



**UNIVERSIDAD NACIONAL AUTONOMA  
DE MEXICO  
FACULTAD DE INGENIERIA**



**PROFESOR**

**ING. Gunnar Eyal Wolf Iszaevich**

**INTEGRANTES**

**Figueroa Solano Carlos Enrique**

**Quintana López Ernesto**

**SISTEMAS OPERATIVOS**

---

**PROYECTO**

**Sistema de archivos multihilos**

---

**Grupo 06**

**Semestre 2024-2**

**19/05/2023**

## INDICE

<b>Planteamiento</b> .....	2
<b>Requisitos solicitados</b> .....	2
<b>Funcionamiento</b> .....	2
<b>Preparación</b> .....	2
<b>Funciones principales</b> .....	5
<i>listar_contenidos()</i> .....	5
<i>copiar_archivo_a_FiUnamFs()</i> .....	5
<i>asignar_espacio()</i> .....	7
<i>copiar_archivo_a_sistema()</i> .....	7
<i>eliminar_archivo()</i> .....	8
<i>Hilos</i> .....	9
<b>Interfaz gráfica</b> .....	10
<i>Código</i> .....	10
<b>Ejemplos de uso</b> .....	11
<i>Opciones</i> .....	11

## Planteamiento

Tenemos un archivo que almacena datos específicos, pero no se puede acceder a él de manera convencional en la computadora. Este archivo simula un disco y se denomina "FiUNAMFS". En su interior, encontramos varios documentos. Nuestra principal tarea es la gestión de estos archivos: eliminar, agregar desde fuentes externas al disco, y copiar desde el disco a fuera. Para realizar estas operaciones, implementaremos dos hilos de ejecución para mejorar la eficiencia y la capacidad de respuesta del sistema.

## Requisitos solicitados

Los requisitos del programa fueron los siguientes:

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.

Aparte de esto decidimos implementar una interfaz grafica muy sencilla, esto para hacer más cómodo su uso del programa.

Todo se hizo en Python 3 con el uso de las siguiente bibliotecas :os, struct, tkinter, threading, math, datetime y time

## Funcionamiento

### Preparación

Para que el programa funcione se debe revisar que el disco a utilizar sea el correcto, en este caso el utilizado sera "FiUnamFS", esto se verificara en la función leer\_superbloque(), con ella se revisa la imagen de disco, extrayendo el nombre de él y la versión, si el los datos no son iguales a los solicitados el programa no seguirá ejecutándose

```
def leer_superbloque():
    global sistema_archivos
    with open(sistema_archivos, 'rb') as file:
        file.seek(0)
        #Leer el primer cluster que es el superbloque
        superbloque = file.read(1024)
        #Extraer información del superbloque
        nombre = superbloque[0:8].decode().strip('\x00')
        version = superbloque[10:15].decode().strip('\x00')
        #Revisar validez del archivo
        if nombre != "FiUnamFS":
            raise ValueError("El sistema de archivos no es FiUnamFS")
        if version != "24-2":
            raise ValueError("Versión del sistema de archivos no compatible")
```

Imagen 1. Código de leer\_superbloque()

Debemos tener en cuenta que la variable `sistema_archivos`, debe tener solamente el nombre del archivo con el que vamos a trabajar, es necesario que el archivo esté en la misma ruta en donde se encuentra el código, para que a la hora de ejecutarse no se presente ningún tipo de problema.

```
#Ruta del sistema de archivos
sistema_archivos = "fiunamfs.img"
#Lista de datos de los archivos a imprimir
archivo_sist=[]
```

Imagen 2. Variable de la ruta de SA y lista para Archivos a imprimir

## Lectura

Para la obtención de la información debemos tener en cuenta que de que maneras se puede acceder a ella, en este caso es de tres maneras:

- Cadenas de texto en formato ASCII-8: Se decodifica mediante LATIN-1 y con ayuda de la función `decode()`

```
def leer_ascii(cabecal,tam):
    global sistema_archivos
    with open(sistema_archivos,'rb') as file:
        file.seek(cabecal)
        #Leemos la información y la decodificamos en Latin-1 -> ASCII 8 bits
        contenido = file.read(tam).decode('Latin-1')
        return contenido
```

Imagen 3. Código de `leer_ascii()`

- Valores entero codificado en hexadecimales de 4 bytes :Se utiliza la función `unpack` de la biblioteca `struct`

```
def leer_enteros(cabecal,tam):
    global sistema_archivos
    #Abrir el sistema de archivos
    with open(sistema_archivos,'rb') as file:
        #Ubicar el cabecal
        file.seek(cabecal)
        contenido = file.read(tam)
        #Usamos unpack para la representación en 32 bits
        contenido, *resto = struct.unpack('<I',contenido)
```

Imagen 4. Código de `leer_enteros()`

- Información en bytes: Esto para leer la información dada en bruto que se encuentra dentro del sistema

```
def leer_info(cabecal,tam):
    global sistema_archivos
    with open(sistema_archivos,'rb') as file:
        file.seek(cabecal)
        #Leemos la información
        contenido = file.read(tam)
        return contenido
```

Imagen 5. Código de `leer_info()`

## Escritura

También tenemos sus funciones de escritura, estas son lo mismo solo que cambia el `file.read()` a `file.write()`



```
def escribir_info(cabezal, contenido):
    global sistema_archivos
    global num_entradas
    with open(sistema_archivos, 'rb+') as file:
        file.seek(cabezal)
        file.write(contenido)

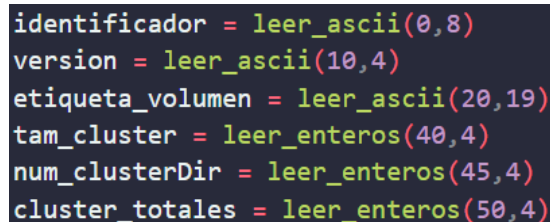
def escribir_ascii(cabezal, contenido):
    global sistema_archivos
    with open(sistema_archivos, 'rb+') as file:
        file.seek(cabezal)
        #Escribir el contenido
        file.write(contenido.encode('Latin-1'))

def escribir_enteros(cabezal, contenido):
    global sistema_archivos
    with open(sistema_archivos, 'rb+') as file:
        file.seek(cabezal)
        #Usamos pack para la representación en 32 bits
        file.write(struct.pack('<I', contenido))
```

Imagen 6. Funciones de escritura

## Variables locales con información del sistema de archivos

Las instrucciones indican que el primer clúster del pseudodispositivo es el superbloque. Este superbloque contiene, en sus primeros bytes, información útil para la ejecución del programa.



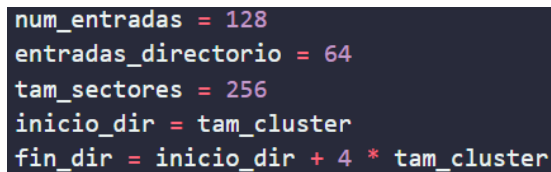
```
identificador = leer_ascii(0,8)
version = leer_ascii(10,4)
etiqueta_volumen = leer_ascii(20,19)
tam_cluster = leer_enteros(40,4)
num_clusterDir = leer_enteros(45,4)
cluster_totales = leer_enteros(50,4)
```

Imagen 7. Variables con información del SA

Cada variable contiene las siguientes cosas:

- `identificador`: El nombre del sistema de archivos en este caso siendo “FiUnamFS”.
- `version`: Versión de la implementación, siendo la “24.2”
- `etiqueta_volumen`: Etiqueta del volumen
- `tam_cluster`: Tamaño del cluster en bytes, siendo el tamaño para todos los clusters usado, permitiéndonos utilizar el cabezal de manera correcta para la lectura y escritura de la información. Teniendo un tamaño de 2048.
- `num_clusterDir`: Numero de clusters que mide el directorio
- `cluster_totales`: Numero de clusters que mide la unidad completa

Posteriormente al obtener la información del sistema, se analizó la información del directorio raíz proporcionada en el apartado de “Especificaciones de FiUnamFS”.



```
num_entradas = 128
entradas_directorio = 64
tam_sectores = 256
inicio_dir = tam_cluster
fin_dir = inicio_dir + 4 * tam_cluster
```

Imagen 8. Variables del directorio

Los valores de cada variable fueron obtenidos de la siguiente manera:

- `num_entradas`: El número de las entradas representa la cantidad de archivos que puede haber en un sistema. Para poder saber esta cantidad debemos tener en cuenta que el directorio toma 4 cluster en donde cada entrada tiene 64 bytes y que el tamaño del cluster es de 2048, teniendo esto aplicamos la siguiente operación

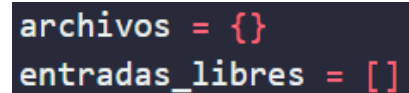
$$num_{entradas} = 4 * \frac{2048}{64} = 128$$

- `entradas_directorio`: Almacena el tamaño de las entradas
- `tam_sectores`: El tamaño de sectores se brinda en el planteamiento del problema
- `inicio_dir` y `fin_dir`: En las variables que representan el inicio y final del directorio se obtiene mediante el planteamiento que define el intervalo del primer al cuarto cluster. Para estar seguros de esto se ocupa la función de `leer_info()` y los intervalos de interés para mostrar el contenido del archivo, viendo que en ese intervalo se podría observar que las 128 entradas del directorio .

## Funciones principales

Las variables `archivos` y `entradas_libres` se utilizarán a lo largo de todo el programa, ya que son elementos clave para su correcto funcionamiento.

- `archivos`: Se encarga de almacenar la información importante de los archivos presentes en el directorio. Además, se le añade un dato extra para identificar el clúster específico del archivo dentro del directorio.
- `entradas_libres`: Determina los clústers del directorio donde se puede ingresar la información de un archivo nuevo. Esto simplifica la tarea de copiar un archivo del sistema al directorio FiUnamFS.

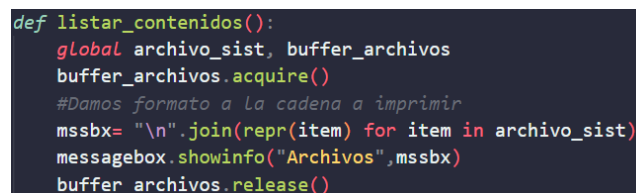


```
archivos = {}
entradas_libres = []
```

*Imagen 9. Almacenadores*

## listar\_contenidos()

Muestra en cierto formato la información de los archivos almacenados en FiUnamFS, esto mediante una ventana emergente



```
def listar_contenidos():
    global archivo_sist, buffer_archivos
    buffer_archivos.acquire()
    #Damos formato a la cadena a imprimir
    mssbx= "\n".join(repr(item) for item in archivo_sist)
    messagebox.showinfo("Archivos",mssbx)
    buffer_archivos.release()
```

*Imagen 10. Código de listar\_contenido()*

## copiar\_archivo\_a\_FiUnamFs()

Para que la función logre transferir un archivo a FiUnamFs, se deben cumplir varios requisitos:

- El directorio debe tener entradas libres.
- La ruta del archivo debe ser válida.
- El tamaño del archivo no debe sobrepasar el espacio disponible en el almacenamiento del sistema. El sistema dispone de 715 clústeres con un tamaño de 2048 bytes cada uno. Este

cálculo se basa en que el superbloque ocupa 1 clúster, el directorio ocupa 5 clústeres, y el sistema total consta de 720 clústeres.

- El nombre del archivo no debe tener más de 14 caracteres.

Si se cumplen todos estos requisitos, se procede a buscar el espacio necesario para almacenar el archivo. Esta tarea se realiza con una función llamada `asignar_espacio()`.

Una vez encontrado el espacio adecuado, se procede a escribir el archivo. Utilizando las funciones `escribir_directorio` y `escribir_info`, se escribe toda la información necesaria tanto en el espacio seleccionado del directorio como en el archivo. Estas funciones aseguran que todos los metadatos del archivo, así como su contenido, se almacenen correctamente en el sistema FiUnamFs.

```
def escribir_dir(nombre,tam,cabecal,fecha_modificacion,fecha_creacion):
    global sistema_archivos
    global num_entradas
    num_entradas -= 1
    nombre = nombre.ljust(14)
    directorio = entradas_libres.pop(0)
    escribir_ascii(directorio, '-')
    escribir_ascii(directorio + 1, nombre)
    escribir_enteros(directorio + 16, tam)
    escribir_enteros(directorio + 20, ceil(cabecal/tam_cluster))
    escribir_ascii(directorio + 24, fecha_creacion)
    escribir_ascii(directorio + 38, fecha_modificacion)

def escribir_info(cabecal, contenido):
    global sistema_archivos
    global num_entradas
    with open(sistema_archivos, 'rb+') as file:
        file.seek(cabecal)
        file.write(contenido)
```

Imagen 11. Funciones de escritura

```
def copiar_archivo_a_FiUnamFs():
    #Verificar que el directorio tenga entradas libres
    global buffer_archivos
    buffer_archivos.acquire()
    if (len(archivos) == num_entradas):
        messagebox.showinfo("ERROR", "No se puede agregar más archivos al directorio")
        buffer_archivos.release()
        return
    else:
        archivo_sistema = filedialog.askopenfilename()
        #Es necesario validar la ruta y la restricción de tamaño
        if os.path.exists(archivo_sistema):
            try:
                with open(archivo_sistema, "rb") as file:
                    nombre = os.path.basename(archivo_sistema)
                    #Que el nombre del archivo no exceda los 14 caracteres.
                    if len(nombre) > 14:
                        messagebox.showinfo("ERROR", "El nombre del archivo es demasiado largo para el sistema.")
                        buffer_archivos.release()
                        return
                    #El archivo no puede superar los (716 * tam_cluster)
                    tam = os.path.getsize(archivo_sistema)
                    if tam > (cluster_totales - num_clusterDir - 1) * tam_cluster:
                        messagebox.showinfo("ERROR", "El tamaño del archivo es demasiado grande para el sistema.")
                        buffer_archivos.release()
                        return
                    fecha_modificacion = str(datetime.fromtimestamp(os.path.getmtime(archivo_sistema)))[0:19].replace("-", "").replace(" ", "").replace(":", "")
                    fecha_creacion = str(datetime.fromtimestamp(os.path.getctime(archivo_sistema)))[0:19].replace("-", "").replace(" ", "").replace(":", "")
                    # En este punto, ya se dispone de la información. Es necesario analizar si hay suficiente espacio para la asignación de memoria.
                    cluster_inicial = asignar_espacio(tam)
                    if cluster_inicial == False:
                        messagebox.showinfo("ERROR", "No hay suficiente espacio de almacenamiento para el archivo seleccionado")
                        buffer_archivos.release()
                        return
                    else:
                        #Se escribe el archivo
                        contenido = file.read()
                        #Se escribe en el espacio para archivos. También hay que actualizar el directorio
                        escribir_dir(nombre, tam, cluster_inicial, fecha_modificacion, fecha_creacion)
                        escribir_info(cluster_inicial, contenido)
                        messagebox.showinfo("Aviso", "Archivo copiado exitosamente")
                        buffer_archivos.release()
                        return
            except:
                messagebox.showinfo("ERROR", "No fue posible abrir el archivo")
                buffer_archivos.release()
                return
        else:
            messagebox.showinfo("ERROR", "No se encontró la ruta")
            buffer_archivos.release()
            return
```

Imagen 12. Código de `copiar_archivo_a_FiUnamFs()`

### asignar\_espacio()

En esta función, primero se obtiene la información de almacenamiento de los archivos actuales dentro del sistema. Para esto, se almacena el clúster inicial y final de cada archivo, ordenándolos de manera ascendente.

- Si el clúster actual coincide con la posición del primer archivo, se debe saltar al final de esa posición.
- Si el clúster actual más los clústeres necesarios sobrepasan la posición inicial de otro archivo, se debe saltar hasta el final de dicho archivo.
- En caso de que no haya más archivos, se debe verificar si el archivo cabe dentro de los límites del tamaño del sistema.

Este proceso asegura que se encuentre un espacio adecuado para almacenar el nuevo archivo sin sobrescribir ningún archivo existente.

```
def asignar_espacio(tam):
    cluster_necesarios = ceil(tam/tam_cluster)
    almacenamiento = []
    #Comienza por el cluster siguiente al directorio
    cluster = 5
    for archivo in archivos.items():
        almacenamiento.append((archivo[1]['cluster_inicial'], archivo[1]['cluster_inicial'] + ceil(archivo[1]['tam'] / tam_cluster)))
    almacenamiento.sort()
    while(cluster < 720):
        #Si el cabezal se encuentra con un clúster que ya está en uso, lo pasa por alto
        if len(almacenamiento) != 0 and cluster == almacenamiento[0][0]:
            cluster = almacenamiento[0][1] + 1
            #Se saca el cluster
            almacenamiento.pop(0)
        else:
            #Colisiona con otro archivo
            if len(almacenamiento) != 0 and (cluster + cluster_necesarios > almacenamiento[0][0]):
                cluster = almacenamiento[0][1] + 1
                almacenamiento.pop(0)
            #Se puede almacenar el archivo
            else:
                return cluster * tam_cluster
        #Se puede almacenar el archivo despues de los demas
        if len(almacenamiento) == 0 and (cluster + cluster_necesarios) < 720:
            return cluster * tam_cluster
        else:
            return False
    return False
```

Imagen 13. Código de asignar\_espacio()

### copiar\_archivo\_a\_sistema()

Esta función tiene como finalidad copiar un archivo desde FiUnamFs a una ubicación en nuestro almacenamiento. Al ejecutar esta función, se solicitará la ruta donde deseamos pegar el archivo y el nombre del archivo a copiar.

- Con los datos proporcionados, se verificará si la ruta dada existe y si el nombre del archivo a copiar es válido.
- Si los datos son válidos, se procede a copiar el contenido del archivo utilizando la función leer\_info. Esta función sitúa el cabezal en el clúster inicial y guarda los datos importantes del archivo, incluyendo su tamaño.
- Finalmente, se crea el archivo en la ruta proporcionada y se escribe la información del archivo copiado desde FiUnamFs.



```
def copiar_archivo_a_sistema():
    ruta = filedialog.askdirectory()
    ruta.rstrip().lstrip()
    nombre = simpdialog.askstring("Copiar archivo", "Ingrese el nombre del archivo a copiar:")
    nombre.rstrip().lstrip()
    #Validamos que el archivo exista en FiUnamfs
    if nombre in archivos:
        #Si no existen archivos con el mismo nombre en la ubicación especificada, procedemos a copiar el archivo
        if (os.path.exists(ruta + "/" + nombre) == False):
            #Almacenamos el contenido del archivo en una variable
            contenido = leer_info(archivos[nombre]['cluster_inicial'] * tam_cluster, archivos[nombre]['tam'])
            #Generamos y escribimos un archivo en la ruta especificada con la información recopilada
            with open(ruta + "/" + nombre, 'wb') as archivo:
                archivo.write(contenido)
            messagebox.showinfo("Aviso", "Archivo guardado con éxito")
        else:
            messagebox.showinfo("ERROR", "Un archivo del mismo nombre está en el directorio")
    else:
        messagebox.showinfo("ERROR", "No existe un archivo con ese nombre")
```

Imagen 14. Código de `copiar_archivo_a_sistema()`

### `eliminar_archivo()`

Para eliminar un archivo de FiUnamFs, primero se verifica si el archivo existe. Si existe, se obtiene toda la información del archivo. Luego, se elimina el contenido y se actualiza el directorio para reflejar la eliminación. Finalmente, se actualiza el diccionario de archivos para que el archivo eliminado ya no se muestre.

Esto se logra utilizando dos funciones adicionales: `eliminar_directorio` y `eliminar_info`. Estas funciones aseguran que tanto el contenido del archivo como su entrada en el directorio sean eliminados correctamente

```
def eliminar_dir(cabeza):
    global num_entradas
    global entradas_libres
    entradas_libres.append(cabeza)
    entradas_libres.sort()
    num_entradas += 1
    escribir_ascii(cabeza, '/.....')
    escribir_ascii(cabeza + 24, '000000000000000000000000')
    with open(sistema_archivos, 'rb+') as file:
        file.seek(cabeza + 16)
        file.write(b'\x00' * 9)
        file.seek(cabeza + 52)
        file.write(b'\x00' * 12)

def eliminar_info(cabeza, tam):
    with open(sistema_archivos, 'rb+') as file:
        file.seek(cabeza)
        file.write(b'\x00' * tam)
```

Imagen 15. Funciones de apoyo para `eliminar_archivo()`

```
def eliminar_archivo():
    global buffer_archivos
    #Tomamos el candado para evitar conflictos, Lo liberamos al salir de la función
    buffer_archivos.acquire()
    nombre = simpdialog.askstring("Eliminar archivo", "Ingrese el nombre del archivo a eliminar:")
    # Validamos que el archivo exista
    if nombre in archivos:
        informacion = archivos[nombre]
        # Eliminamos toda información del archivo
        eliminar_dir(informacion['cluster_directorio'])
        eliminar_info(informacion['cluster_inicial'] * tam_cluster, informacion['tam'])
        buffer_archivos.release()
        messagebox.showinfo("Aviso", "Archivo eliminado exitosamente")
        return
    else:
        buffer_archivos.release()
        messagebox.showinfo("ERROR", "No existe un archivo con ese nombre")
        return
```

Imagen 16. Código de `eliminar_archivo()`

## Hilos

Al ejecutar el código, lo primero que se ejecuta son los dos hilos (daemon) para cargar los datos de los archivos y guardar los cambios hechos en los archivos, así como el candado, posteriormente se espera 1 segundo para cargar el menú.

```
# INICIO
#Hilo para guardar la información de los archivos
Thread(target=guardar_info_archivos,daemon=True).start()
buffer_archivos=Lock()
#Hilo para cargar la información de los archivos para imprimir
Thread(target=cargar_contenidos, daemon=True).start()
#Variable de control (Evita que se modifique el diccionario de archivos mientras se lee)

#Permite cargar los datos antes de desplegar el menu
sleep(1)
mostrar_menu()
```

Imagen 17. Hilos y Lock

El primer hilo hace referencia a la función `guardar_info_archivos()`, la cual es un ciclo while infinito, que se ejecuta cada 0.1 segundos, no utiliza el candado.

La función, en esencia, lee cada uno de los archivos y va actualizando su información en nuestro diccionario `archivos{}`.

```
def guardar_info_archivos():
    """
    Se mostrarán únicamente los archivos que tienen un nombre específico
    Se deberá de recorrer el directorio
    Ya que es un hilo, guardará constantemente los datos
    """
    while True:
        cabecal = inicio_dir
        global num_entradas
        global archivos
        archivos.clear()
        num_entradas = 128
        #Guarda la informacion de los archivos en el diccionario 'archivos'
        while(cabecal != fin_dir):
            archivo = {}
            with open(sistema_archivos,'rb') as file:
                file.seek(cabecal)
                #Comprueba si la entrada tiene algun contenido o esta vacia
                entrada = leer_ascii(cabecal,1)
                if entrada == '.':
                    #Lee la informacion del archivo por partes
                    archivo['nombre'] = leer_ascii(cabecal + 1, 14) #Nombre
                    archivo['tam'] = leer_enteros(cabecal + 16, 4) # Tamaño
                    archivo['cluster_inicial'] = leer_enteros(cabecal + 20, 4) #Tamaño del cluster
                    fecha_objeto = datetime.strptime(leer_ascii(cabecal + 24, 13), "%Y%m%d%H%M%S")
                    cadena_formateada = fecha_objeto.strftime("%Y-%m-%d %H:%M:%S")
                    archivo['fecha_c'] = cadena_formateada #fecha de creación del archivo
                    fecha_objeto = datetime.strptime(leer_ascii(cabecal + 38, 13), "%Y%m%d%H%M%S")
                    cadena_formateada = fecha_objeto.strftime("%Y-%m-%d %H:%M:%S")
                    archivo['fecha_m'] = cadena_formateada #fecha de modificacion del archivo
                    archivo['cluster_directorio'] = cabecal
                    #Se guarda la informacion recabada de los archivos
                    archivos[archivo['nombre'].rstrip()] = archivo
                    cabecal += 64
                    num_entradas -= 1
                    pass
                else:
                    #Es ignorado y deja avanzar el cabecal
                    entradas_libres.append(cabecal)
                    cabecal += 64
            #Los datos se guardan cada 0.1 segundos
            sleep(.1)
```

Imagen 18. Hilo 1 – `guardar_info_archivos()`

El segundo hilo hace referencia a la función `cargar_contenidos()`, la cual consta de un ciclo while infinito que se ejecuta cada segundo, primero requiere obtener el candado, y al terminar sus

procesos, libera el candado por 1 segundo, para que el resto de funciones, como listar\_contenido(), puedan adquirirlo, con el fin de evitar conflictos al modificar el diccionario de archivos.

La función, esencialmente, guarda los datos de cada uno de los archivos de nuestro diccionario archivo{} en la lista archivo\_sist[].

```
def cargar_contenidos():
    # Se cargarán los archivos que tienen un nombre
    global buffer_archivos
    global archivo_sist
    buffer_archivos.acquire()
    while True:
        #Limpiamos la lista para que se borren los archivos que hayan sido eliminados
        archivo_sist.clear()
        #Guardamos los datos de los archivos en una lista
        for archivo in archivos.items():
            archivo_sist.append(f"-{archivo[0]:<14}|{archivo[1]['tam']:<10}|{archivo[1]['fecha_m']}|{archivo[1]['fecha_m']}")
        #Permitimos que otro proceso (que modifique la lista) tome el candado durante una ventana de 1s
        buffer_archivos.release()
        sleep(1)
        buffer_archivos.acquire()
```

Imagen 19. Hilo 2 – cargar\_contenidos()

La función salir\_programa() se creó para dejar a los procesos correr 0.2 segundos más, en realidad no afecta, pues las variables como el diccionario de archivo, se almacenan en la RAM, sin embargo, si se almacenasen en el disco, lo ideal sería dejar que los procesos terminen antes de cerrar el programa.

```
def salir_programa():
    global root
    #Esperamos 0.2s para permitir el guardado de la información de los archivos
    sleep(0.2)
    root.quit()
```

Imagen 20. Función salir\_programa()

## Interfaz gráfica

### Código

Para la interfaz se utilizó la biblioteca Tkinter, realmente no hay mucho que explicar, root es un objeto de tipo Tkinter, podemos agregar botones dentro de la misma función, por lo que aparecerán dentro del objeto root, para leer datos tipo string (nombres de archivos) usamos simpledialog, mientras que para leer archivos o directorios usamos filedialog, para mostrar mensajes se usó messagebox.

En general, la biblioteca se encarga de hacer prácticamente todo lo demás.

```
def mostrar_menu():
    root = tk.Tk()
    root.title("Sistema de Archivos FIUNAMFS")
    root.geometry("400x400")

    # Crear botones para cada operación
    btn_listar_contenidos = tk.Button(root, text="listar contenidos", command=listar_contenidos)
    btn_listar_contenidos.pack(pady=10)

    btn_copiar_a_fiunamfs = tk.Button(root, text="Copiar archivo del sistema a FIUNAMFS", command=copiar_archivo_a_FiUnamFs)
    btn_copiar_a_fiunamfs.pack(pady=10)

    btn_copiar_a_sistema = tk.Button(root, text="Copiar archivo de FIUNAMFS al sistema", command=copiar_archivo_a_sistema)
    btn_copiar_a_sistema.pack(pady=10)

    btn_eliminar_archivo = tk.Button(root, text="Eliminar archivo de FIUNAMFS", command=eliminar_archivo)
    btn_eliminar_archivo.pack(pady=10)

    btn_salir = tk.Button(root, text="Salir", command=root.quit)
    btn_salir.pack(pady=10)

    root.mainloop()
```

Imagen 21. Código del menú

## Ejemplos de uso

Al ejecutar el código nos aparecerán cinco opciones en una ventana llamada “Sistema de Archivos FIUNAMFS”

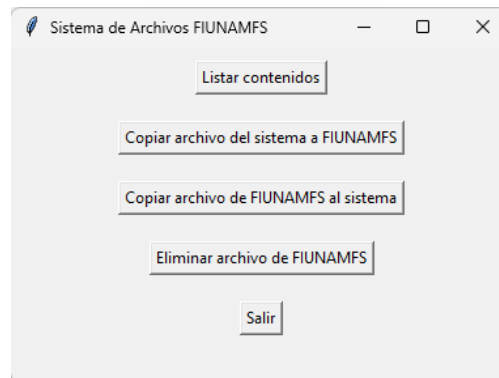


Imagen 22. Interfaz del menú

## Opciones

### Lista contenidos

Al darle click a esta opción nos aparecerá una ventana emergente con todos los archivos que almacena el documento “fiunamfs.img”

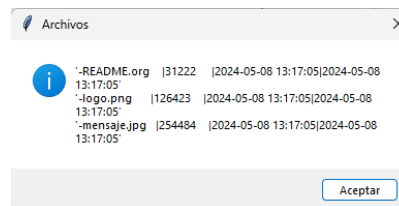


Imagen 23. Lista de archivos almacenados

### Copiar archivo del sistema a FIUNAMFS

Al darle click nos abrirá el explorador de archivos, aquí podremos escoger el archivo que queramos pasar a FIUNAMFS, en este caso será “Prueba.txt”

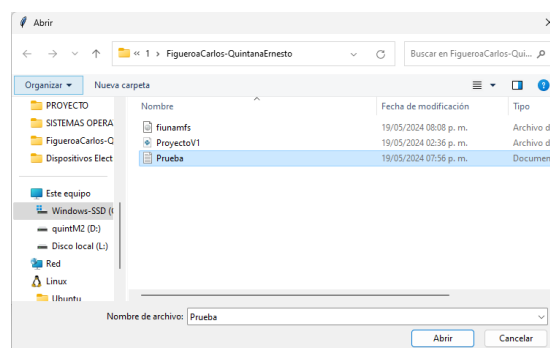


Imagen 24. Seleccionando archivo a copiar

Si todo se hace de manera correcta nos aparecerá la siguiente ventana emergente

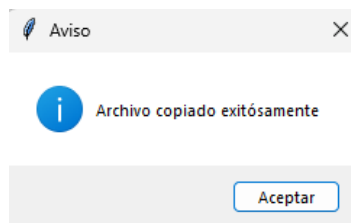


Imagen 25. Ventana emergente exitosa

Comprobamos si el archivo se pasó correctamente

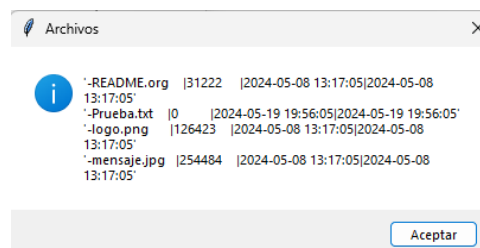


Imagen 26. Lista de archivo con el archivo nuevo

## Copiar FIUNAMFS al sistema

Al darle click a esta opción no dejara escoger el lugar a donde queramos pegar el archivo que ser a copiado de FIUNAMFS, en este caso pegaremos el archivo en la carpeta “FigueroaCarlos-QuintanaErnesto”

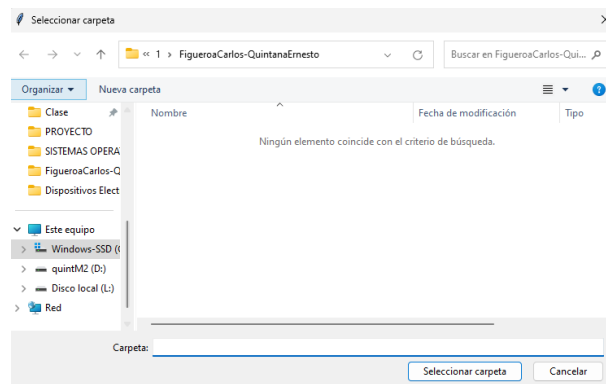


Imagen 27. Seleccionando ruta para pegar el archivo

Después nos saldrá una venta que nos pide el nombre del archivo que deseemos copiar, para esta ocasión será el archivo “logo.png”

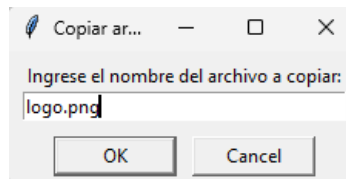


Imagen 28. Ventana que solicita el nombre del archivo

Nos saldrá esta ventana si el nombre del archivo existe

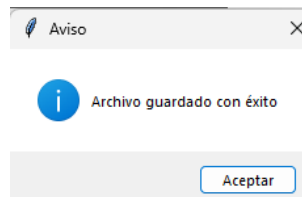


Imagen 29. Ventana exitosa 2

Podemos ver que el archivo se copió correctamente

fiunamfs	19/05/2024 07:58 p. m.	Archivo de imagen de d...	1,440 KB
ProyectoV1	19/05/2024 02:36 p. m.	Archivo de origen Python	16 KB
Prueba	19/05/2024 07:56 p. m.	Documento de texto	0 KB
logo	19/05/2024 08:03 p. m.	Archivo PNG	124 KB

Imagen 16. Carpeta con archivo logo.png

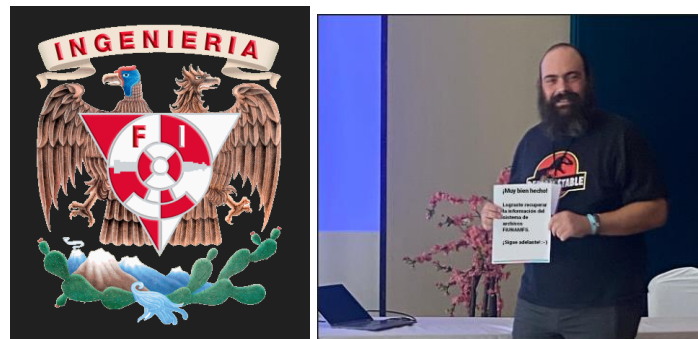


Imagen 30. Contenido de logo.png y mensaje.png

## Eliminar archivo de FIUNAMFS

Al dar click nos pedirá introducir el nombre del archivo que queramos eliminar, para el ejemplo eliminaremos el archivo “Prueba.txt”

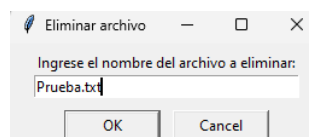
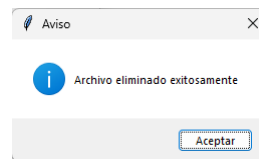
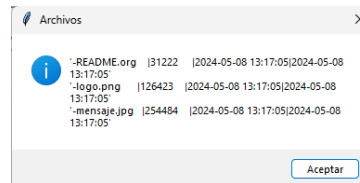


Imagen 31. Nombre del archivo a eliminar



*Imagen 32. Ventana exitosa 3*

Verificamos si el archivo fue eliminado de manera correcta



*Imagen 33. Lista de archivos sin Prueba.txt*

## Salir

Simplemente cierra el programa.