado previamente en el documento, la implementación del manejo de hilos y la interfaz de usuario están estrechamente ligados, esto debido a que la propia interfaz es la que hace uso de los hilos para realizar sus operaciones. Siendo así, la interfaz resulta ser un tanto sencilla pero bastante agradable al usuario, esta misma se hizo por medio de la biblioteca "tkinter" la cual proporciona las facilidades necesarias para manejar cada operación especificada previamente.
Entonces, la implementación de la interfaz se hace con el uso de una clase "InferfazUsuario" en la cual se inicializan sus definiciones para su despliegue. Posteriormente, dentro de la misma clase se hace la inicialización de 2 hilos dentro de 2 pestañas respectivamente, siendo éstos el hilo de copiado al sistema local y el de eliminación, para así tenerlos preparados al momento de ser llamados para operar.

```
# Implementación de la interfaz de usuario
class InterfazUsuario:
    def __init__(self, root):
        self.root = root
        self.root.title("Interfaz Micro Sistema de Archivos")
        self.root.geometry("800x500")

        self.tabControl = ttk.Notebook(root)

        self.tabEliminar = ttk.Frame(self.tabControl)
        self.tabCopiarAr = ttk.Frame(self.tabControl)

        # Pestañas de eliminación y copiado
        self.tabControl.add(self.tabEliminar, text="Eliminar")
        self.tabControl.add(self.tabCopiarAr, text="Copiar Archivos")

        self.tabControl.pack(expand=1, fill="both")

        # Inicialización de los hilos de copiado y eliminación
        self.init_eliminar_tab()
        self.init_copiarAr_tab()


        # Iniciar el hilo para listar contenido en la pestaña de eliminar
        self.hilo_listar_eliminar = threading.Thread(target=self.hilo_listar_eliminar_func)
        self.hilo_listar_eliminar.start()

        # Inicia el hilo para lsitar contenido en la pestaña de copiar Archivos
        self.hilo_listar_copiar_ar = threading.Thread(target=self.hilo_listar_copiarAr_func)
        self.hilo_listar_copiar_ar.start()
```

De igual manera, en la implementación, se hizo uso de dos hilos latentes y concurrentes con respecto a los 2 anteriores, siendo los referentes al listado del contenido del sistema de archivos, esto para que el usuario tenga noción de lo que se encuentra dentro del propio micro-sistema en todo momento. Siendo así, la implementación de ambas pestañas en ejecución se ve de la siguiente manera.





Prosiguiendo con el código, se tiene la creación y definición de las pestañas de copiado y eliminación ya presentadas, implementando en ambas instancias al recuadro con el listado de las entradas del directorio.

```
# Pestaña de eliminación
def init_eliminar_tab(self):
    label = ttk.Label(self.tabEliminar, text="Ingrese el nombre del archivo a eliminar del micro sistema externo:")
    label.pack()

    self.entry_eliminar = ttk.Entry(self.tabEliminar)
    self.entry_eliminar.pack()

    self.txtListarContenidoEliminar = scrolledtext.ScrolledText(self.tabEliminar, width=60, height=20)
    self.txtListarContenidoEliminar.pack(pady=10)

    button_eliminar = ttk.Button(self.tabEliminar, text="Eliminar", command=self.eliminar_thread)
    button_eliminar.pack()

    button_actualizar = ttk.Button(self.tabEliminar, text="Actualizar", command=self.actualizar_listado_eliminar)
    button_actualizar.pack()

# Pestaña de copiado al sistema local
def init_copiarAr_tab(self):
    label = ttk.Label(self.tabCopiarAr, text="Ingrese el nombre del archivo que desea copiar a su ordenador")
    label.pack()

    self.entry_copiarAr = ttk.Entry(self.tabCopiarAr)
    self.entry_copiarAr.pack()

    self.txtListarContenidoCopiarAr = scrolledtext.ScrolledText(self.tabCopiarAr, width=60, height=20)
    self.txtListarContenidoCopiarAr.pack(pady=10)

    button_copiar = ttk.Button(self.tabCopiarAr, text="Copiar", command=self.copiarAr_thread)
    button_copiar.pack()

    button_actualizar = ttk.Button(self.tabCopiarAr, text="Actualizar", command=self.actualizar_listado_copiarAr)
    button_actualizar.pack()
```

Tras ello, se hace el manejo y funcionamiento del hilo de copiado, el cual por medio de un recuadro recibe el nombre del archivo a copiar, para que así se haga todo el proceso previamente abordado del método “copiar_desde_fiunamfs”, para que, al finalizar el proceso se mande un mensaje indicando que el copiado y pegado se realizó de manera exitosa, o en caso de que el archivo no se encuentre en el micro-sistema, se indique esto mismo por medio de otro mensaje.

```
# Hilo de copiado
def copiarAr_thread(self):
    nombre_archivo = self.entry_copiarAr.get()
    hilo_copiarAr = threading.Thread(target=self.hilo_copiarAr_func, args=(nombre_archivo,))
    hilo_copiarAr.start()

# Funcionalidad de hilo de copiado
def hilo_copiarAr_func(self, nombre_archivo):
    try:
        carpeta_localSO = os.path.join(os.getcwd(), "LocalSO")
        nombre_archivo_local = os.path.join(carpeta_localSO, nombre_archivo)
        if os.path.exists(nombre_archivo_local):
            messagebox.showwarning("Copiar", f"El archivo {nombre_archivo} ya existe en la carpeta 'LocalSO'.")
            return
        with open(NOMBRE_FIUNAMFS_IMG, "r+b") as img_file:
            copiar_desde_fiunamfs(img_file, nombre_archivo)
            messagebox.showinfo("Copiar", f"El archivo {nombre_archivo} ha sido copiado a la carpeta 'LocalSO'.")
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

Así pues, para el caso del hilo de eliminación, se sigue el mismo procedimiento pero ahora para la operación de borrado del archivo introducido en el recuadro, usando el método “eliminar_archivo_fiunamfs”.

```python
# Hilo de eliminación
def eliminar_thread(self):
    nombre_archivo = self.entry_eliminar.get()
    hilo_eliminar = threading.Thread(target=self.hilo_eliminar_func, args=(nombre_archivo,))
    hilo_eliminar.start()

#Funcionamiento del hilo de eliminación
def hilo_eliminar_func(self, nombre_archivo):
    try:
        with open(NOMBRE_FIUNAMFS_IMG, "r+b") as img_file:
            resultado = eliminar_archivo_fiunamfs(img_file, nombre_archivo)
            if resultado == "eliminado":
                messagebox.showinfo("Eliminar", f"El archivo {nombre_archivo} ha sido eliminado correctamente.")
            elif resultado == "ya_eliminado":
                messagebox.showinfo("Eliminar", f"El archivo {nombre_archivo} ya ha sido eliminado anteriormente.")
            elif resultado == "no_encontrado":
                messagebox.showwarning("Eliminar", f"El archivo {nombre_archivo} no se encontró en el directorio.")
            self.actualizar_listado_eliminar()  # Actualizar la lista en el hilo principal
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

Posteriormente se tiene el manejo del recuadro de listado dentro de las 2 pestañas ya señaladas.

```python
# Recuadro de despliegue del listado para la pestaña de eliminación
def hilo_listar_eliminar_func(self):
    try:
        with open(NOMBRE_FIUNAMFS_IMG, "rb") as img_file:
            contenido = listar_contenido_directorio(img_file)
            self.txtListarContenidoEliminar.delete('1.0', tk.END)
            self.txtListarContenidoEliminar.insert(tk.END, contenido)
    except Exception as e:
        messagebox.showerror("Error", str(e))

# Recuadro de despliegue del listado para la pestaña de copiado
def hilo_listar_copiarAr_func(self):
    try:
        with open(NOMBRE_FIUNAMFS_IMG, "rb") as img_file:
            contenido = listar_contenido_directorio(img_file)
            self.txtListarContenidoCopiarAr.delete('1.0', tk.END)
            self.txtListarContenidoCopiarAr.insert(tk.END, contenido)
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

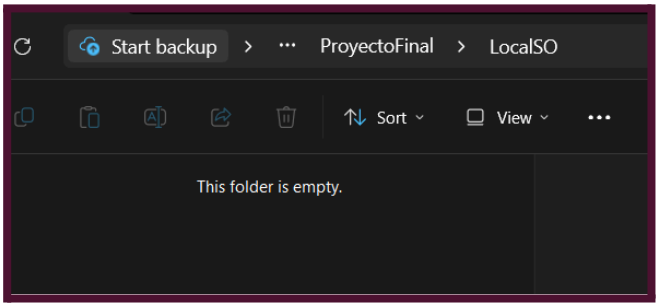
Finalmente, por medio de los botones de actualizado previamente definidos en cada pestaña, se hace la llamada nuevamente de cada hilo de listado para justamente actualizar el listado de archivos desplegado hacia el usuario, esto con el fin de mantener siempre informado al usuario sobre cualquier cambio realizado en el micro-sistema.

```python
# Actualización del listado en la pestaña de eliminación
def actualizar_listado_eliminar(self):
    self.hilo_listar_eliminar_func()

# Actualización del listado en la pestaña de eliminación
def actualizar_listado_copiarAr(self):
    self.hilo_listar_copiarAr_func()
```

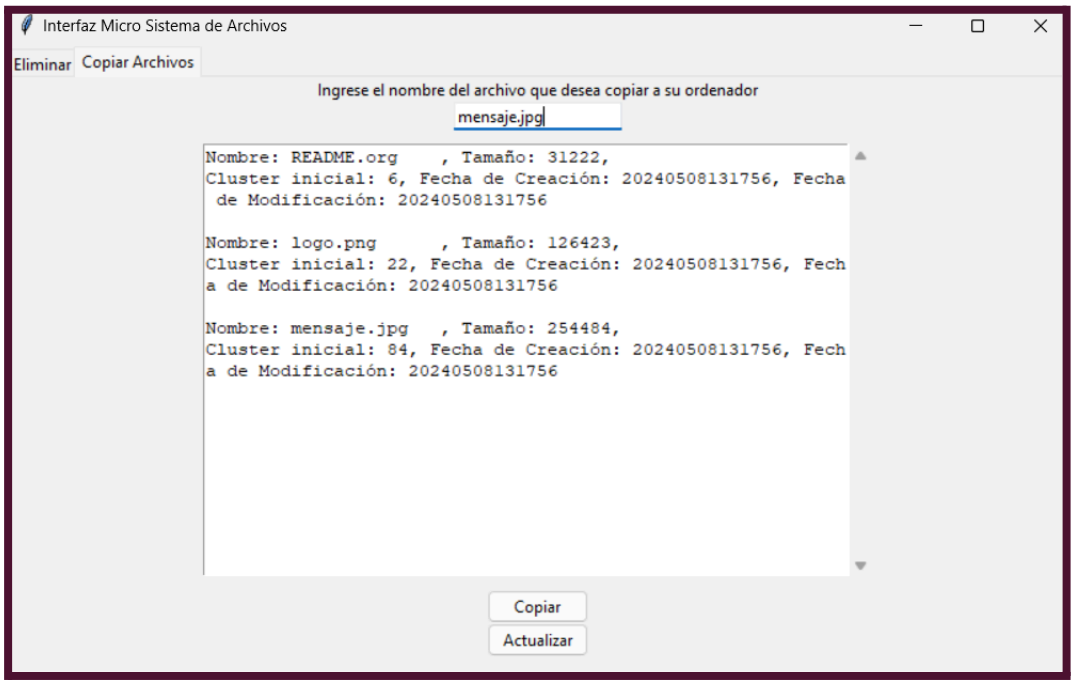
3. *Demostración del Funcionamiento del Programa*

A continuación se hace una muestra de ejecución tanto del copiado desde el micro-sistema de archivos a una carpeta local, como de la eliminación de un archivo del mismo micro-sistema.

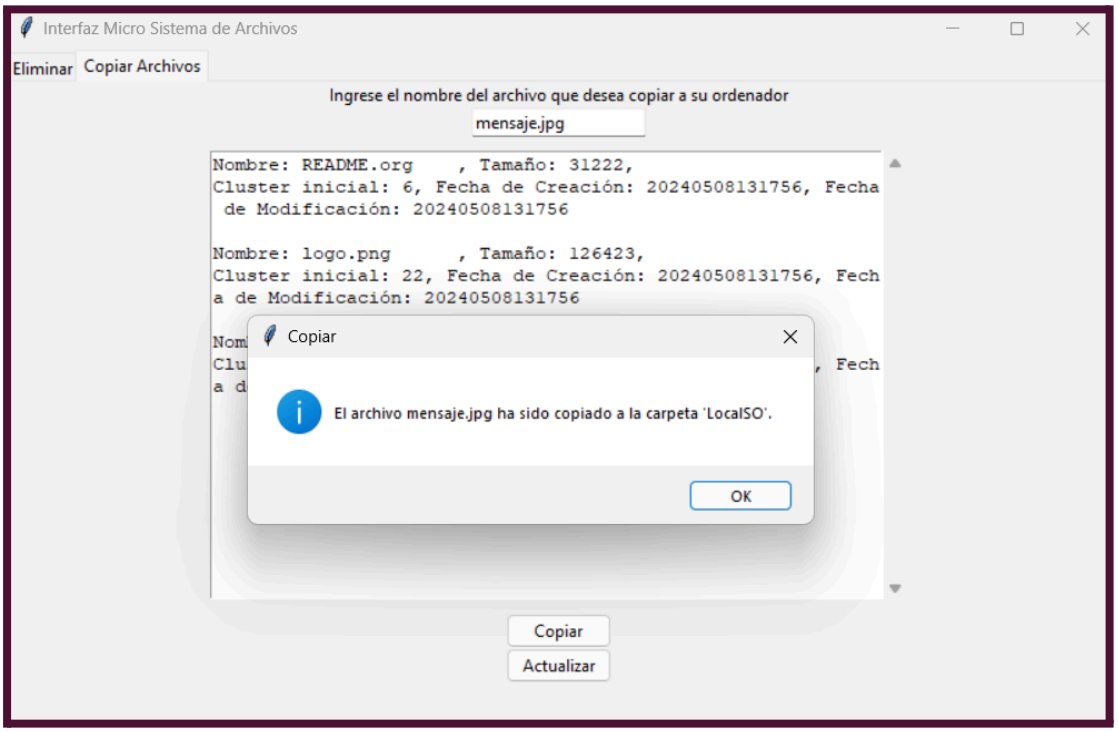
- *Copiado a carpeta local:*
  Primeramente se tiene a la carpeta local "LocalSO" vacía.



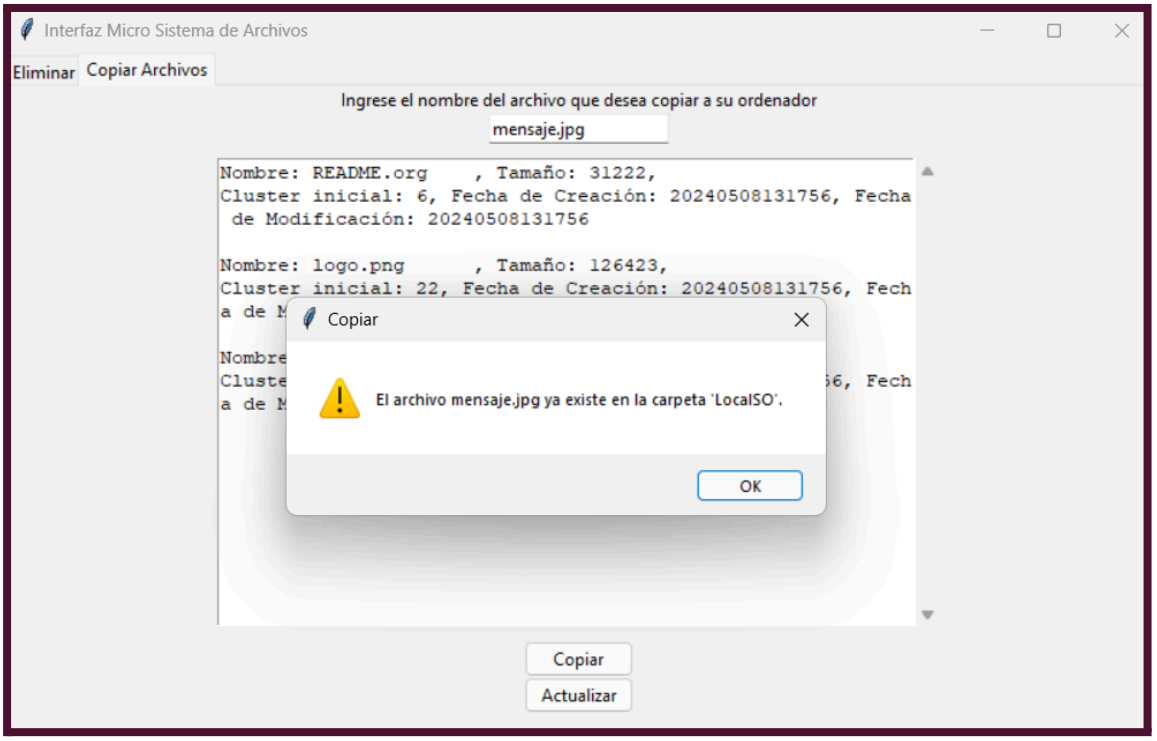
Tras ello, mediante la interfaz gráfica del programa se elige realizar el copiado del archivo "mensaje.jpg" del micro-sistema de archivos. Como puede observarse, justamente se muestra el listado actual de los archivos dentro del micro-sistema, incluyendo toda la información de cada archivo.



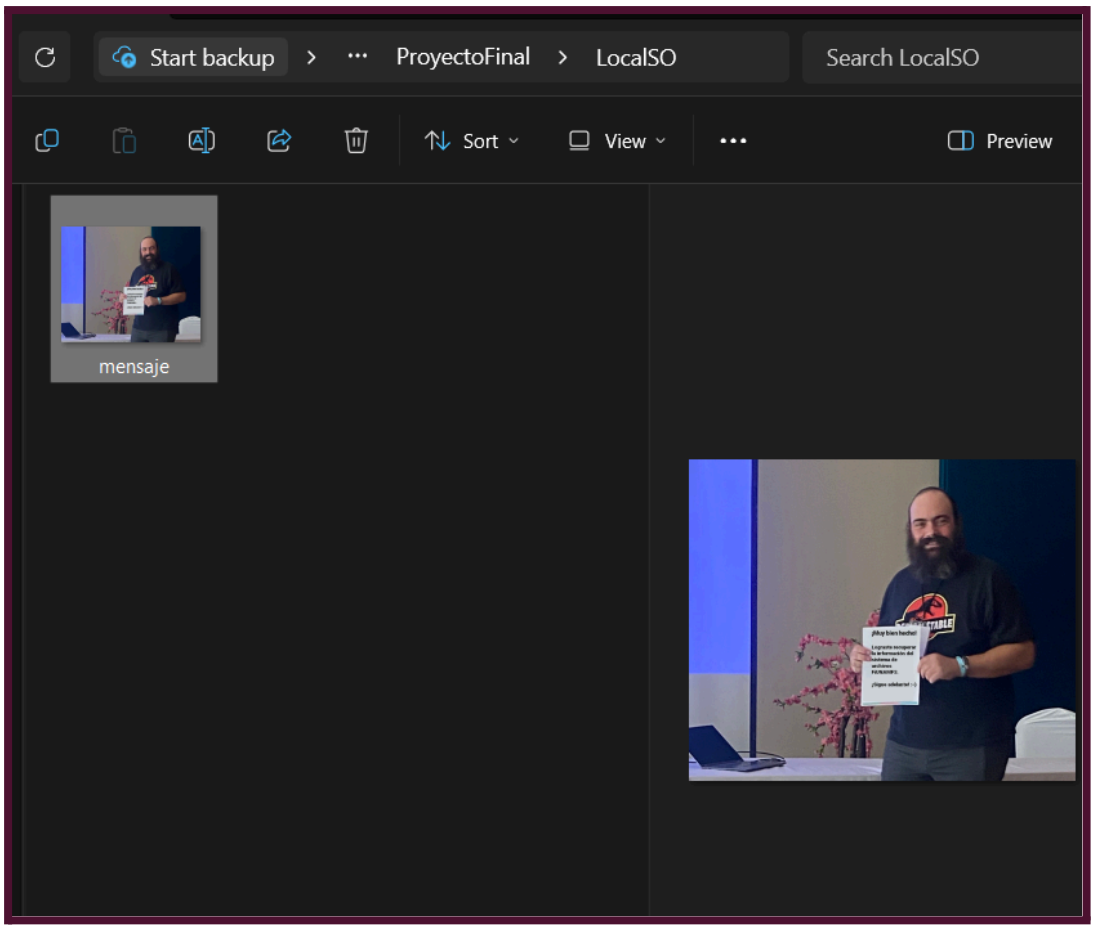
Se realiza de forma exitosa el copiado.



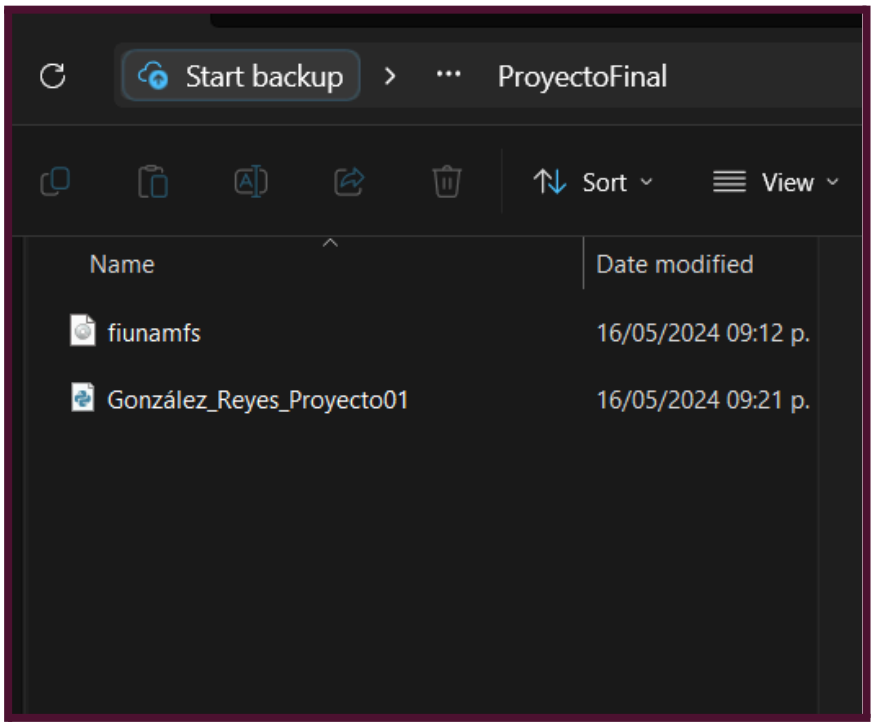
Se intenta realizar nuevamente el copiado.



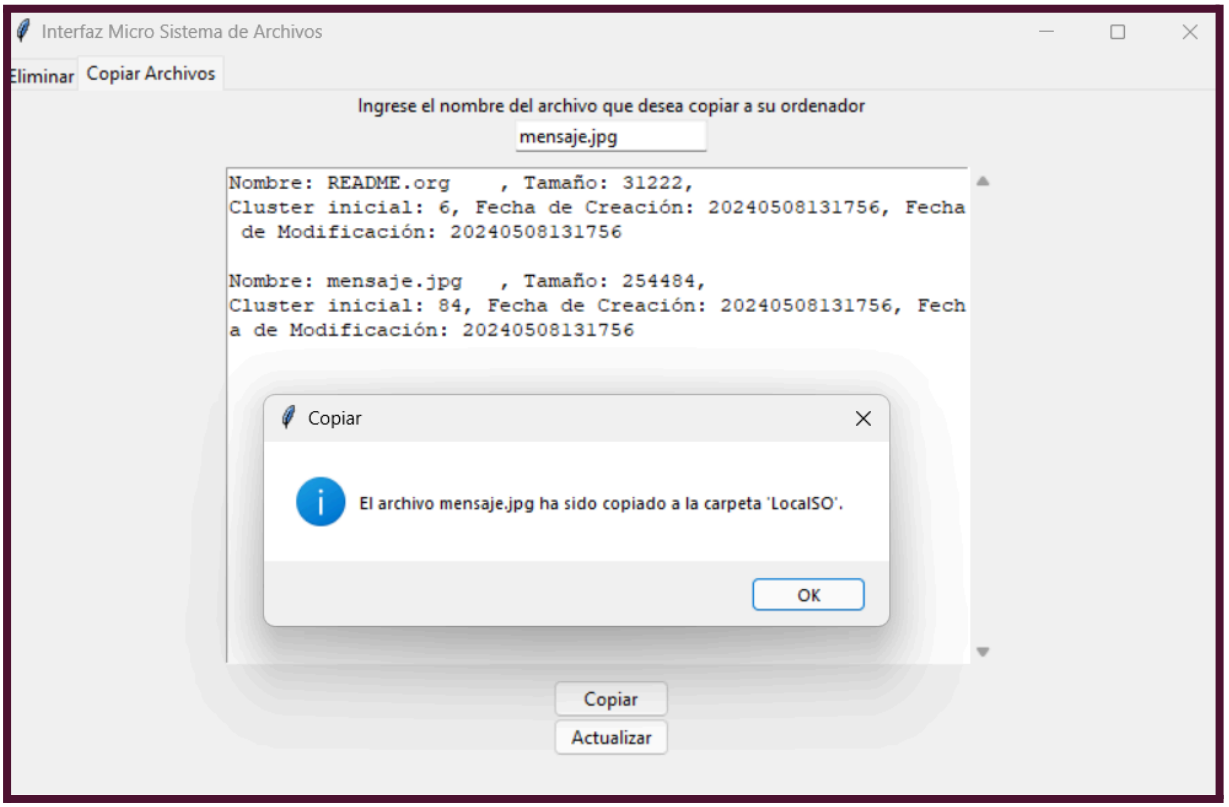
A continuación, se muestra la carpeta “LocalSO” con el archivo “mensaje.jpg” recién copiado.



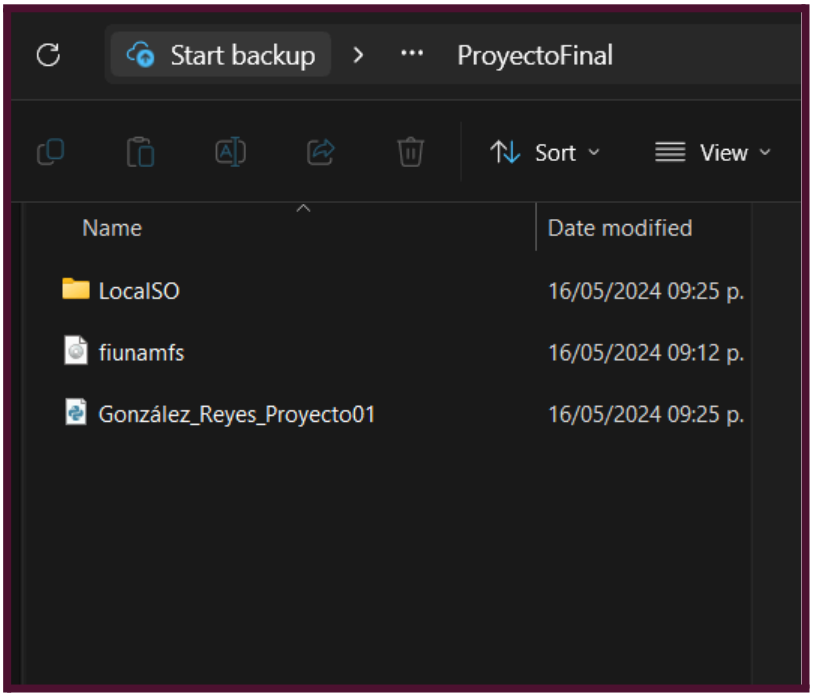
En caso de no existir la carpeta “LocalSO” en el sistema local, como ya fue mencionado anteriormente, el programa se encarga de realizar su creación en conjunto con el copiado correspondiente.



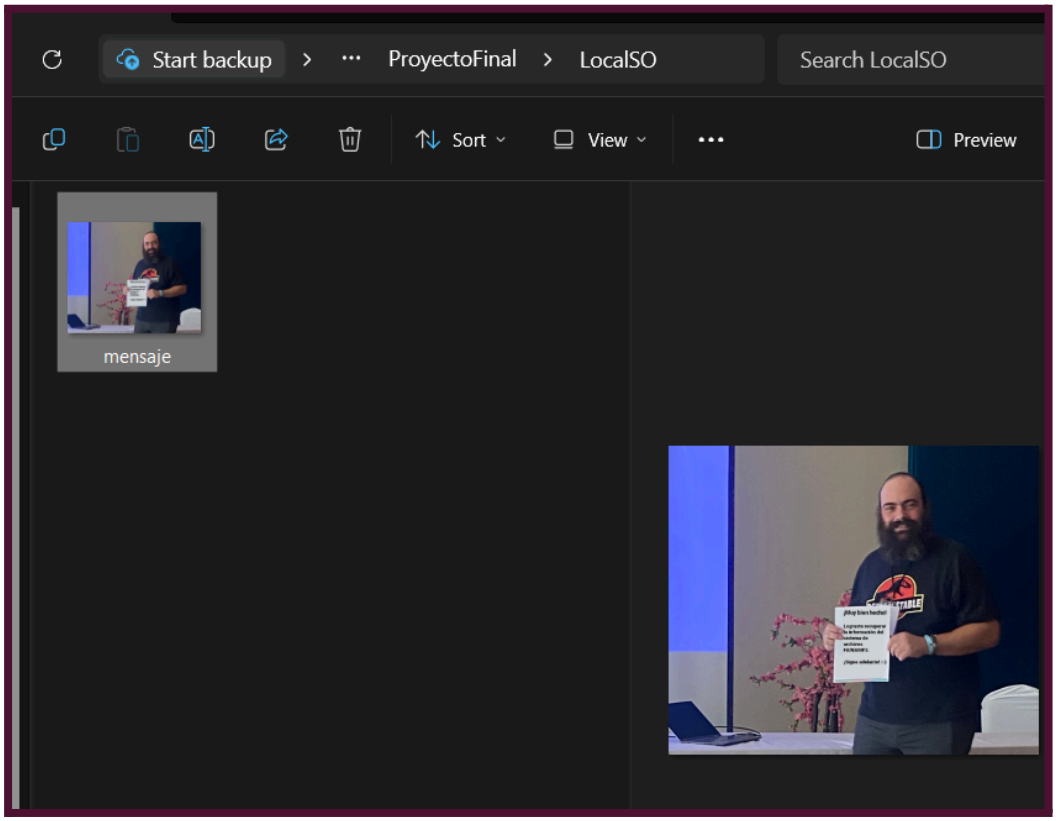
Se realiza la operación de copiado sobre el archivo “mensaje.jpg” a pesar de no existir la carpeta “LocalSO”.



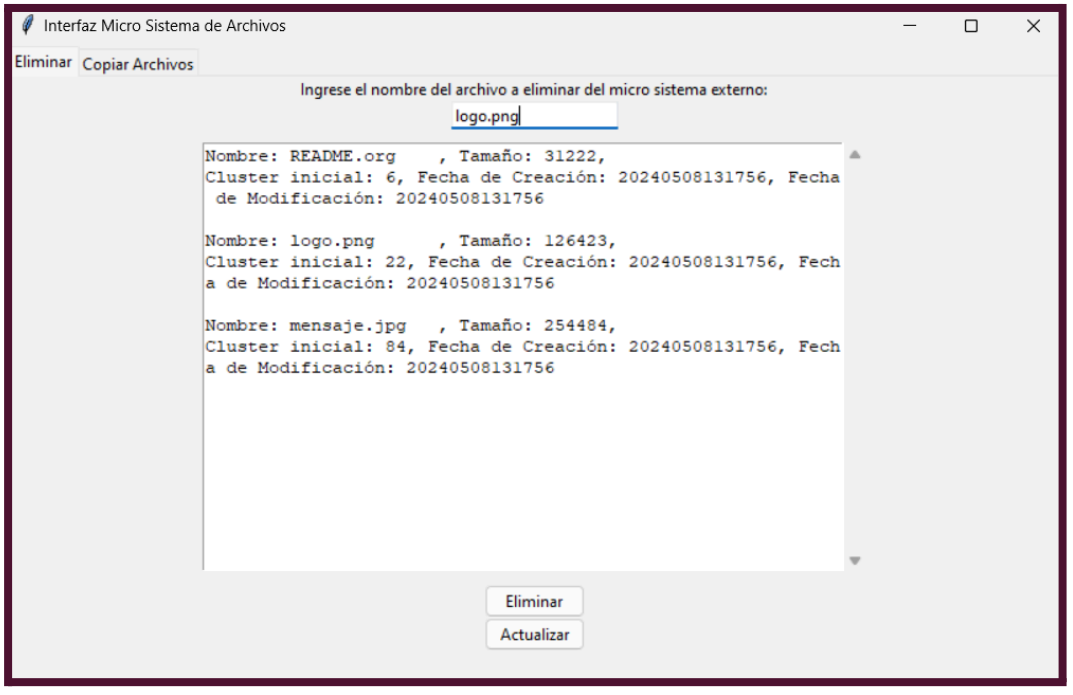
Carpeta “LocalSO” creada con éxito.



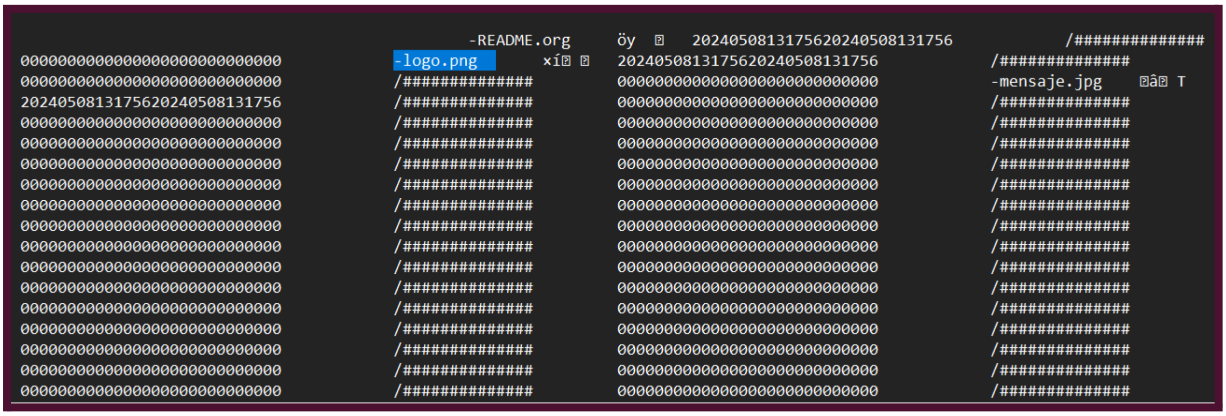
Archivo “mensaje.jpg” copiado con éxito en la nueva carpeta creada.



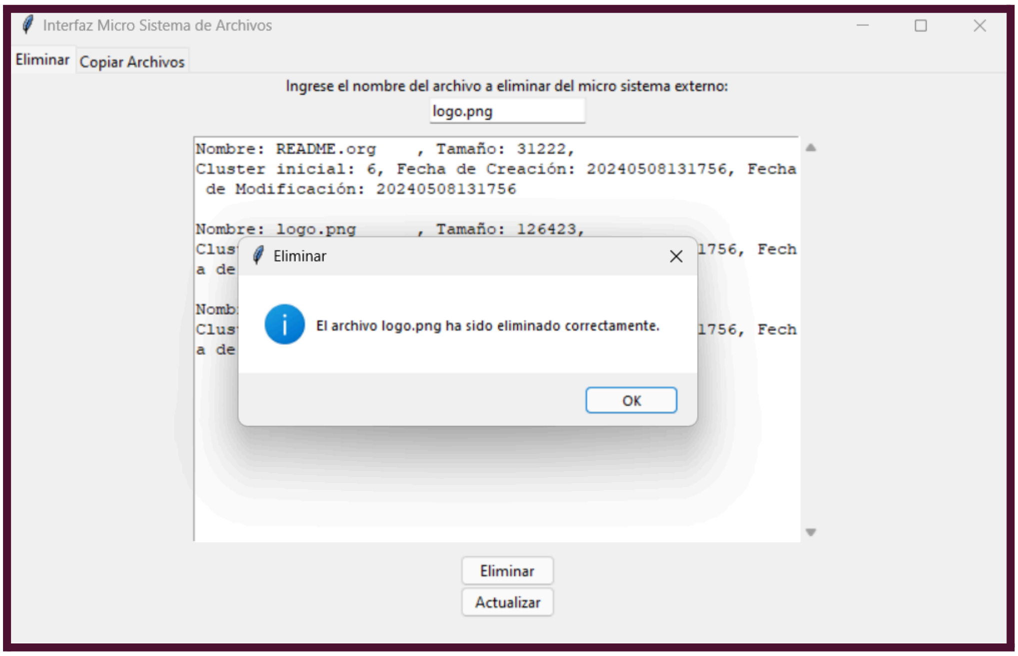
- *Borrado del micro-sistema de archivos*:
  Se tienen tres archivos en el micro-sistema, por lo que se eliminará el archivo denominado “logo.png”.



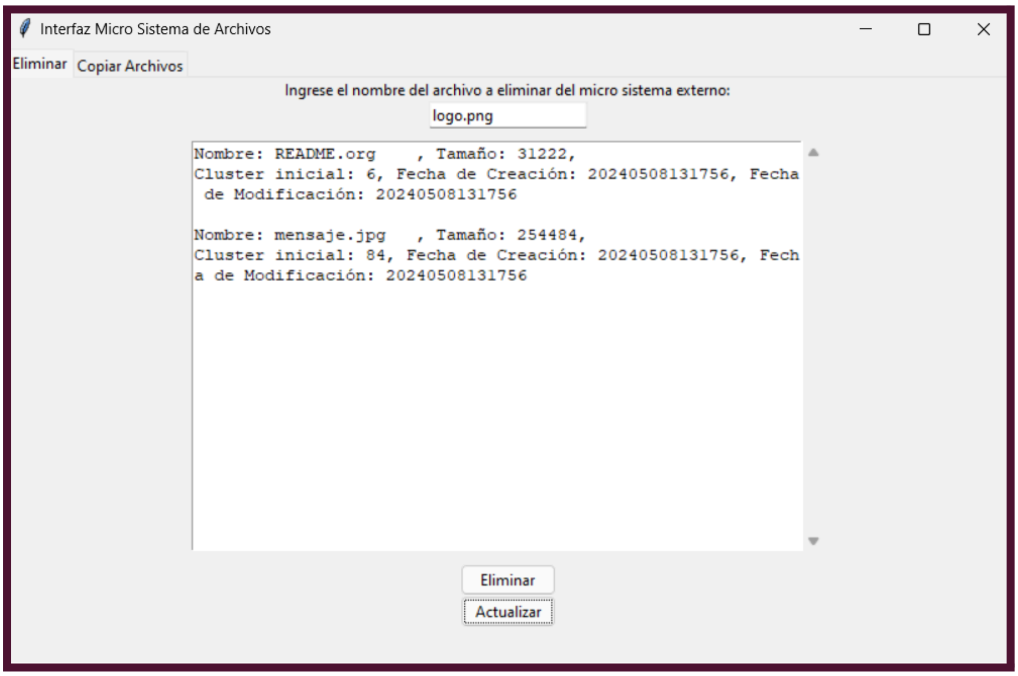
Se muestra el archivo “logo.png” tal cual como se encuentra en el micro-sistema.



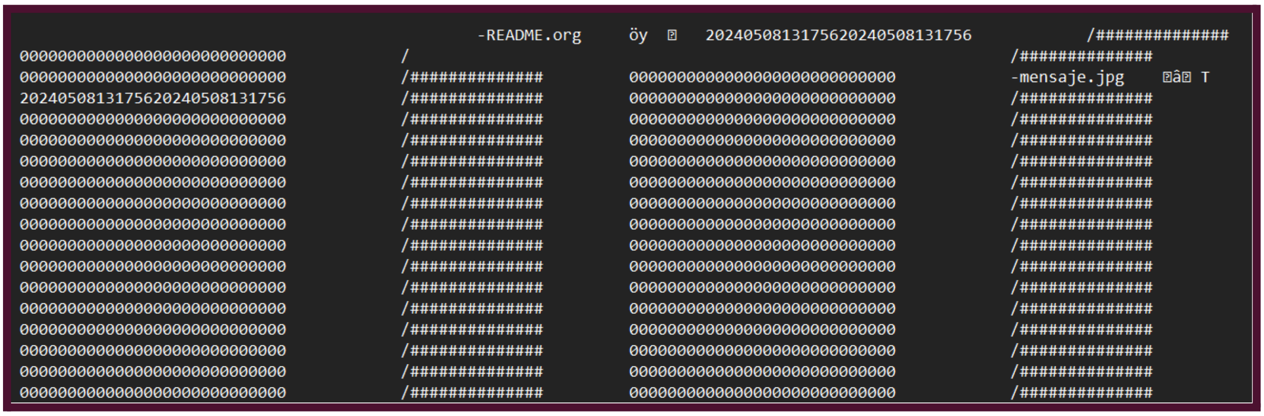
Se realiza la eliminación del archivo.



Actualizando el listado tras la eliminación del archivo.



Se corrobora finalmente la eliminación del archivo "logo.png" en el micro-sistema.



Con todo lo anterior, se demuestra el correcto funcionamiento tanto de la interfaz gráfica empleada, como todos los aspectos implementados para la realización del proyecto.

4. Referencias

- Python Docs. (S/F). *OS — Miscellaneous operating system interfaces*. Python.org. Recuperado de: https://docs.python.org/3.10/library/os.html

- Python Docs. (S/F). *Struct — Interpret bytes as packed binary data*. Python.org. Recuperado de: https://docs.python.org/3/library/struct.html

- Python Docs. (S/F). *Threading — Paralelismo basado en hilos.* Python.org. Recuperado de: https://docs.python.org/es/3.8/library/threading.html

- Python Docs. (S/F). *Tkinter Dialogs*. Python.org. Recuperado de: https://docs.python.org/3/library/dialog.html

- Python Docs. (S/F). *Tkinter.ttk — Tk themed widgets*. Python.org. Recuperado de: https://docs.python.org/es/3/library/tkinter.ttk.html

- Python Docs. (S/F). *Tkinter.messagebox — Tkinter message prompts*. Python.org. Recuperado de: https://docs.python.org/3/library/tkinter.messagebox.html m