Universidad Nacional Autónoma de México Facultad de Ingeniería
División de Ingeniería Eléctrica

Alumnos:
Hernández Ramírez Miguel Angel
López Tavera Alexa Fernanda

Profesor:
Ing. Gunnar Eyal Wolf Iszaevich

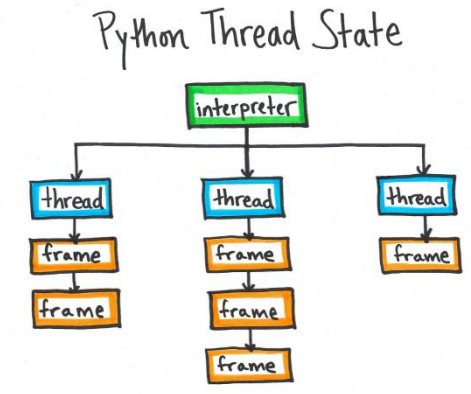
Asignatura: Sistemas Operativos

Grupo:
06

Tema:
Proyecto “(Micro) sistema de archivos multihilos”

Semestre:
2024-2

Fecha de Entrega: 19/Mayo/2024



1. *Threads In Python Presentation*, s.f

Índice

1. Planteamiento del Problema

Se pide crear un programa que sea capaz de obtener, crear y modificar información en el microsistema de archivos de la Facultad de Ingeniería que lleva por nombre: "*FiUnamFS*".

Este sistema debe ser capaz de:

1. Listar los contenidos del directorio.
2. Copiar uno de los archivos de dentro del *FiUnamFS* hacia tu sistema.
3. Copiar un archivo de tu computadora hacia tu *FiUnamFS*.
4. Eliminar un archivo del *FiUnamFS*.
5. El programa que desarrollen debe contar, por lo menos, dos hilos de ejecución, operando concurrentemente, y que se comuniquen su estado mediante mecanismos de sincronización.

2. Código explicado

Para la realización de nuestro proyecto elegimos utilizar el lenguaje de programación "Python" ya que, su sintaxis es más flexible a comparación de otros lenguajes.

A continuación, se presentan las partes por las que está compuesto nuestro código:

Las primeras 3 líneas del código se encargan de importar de los módulos que requeriremos a lo largo del programa.

```
1   import os
2   import struct
3   import threading
4   import tkinter as tk #interfaz grafica, porfavo
5   from tkinter import filedialog, messagebox
```

1. El primer módulo *"import os"* nos ayudará con el manejo de archivos.
2. El segundo módulo "*import struct*" se utiliza para trabajar con datos binarios y convertirlos en cadenas.
3. El tercer módulo es "*import threading*" que permite manejar la concurrencia entre hilos que se nos pide en las especificaciones del proyecto.
4. El cuarto módulo "*import tkinter*" es para crear interfaces gráficas, en la línea 5, declaramos herramientas de este módulo para diálogos de archivos y mensajes emergentes.

Las siguientes 5 líneas del código son la definición de constantes que utilizaremos a lo largo del código.

1. La primera constante define el tamaño del

```
7   TAMANO_CLUSTER = 1024
8   TAMANO_ENTRADA = 64
9   DIRECTORIO_INICIO = TAMANO_CLUSTER  # Cluster 1
10  DIRECTORIO_TAMANO = 4 * TAMANO_CLUSTER  # 4 clusters para el directorio
11  MAXIMO_CLUSTERS = 1440 // 4  # Asumiendo que el tamaño total es 1440KB
```

cluster que se define como 1024 bytes.

2. La segunda constante define el tamaño de entrada del directorio como 64 bytes.
3. La tercer constante define como inicio del directorio en el archivo de imagen.
4. La cuarta constante define el tamaño total del directorio en 4 clusters.
5. La última constante nos da el número máximo de clusters que se basa en el tamaño total de 1440KB.

La línea 13, crea un candado (lock) que ayuda a sincronizar el acceso a recursos compartidos entre los hilos implementados.

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

Enseguida definimos la primera función del programa.

```
15  def leer_superbloque(fiunamfs_img):
16      resultado = ""
17      with open(fiunamfs_img, 'rb') as f:
18          f.seek(0)
19          nombre_fs = f.read(8).decode('ascii').strip()
20          f.seek(10)
21          version = f.read(5).decode('ascii').rstrip('\x00').strip()
22          resultado += f"Nombre FS leído: {nombre_fs}, Versión leída: {version}\n"
23          if nombre_fs != "FiUnamFS" or version.replace('-', '.') != "24.2":
24              resultado += "Valores no coinciden, se esperaba FiUnamFS y 24.2\n"
25          else:
26              resultado += "Superbloque válido\n"
27      return resultado
```

La función "*leer_superbloque*" se encargará, como su nombre lo dice, se encarga de leer el superbloque del sistema de archivos.

Con la línea 16, abrimos el archivo de imagen del sistema de archivos en modo de *"solo lectura binaria"*, en las siguientes líneas hasta la línea 22, se lee y decodifica el nombre del sistema de archivos y su versión desde el superbloque.

Por último, verifica si los valores leídos coinciden con los esperados y agrega el resultado a *"resultado"* y devuelve "*resultado*".

La siguiente función "*listar_directorio*" como su nombre lo dice, lista el contenido del directorio.

```
29  def listar_directorio(fiunamfs_img):
30      resultado = ""
31      with open(fiunamfs_img, 'rb') as f:
32          f.seek(DIRECTORIO_INICIO)
33          for _ in range(DIRECTORIO_TAMANO // TAMANO_ENTRADA):
34              entrada = f.read(TAMANO_ENTRADA)
35              tipo_archivo = entrada[0:1].decode('ascii')
36              nombre = entrada[1:16].decode('ascii').rstrip()
37              if nombre != '#' * 15:
38                  tam_archivo = struct.unpack('<I', entrada[16:20])[0]
39                  cluster_ini = struct.unpack('<I', entrada[20:24])[0]
40                  fecha_creacion = entrada[24:38].decode('ascii')
41                  fecha_modificacion = entrada[38:52].decode('ascii')
42                  resultado += f"Tipo: {tipo_archivo}, Nombre: {nombre}, Tamaño: {tam_archivo}, Cluster inicial: {cluster_ini}, Creación: {fecha_cr
43      return resultado
```

Para la línea 31, se abre el archivo en modo binario de solo lectura, después, se posiciona en el inicio del directorio, itera sobre las entradas del directorio, leyendo y decodificando cada entrada.

Al final, verifica si la entrada no está vacía, si no lo está, agrega los detalles de la entrada al "*resultado*" y regresa "*resultado*".

La tercera función de nuestro programa "*copiar_a_sistema*" nos sirve para copiar un archivo desde el sistema de archivos FiUnamFS al sistema de archivos local.

```
45  def copiar_a_sistema(fiunamfs_img, nombre_archivo, destino):
46      with lock:
47          app.lock_label.config(text="Estado del Lock: Adquirido (Copiar a sistema)")
48      with open(fiunamfs_img, 'rb') as f:
49          f.seek(DIRECTORIO_INICIO)
50          for _ in range(DIRECTORIO_TAMANO // TAMANO_ENTRADA):
51              entrada = f.read(TAMANO_ENTRADA)
52              tipo_archivo = entrada[0:1]
53              if tipo_archivo == b'-':
54                  nombre, tam, cluster_ini = (
55                      entrada[1:16].decode('ascii').rstrip(),
56                      struct.unpack('<I', entrada[16:20])[0],
57                      struct.unpack('<I', entrada[20:24])[0]
58                  )
59                  if nombre.rstrip('\x00').strip() == nombre_archivo.rstrip('\x00').strip():
60                      f.seek(cluster_ini * TAMANO_CLUSTER)
61                      datos = f.read(tam)
62                      with open(destino, 'wb') as archivo_destino:
63                          archivo_destino.write(datos)
64                      return
```

Con el *with lock* definido en la línea 46, se adquiere un candado que nos asegura que la operación sea atómica, es decir, que no se presenten condiciones de carrera entre hilos.

En las líneas siguientes, se abre la imagen en modo lectura binaria, se posiciona en el inicio del directorio y lee las entradas buscando el archivo con el nombre especificado.

Si encuentra el archivo, se posiciona en el clúster inicial del archivo y lee su contenido.

Finalmente escribe el contenido leído en el destino definido.

La función siguiente "*copiar_a_fiunamfs*" se encarga de copiar un archivo desde el sistema local hasta el sistema de archivos FiUnamFS.

```
66  def copiar_a_fiunamfs(fiunamfs_img, archivo_origen, nombre_destino):
67      with lock:
68          app.lock_label.config(text="Estado del Lock: Adquirido (Copiar a FiUnamFS)")
69      with open(fiunamfs_img, 'r+b') as f:
70          tam_origen = os.path.getsize(archivo_origen)
71          cluster_libre = 5
72          posicion_entrada_libre = None
73
74          f.seek(DIRECTORIO_INICIO)
75          for _ in range(DIRECTORIO_TAMANO // TAMANO_ENTRADA):
76              posicion_actual = f.tell()
77              entrada = f.read(TAMANO_ENTRADA)
78              tipo_archivo = entrada[0:1]
79              cluster_ini = struct.unpack('<I', entrada[20:24])[0]
80
81              if tipo_archivo == b'/' and posicion_entrada_libre is None:
82                  posicion_entrada_libre = posicion_actual
83
84              if cluster_ini >= cluster_libre:
85                  cluster_libre = cluster_ini + 1
86
87          if posicion_entrada_libre is None:
88              raise Exception("No hay espacio en el directorio")
89          else:
90              with open(archivo_origen, 'rb') as archivo_origen_f:
91                  f.seek(cluster_libre * TAMANO_CLUSTER)
92                  f.write(archivo_origen_f.read())
93
```

Lo primero que se hace es adquirir un lock para asegurar, al igual que en la función anterior, que la operación se ejecute de forma ininterrumpida.

Posteriormente se abre el archivo en modo "lectura y escritura binaria", obtiene el tamaño del archivo de origen y busca un cluster libre y una entrada libre en el directorio.

```
94                  f.seek(posicion_entrada_libre)
95                  f.write(b'-' + nombre_destino.ljust(15).encode('ascii'))
96                  f.write(struct.pack('<I', tam_origen))
97                  f.write(struct.pack('<I', cluster_libre))
```

Si no encuentra un espacio, lanza una excepción, pero si encuentra una entrada libre, se posiciona en el clúster libre y escribe el contenido.

Finalmente, se actualiza la entrada del directorio con la información del archivo copiado.

Para la función "*eliminar_archivo*" se adquiere nuevamente un candado que nos asegure que el acceso a esta sección será exclusiva.

```
99  def eliminar_archivo(fiunamfs_img, nombre_archivo):
100     with lock:
101         app.lock_label.config(text="Estado del Lock: Adquirido (Eliminar archivo)")
102     with open(fiunamfs_img, 'r+b') as f:
103         f.seek(DIRECTORIO_INICIO)
104         for _ in range(DIRECTORIO_TAMANO // TAMANO_ENTRADA):
105             posicion = f.tell()
106             entrada = f.read(TAMANO_ENTRADA)
107             nombre = entrada[1:16].decode('ascii').rstrip()
108             if nombre.rstrip('\x00').strip() == nombre_archivo.rstrip('\x00').strip():
109                 f.seek(posicion)
110                 f.write(b'/' + b' ' * 15)
111                 return
```

Inmediatamente después, busca la entrada del archivo a eliminar, si lo encuentra, se marcará la entrada como "libre" y se imprimirá el mensaje de "*Archivo eliminado*".

Por último, tenemos la clase "Application". Esta clase es la encargada de crear una interfaz gráfica que permita la interacción con el sistema de archivos *FiUnamFS*, a través de herencia de la clase base Tkinter.

```
113 class Application(tk.Tk):
114     def __init__(self):
115         super().__init__()
116         self.title("Sistema de Archivos FiUnamFS")
117         self.geometry("800x500")  # Ajustar tamaño de ventana
118
119         # Configuración inicial
120         self.fiunamfs_img = r"C:\Users\alexa\OneDrive\Escritorio\ProyectoFinal_SO\fiunamfs.img"  # Cambia la ruta por la ubicación de tu archivo
121
122         self.create_widgets()
123
124     def create_widgets(self):
125         self.resultado_texto = tk.Text(self, height=30, width=100)  # Ajustar el tamaño del cuadro de texto
126         self.resultado_texto.pack(fill=tk.BOTH, expand=True)  # Hacer que el cuadro de texto se expanda
127
128         self.lock_label = tk.Label(self, text="Estado del Lock: Desbloqueado")
129         self.lock_label.pack()
130
131         tk.Button(self, text="Leer el superbloque", command=self.leer_superbloque).pack(fill=tk.X, pady=5)
132         tk.Button(self, text="Listar directorio", command=self.listar_directorio).pack(fill=tk.X, pady=5)
133         tk.Button(self, text="Copiar archivo de FiUnamFS a sistema", command=self.copiar_a_sistema).pack(fill=tk.X, pady=5)
134         tk.Button(self, text="Copiar archivo de sistema a FiUnamFS", command=self.copiar_a_fiunamfs).pack(fill=tk.X, pady=5)
135         tk.Button(self, text="Eliminar archivo de FiUnamFS", command=self.eliminar_archivo).pack(fill=tk.X, pady=5)
136         tk.Button(self, text="Salir", command=self.quit).pack(fill=tk.X, pady=5)
```

La primer parte, *"def __init__"* es el constructor de la clase, aquí se configura el título de la ventana, su tamaño y la ruta del archivo de imagen del sistema de archivos. Con <u>create_widgets,</u> crearemos los botones de la interfaz.

A partir de la línea 131 hasta la línea 136, se crean los botones que aparecerán en la ventana principal; cada botón se asociará a una acción específica como:

- Leer el superbloque
- Listar el directorio
- Copiar archivos al sistema
- Copiar archivos a fiunamfs.
- Eliminar archivos

```
138     def leer_superbloque(self):
139         resultado = leer_superbloque(self.fiunamfs_img)
140         self.mostrar_resultado(resultado)
141
142     def listar_directorio(self):
143         resultado = listar_directorio(self.fiunamfs_img)
144         self.mostrar_resultado(resultado)
145
146     def mostrar_resultado(self, resultado):
147         self.resultado_texto.delete(1.0, tk.END)  # Limpiar el contenido anterior
148         self.resultado_texto.insert(tk.END, resultado)
149
150     def copiar_a_sistema(self):
151         nombre_archivo = simpledialog.askstring("Nombre del archivo", "Ingresa el nombre del archivo en FiUnamFS:")
152         if nombre_archivo:
153             destino = filedialog.asksaveasfilename(initialfile=nombre_archivo)
154             if destino:
155                 try:
156                     copiar_a_sistema(self.fiunamfs_img, nombre_archivo, destino)
157                     messagebox.showinfo("Éxito", f"Archivo copiado con éxito a {destino}")
158                 except Exception as e:
159                     messagebox.showerror("Error", f"Error al copiar el archivo: {e}")
160
```

```python
    def copiar_a_fiunamfs(self):
        origen = filedialog.askopenfilename(title="Selecciona el archivo a copiar")
        if origen:
            nombre_destino = simpledialog.askstring("Nombre del archivo", "Ingresa el nombre del archivo en FiUnamFS:")
            try:
                copiar_a_fiunamfs(self.fiunamfs_img, origen, nombre_destino)
                messagebox.showinfo("Éxito", "Archivo copiado con éxito a FiUnamFS")
            except Exception as e:
                messagebox.showerror("Error", f"Error al copiar el archivo: {e}")

    def eliminar_archivo(self):
        nombre_archivo = simpledialog.askstring("Nombre del archivo", "Ingresa el nombre del archivo a eliminar de FiUnamFS:")
        if nombre_archivo:
            try:
                eliminar_archivo(self.fiunamfs_img, nombre_archivo)
                messagebox.showinfo("Éxito", "Archivo eliminado con éxito")
            except Exception as e:
                messagebox.showerror("Error", f"Error al eliminar el archivo: {e}")

if __name__ == "__main__":
    app = Application()
    app.mainloop()
```

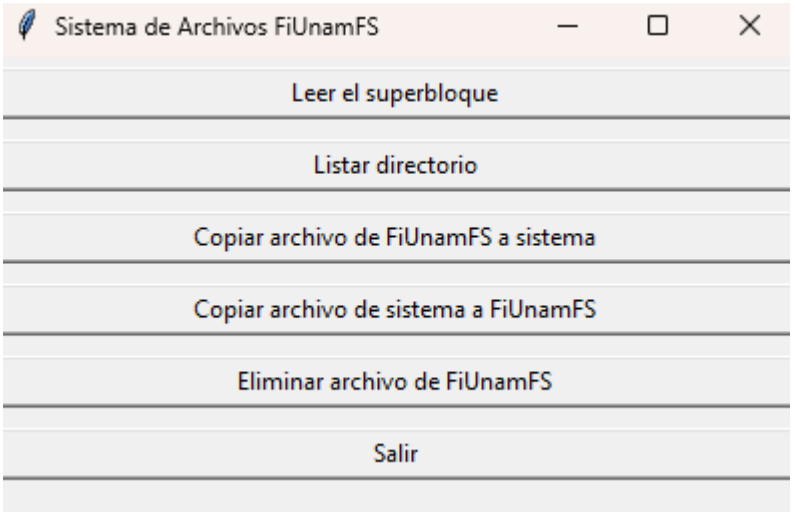
Que a su vez llamarán a las funciones previamente descritas para realizar la acción requerida.
En el caso de las funciones para copiar archivos y eliminar, el programa pedirá al usuario el nombre del archivo.

Por último, se ejecuta la clase Application.

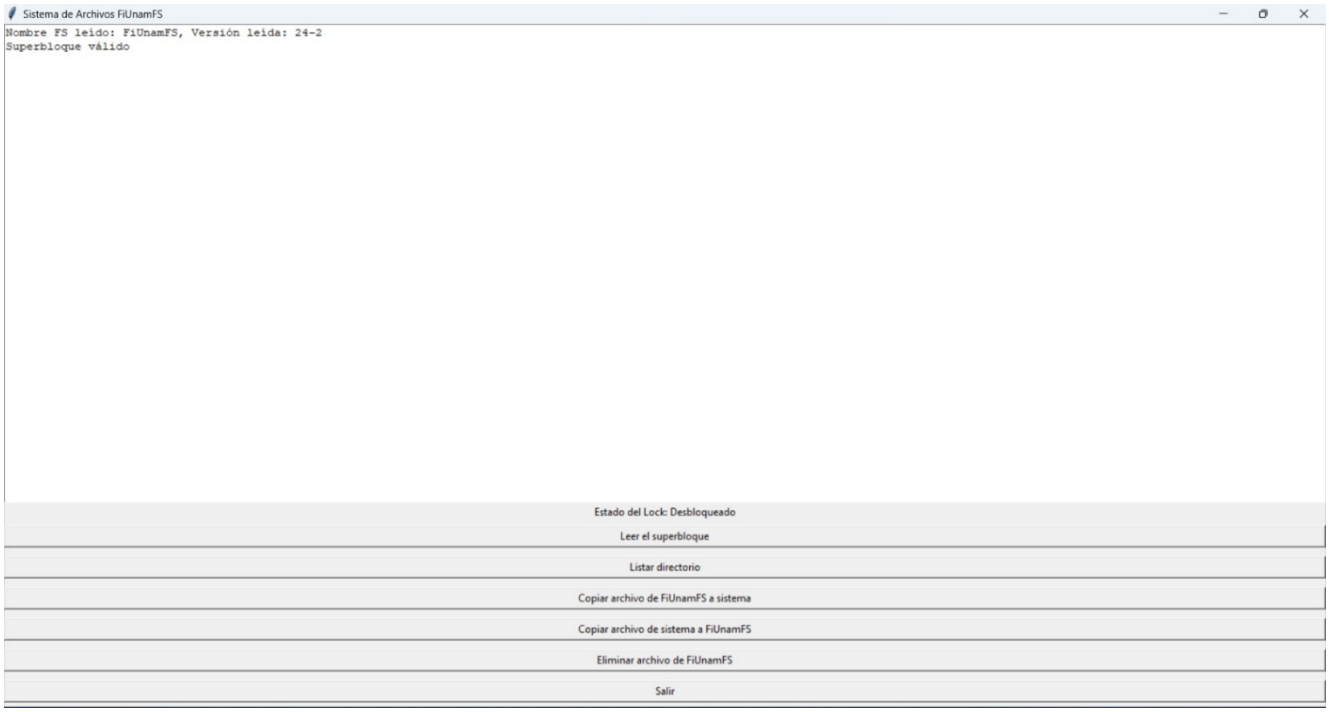
3. Funcionamiento del código: Ejemplo de ejecución

Nota: Recuerda modificar la ruta de ubicación a la ruta en donde tengas tu archivo .img.

Al iniciar la ejecución del programa, se abre la siguiente ventana:

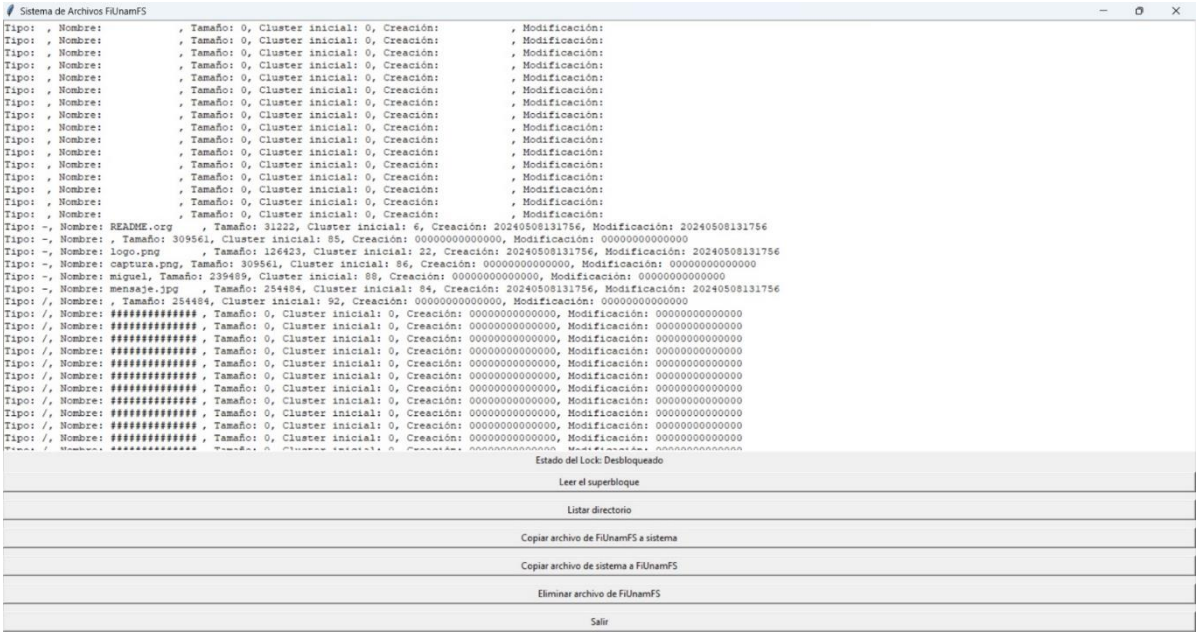


En donde se nos muestran las operaciones disponibles que puede realizar nuestro programa, seleccionaremos la primera opción "*Leer el super bloque*":



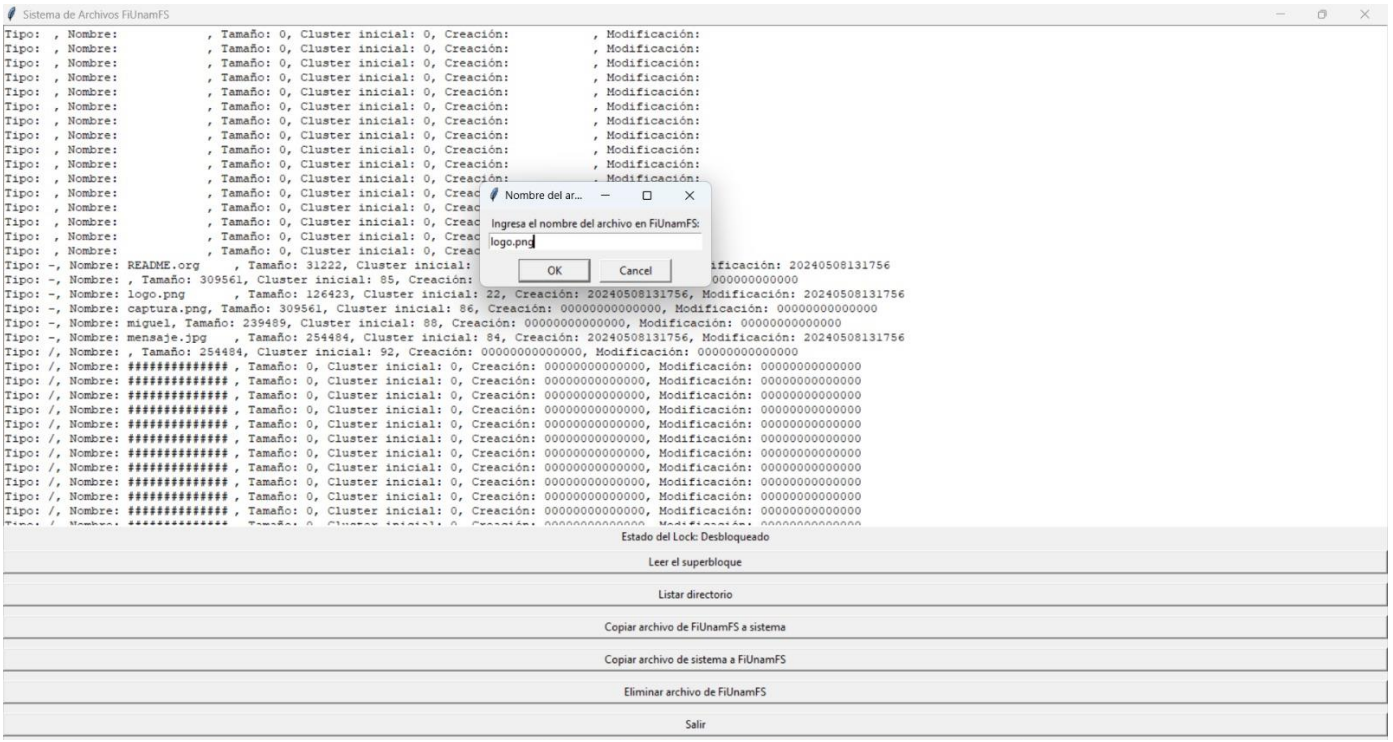
Al seleccionar esta opción nos muestra un mensaje de validación del superbloque.

Ahora seleccionaremos la segunda opción "*Listar directorio*":

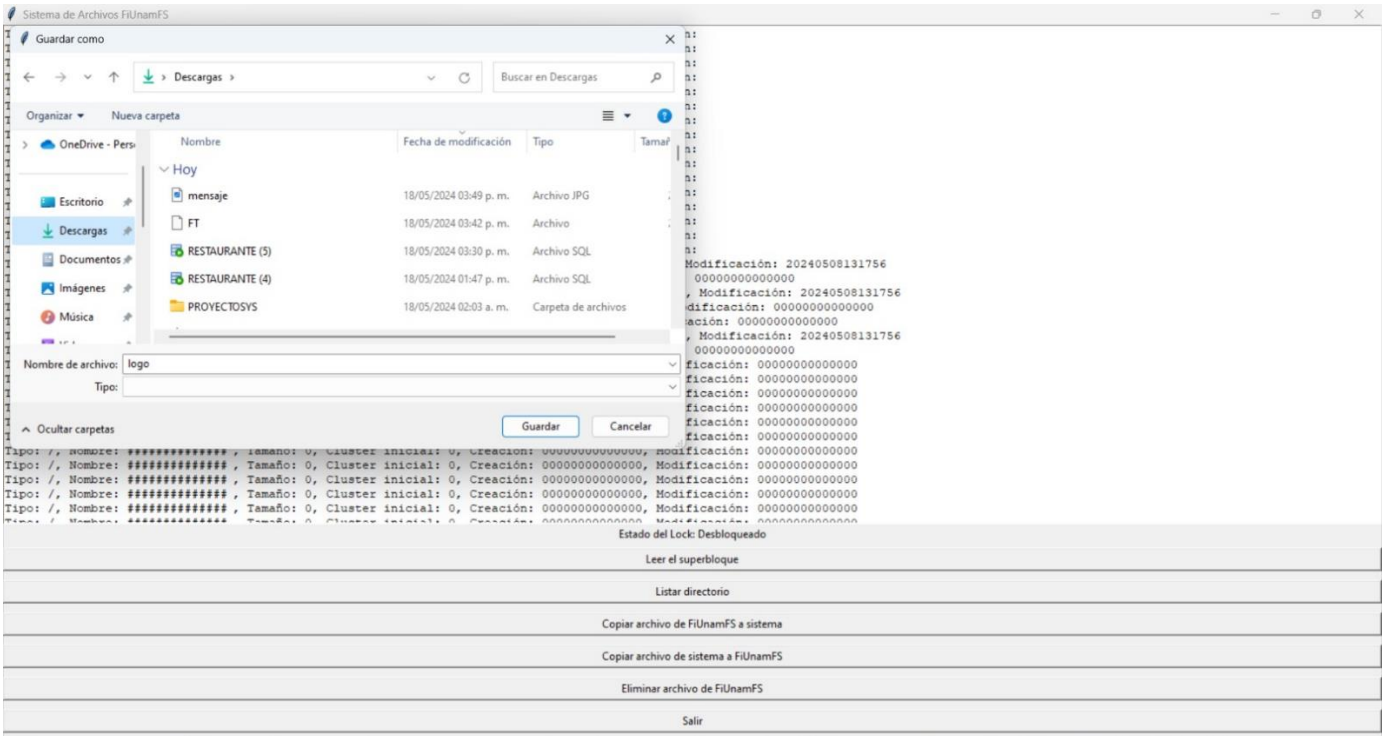


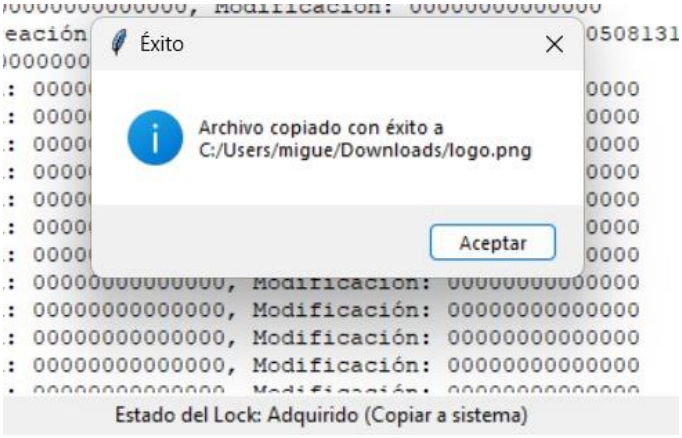
Cuando elegimos esta opción, en pantalla se muestra la información del directorio, el tipo, el nombre del archivo, el tamaño, el clúster inicial, creación y modificación, los espacios en blanco del directorio se representan con el símbolo "#" como fue requerido.
Como podemos ver, nos muestra también los archivos que ya están dentro del directorio.

Ahora seleccionaremos la tercera opción "*Copiar archivo de FiUnamFS a sistema*"



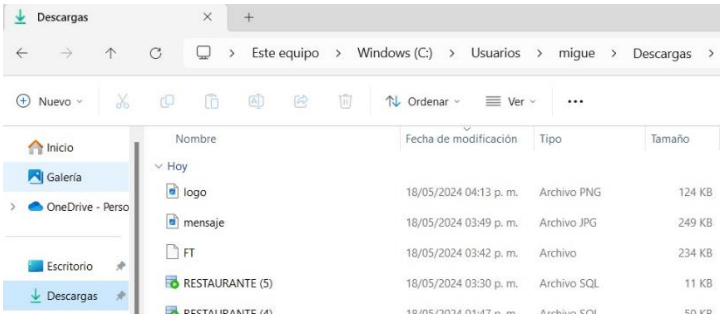
Cuando damos a la tercera opción, se abre una ventana más pequeña que nos pedirá escribir el nombre del archivo que se encuentra en el sistema FiUnamFS, al darle "OK", nos abrirá la ventana de nuestro explorador de archivos para seleccionar la ruta deseada:



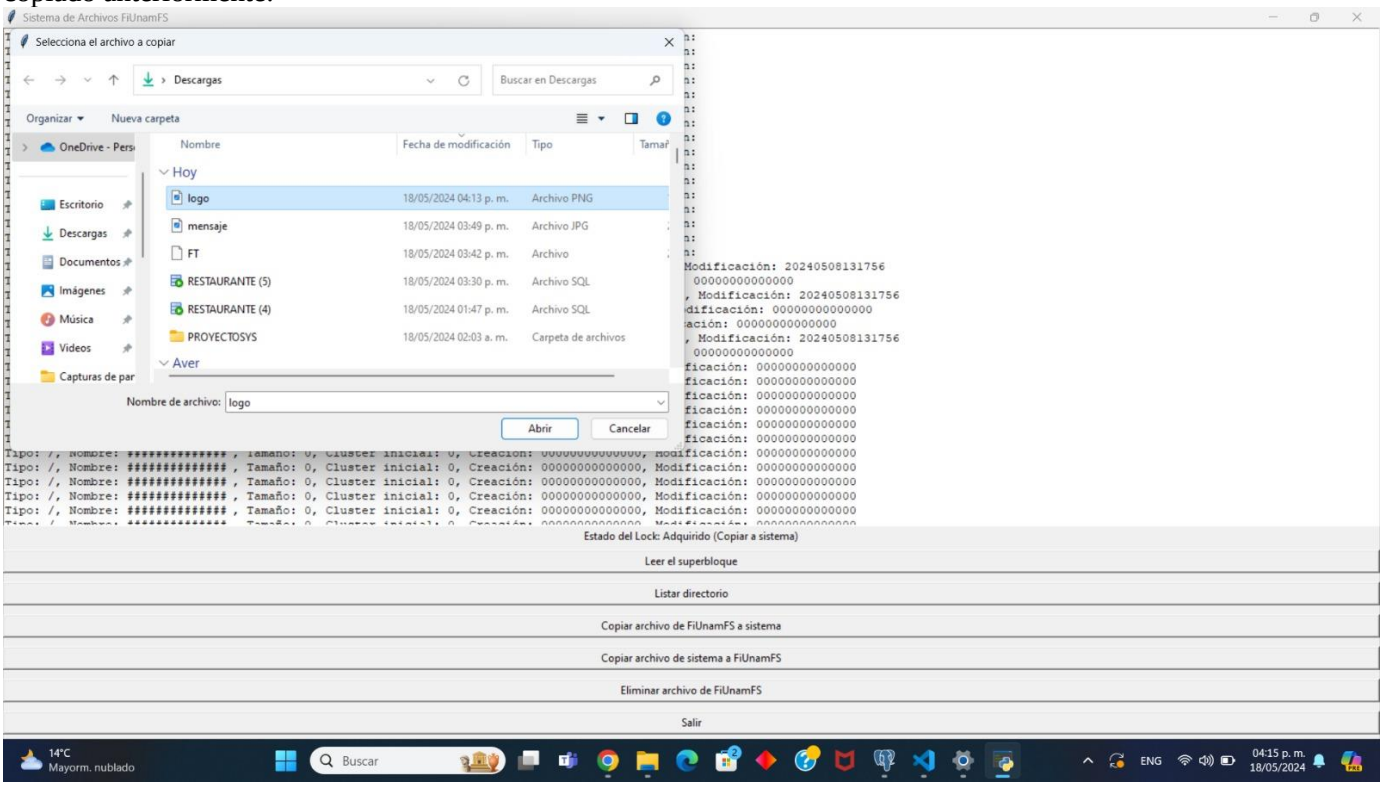


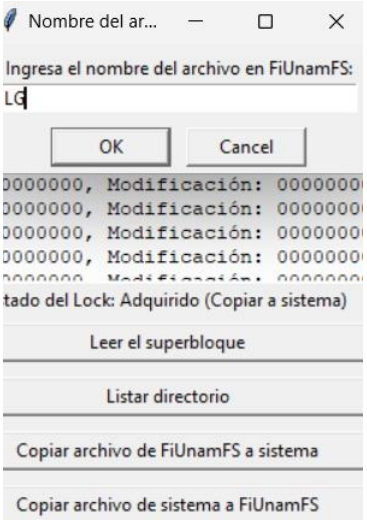
Al finalizar la operación, nos mostrará una nueva ventana que nos dirá si se copió con éxito nuestro archivo.
Como podemos ver, el estado del lock adquirido es el de "Copiar a sistema".

Podemos verificar que se ha realizado la copia:

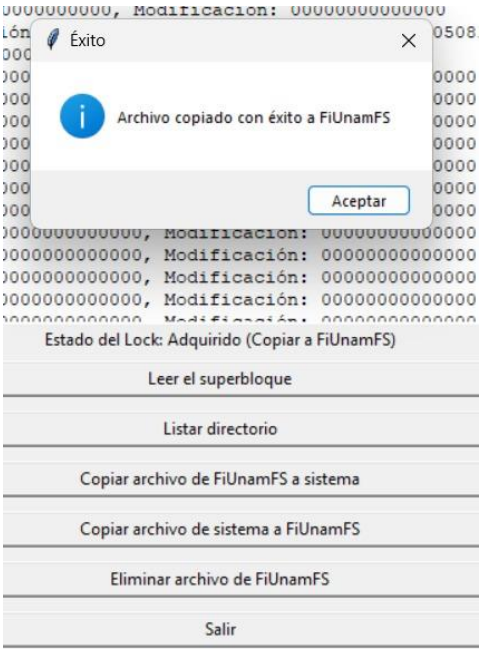


Ahora para ejemplificar la cuarta opción "*Copiar archivo de sistema a FiUnamFS*", utilizaremos el mismo archivo copiado anteriormente.



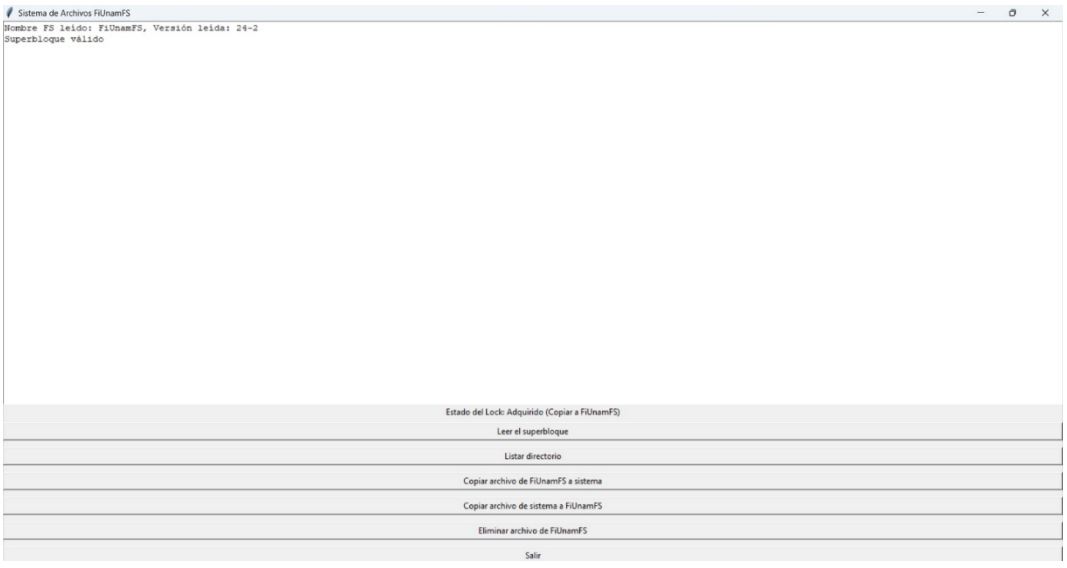


Para verificar que sí funciona, renombraremos el archivo con el nombre "LG".

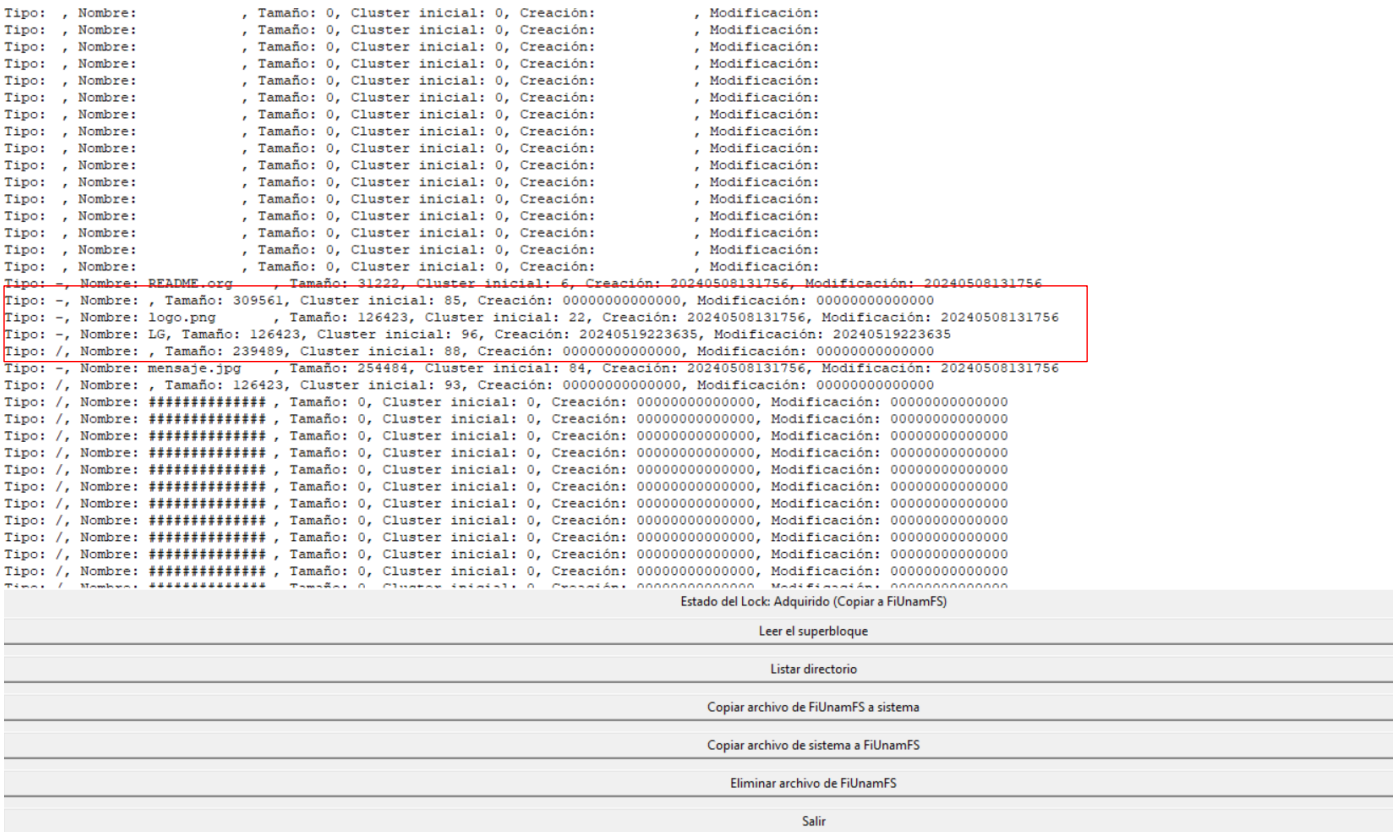


Como podemos ver, manda un mensaje de que se ha copiado con éxito el archivo al sistema FiUnamFS.
También podemos observar que el estado del lock ha cambiado a "*Copiar a FiUnamFS*"

Para verificar el estado actual del directorio, leemos nuevamente el superbloque:

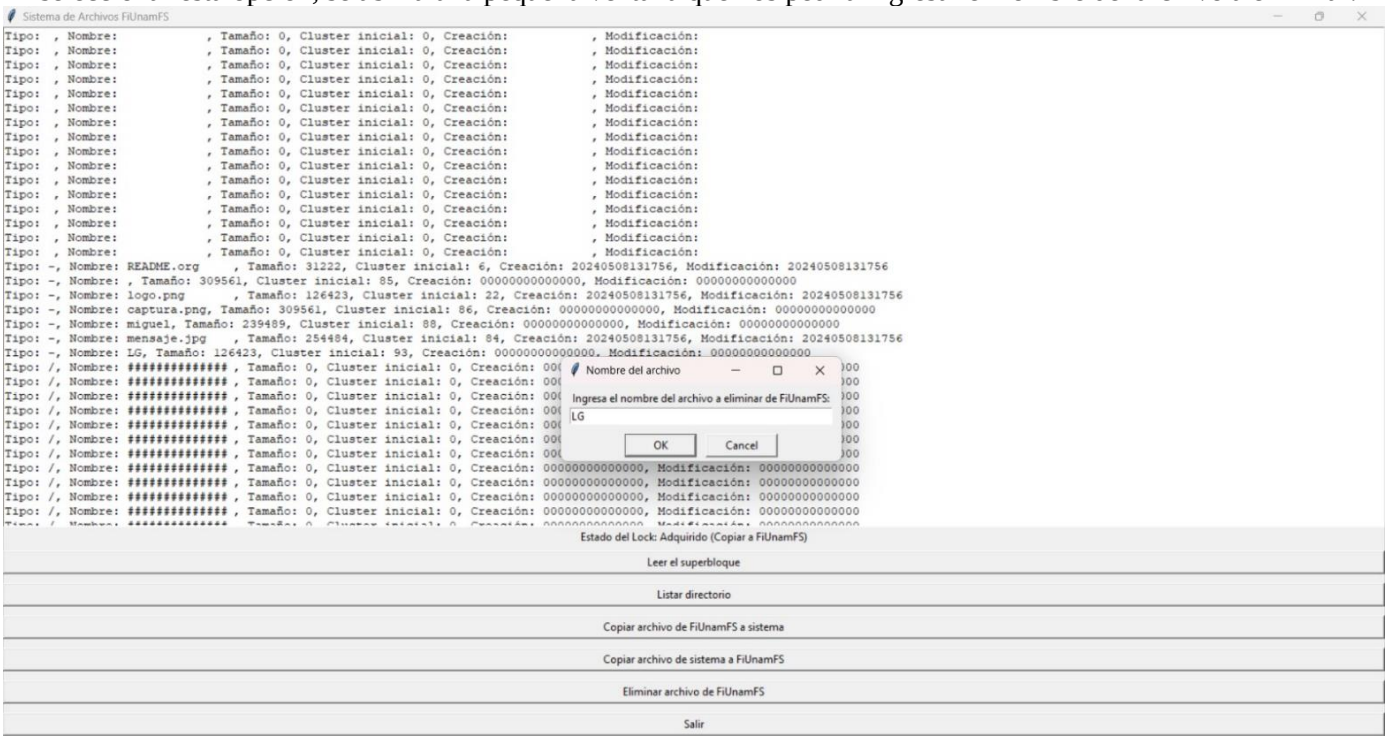


Ahora listamos nuevamente en la opción "*Listar directorio*"

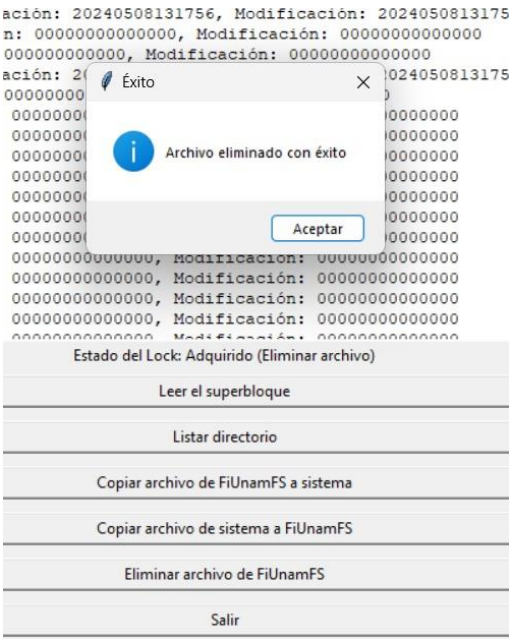


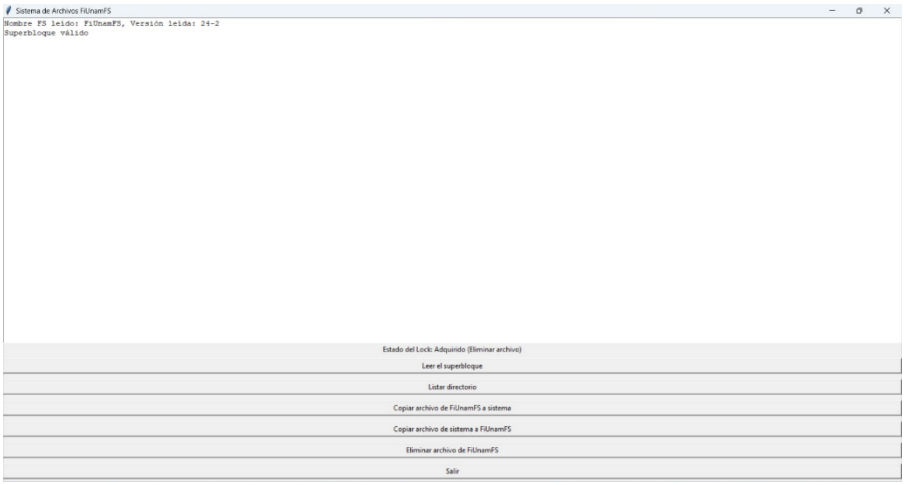
Dentro del recuadro rojo, podemos observar que se copió correctamente el archivo.

Finalmente, para comprobar la última acción que puede realizar nuestro programa "*Eliminar archivo de FiUnamFS*", eliminaremos el archivo que acabamos de copiar cuyo nombre es "LG":
Al seleccionar esta opción, se abrirá una pequeña ventana que nos pedirá ingresar el nombre del archivo a eliminar:

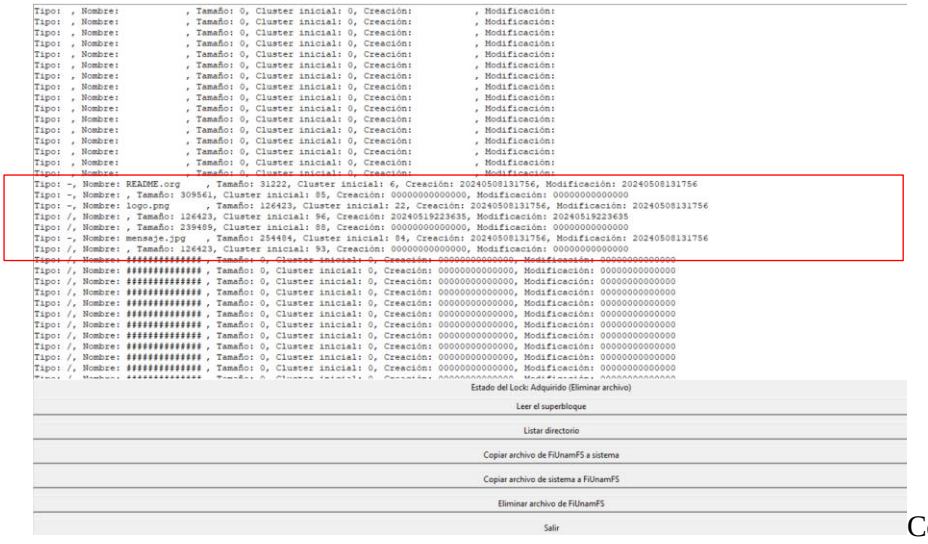


Cuando se realice la operación, el programa nuevamente muestra un mensaje que confirma que se ha eliminado el archivo y con ello, el estado del lock cambiará a “Eliminar archivo”





Finalmente, para comprobar que el archivo fue eliminado, releemos el superbloque y listamos el directorio nuevamente:



Como podemos observar, el archivo se

eliminó exitosamente:

4. Conclusiones

- **Hernández Ramírez Miguel Ángel:**

  El desarrollar un sistema de archivos como el que se ha creado en este proyecto fue una tarea demandante y un poco estresante para mí.
  El estructurar cómo debía funcionar el proyecto y posteriormente, implementarlo de manera programada fue un desafío muy grande, en especial el tema de concurrencia con hilos.
  Tuvimos que investigar un poco cómo se manejaba ese tema y realizar algunos programas piloto para poder llegar al proyecto final que se presenta y se documenta aquí.
  A pesar de que tuve, personalmente, momentos de frustración cuando intentaba programar funciones e implementar algunas ideas que me surgían y no funcionaban como lo esperaba, estoy satisfecho con el programa entregado pues fue resultado de un proceso de programación de días y de aplicar nuestros conocimientos adquiridos en la clase.

- **López Tavera Alexa Fernanda:**

  Para mi fue un proyecto que a primera vista se veía muy complicado y complejo; conforme fuimos desarrollando por partes lo que debía hacer el proyecto, me pareció menos abrumador.
  El plasmar en código las funciones que queríamos que realizará el proyecto fue lo más complicado, pero una vez iniciada la parte de programación, nos ayudó a darnos una idea de qué podíamos modificar o implementar en otras funciones y así, ir construyendo el "esqueleto" del proyecto en la que nos íbamos basando para construir el cuerpo completo.
  La parte gráfica considero que fue una buena práctica para mi pues usualmente hacía mis prácticas en terminal y me costaba mucho personalmente entender cómo modificar los archivos ahí; el desarrollar el front end de este proyecto con ayuda de mi compañero me hizo darme cuenta de lo practico que es para el usuario tener una forma visual para usar el programa y en general, cualquier programa computacional.
  Sinceramente, le agradezco mucho a mi compañero su orientación y su paciencia para conmigo en este proyecto, pues a pesar de que tenía los conocimientos de clase, me ayudó mucho aprender de su forma de programar.

5. Referencias bibliográficas

De las imágenes utilizadas:

1. Threads In Python Presentation. (s.f). Slidemake [Imagen]. Disponible en: https://www.slidemake.com/presentation/Threads-in-python-64d28fc882db6478d9c49e9f

De la información consultada:

1. *The Python Standard Library*. (s. f.). Python Documentation. https://docs.python.org/3/library/index.html

2. CodigoFacilito. (2018, 6 febrero). *Hilos en Python - Bytes* [Vídeo]. YouTube. https://www.youtube.com/watch?v=J9wOU5uWrjw

3. PyAr - Python Argentina. (2021, 28 octubre). PyConAr 2021 - *Concurrencia y paralelismo en Python: Multithreading vs Multiprocessing vs Async* [Vídeo]. YouTube. https://www.youtube.com/watch?v=u77Az26bFPA

4. Sistemas Operativos 2024-05-09: *Sistemas de archivos. (2024, 9 mayo)*. [Vídeo]. Gunnar Wolf. https://www.youtube.com/live/UHTWk5q3IOk?si=NicEnJR4Z9-8hSxs

5. Python, R. (2023, 30 enero*). Python GUI programming with Tkinter*. https://realpython.com/python-gui-tkinter/