UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

PROYECTO: (MICRO) SISTEMA DE ARCHIVOS MULTIHILOS

SISTEMAS OPERATIVOS

GRUPO: 6

PROFESOR: GUNNAR EYAL WOLF ISZAEVICH

INTEGRANTES:

CRUZ MALDONADO ARMANDO

DÍAS GONZÁLEZ RIVAS ÁNGEL IÑAQUI

FECHA DE ENTREGA:19 DE MAYO DEL 2024

Documentación

REQUISITOS DEL PROGRAMA

Para la solución al problema planteado en este proyecto consiste en un programa realizado en Python donde se busca que se cumplan las siguientes funciones:

1. listar contenidos del directorio
2. copiar uno de los archivos de dentro del microsistema FiUnamFS, hacia nuestro equipo
3. copiar archivos de nuestro ordenador hacia (FiUnamFS)
4. eliminar un archivo de (FiUnamFS)
5. El programa debe de ejecutarse mediante 2 hilos de ejecución.

FUNCIONAMIENTO DEL PROGRAMA

Para el funcionamiento del programa se explicará el código realizado comenzando por el uso de las librerías threading, struct, queue y os. Donde threading será la librería que nos permitirá el uso de hilos en nuestro programa, struct no permite interpretar cadenas de bytes como datos estructurados, utilizando un formato específico, la librería queue proporciona clases para implementar colas esto para el uso compartido de los hilos en el programa y la librería os nos permite trabajar con las funciones para interactuar con el sistema operativo del equipo.

```
import threading
import struct
import queue
import os
```

Comenzamos con la definición de la clase (FiUnamFS), este será nuestra base en este caso el microsistema donde contiene todas las funciones del programa en cuestión, las primeras 4 líneas debajo de la definición de la clase son las definiciones constantes del sistema de archivos que fueron definidos en los planteamientos del programa.

La primera definición def __init__ (self, disk_file), servirá para el uso y el control del micro sistema, aquí se inician las funciones para, los datos de tienen el disco, los datos del super bloque, el control de la sincronización de hilos

La segunda definición def read_sector(self, sector_number) esta será la definición la cual nos permite leer un sector en específico del archivo en disco, además de retornar los datos leídos por el contenido que tenga el disco

La tercera definicion def write_sector(self, sector_number, data) se encarga escribir datos en un sector del disco del micro sistema este no retorna ningún valor solo representa el método de almacenamiento.

```python
class FiUnamFS:
    # Definición de constantes del sistema de archivos
    SECTOR_SIZE = 1024
    CLUSTER_SIZE = 4 * SECTOR_SIZE
    VOLUME_SIZE = 1440 * 1024  # 1440 KB
    FILE_ENTRY_SIZE = 64

    def __init__(self, disk_file):
        # Inicialización de variables y lectura del superbloque
        self.disk_file = disk_file
        self.superblock = self.read_superblock()
        self.lock = threading.Lock()  # Lock para sincronización de hilos
        self.command_queue = queue.Queue()  # Cola de comandos para gestionar operaciones
        self.is_running = True  # Bandera para controlar la ejecución de los hilos

    def read_sector(self, sector_number):
        # Lectura de un sector específico del archivo de disco
        with open(self.disk_file, 'rb') as f:
            f.seek(sector_number * self.SECTOR_SIZE)
            return f.read(self.SECTOR_SIZE)

    def write_sector(self, sector_number, data):
        # Escritura de un sector específico del archivo de disco
        with open(self.disk_file, 'r+b') as f:
            f.seek(sector_number * self.SECTOR_SIZE)
            f.write(data)
```

La definicion def read_superblock(self) se encarga de leer el superbloque del sistema de archivos FiUnamFS desde el primer sector del archivo de disco, comienza con la lectura del superbloque llamando al método read_sector (sector_number) para leer el primer sector del archivo de disco, donde se encuentra el superbloque. A partir de ahí comienza la interpretación del superbloque que interpreta los datos obtenidos en el sector, codifica los bytes del superbloque en cadenas de código ASCII, extrae varios campos del superbloque en este caso las definiciones mencionadas en la definicion de la clase, de este modo pasa al retorno de la información del superbloque el cual devuelve un diccionario que contiene la información extraída del superbloque.

```python
def read_superblock(self):
    # Lectura del superbloque para obtener información del sistema de archivos
    superblock_data = self.read_sector(0)  # Leer el primer sector donde se encuentra el superbloque
    identifier = superblock_data[0:9].decode('ascii').strip('\x00')  # Identificador del sistema de archivos
    version = superblock_data[10:15].decode('ascii').strip('\x00')  # Versión del sistema de archivos
    if identifier != "FiUnamFS":
        raise Exception("El sistema de archivos no es FiUnamFS")
    if version != "24-2":
        raise Exception("La versión del sistema de archivos no es compatible")

    # Extraer información del superbloque
    volume_label = superblock_data[20:36].decode('ascii')
    cluster_size = struct.unpack('<I', superblock_data[40:44])[0]
    directory_clusters = struct.unpack('<I', superblock_data[45:49])[0]
    total_clusters = struct.unpack('<I', superblock_data[50:54])[0]
    direc_inicio = 1 * cluster_size  # Inicio del directorio
    direct_size = 4 * cluster_size  # Tamaño total del directorio
    entrada_direc_size = 64  # Tamaño de cada entrada de directorio

    return {
        'volume_label': volume_label,
        'cluster_size': cluster_size,
        'directory_clusters': directory_clusters,
        'total_clusters': total_clusters,
        'direc_inicio': direc_inicio,
        'direct_size': direct_size,
        'entrada_direc_size': entrada_direc_size
    }
```

La definicion def list_contents(self) se encarga de listar los contenidos del directorio del sistema de archivos, usa el siguiente método primero hace la lectura del directorio, donde a partir de la variable directory_data almacenara los datos del directorio del sistema y con una cadena for va ir almacenando cada sector del directorio a partir de esto comienza la segunda etapa, donde se procesan las entradas del directorio dentro del bucle for, analiza cada entrada de directorio y extrae información importante, como el tipo de archivo, el nombre del archivo, el tamaño del archivo, por decir algunos, y por ultimo pasa para la impresión de la información del bloque este se encarga de mostrar el contenido del directorio.

```
def list_contents(self):
    # Listado de los contenidos del directorio del sistema de archivos
    directory_data = b""
    for i in range(self.superblock['directory_clusters']):
        directory_data += self.read_sector(i + 1)  # Leer todos los sectores del directorio

    entry_size = 64  # Tamaño de cada entrada de directorio
    for i in range(0, len(directory_data), entry_size):
        entry = directory_data[i:i + entry_size]
        file_type = entry[0:1]
        file_name = entry[1:16].decode('ascii').strip()
        file_size, = struct.unpack('<I', entry[16:20])
        cluster_start, = struct.unpack('<I', entry[20:24])
        creation_time = entry[24:38].decode('ascii').strip()
        modification_time = entry[38:52].decode('ascii').strip()

        # Filtrar entradas vacías o no válidas
        if not file_name or file_name == "--------------" or file_name.startswith('.'):
            continue

        if file_size > 0 and cluster_start > 0 and creation_time and modification_time:
            # Mostrar información del archivo
            print(f"ARCHIVO: {file_name}, TAMAÑO: {file_size}, "
                  f"CLUSTER INICIAL: {cluster_start}, "
                  f"FECHA CREACION: {creation_time}, "
                  f"FECHA MODIFICACION: {modification_time}\n")
```

la definición copy_file_to_system(self, filename, destination_path) se encarga de copiar un archivo desde el sistema de archivos FiUnamFS al sistema local este será la función principal para que el sistema copie archivos del micro sistema a cualquier equipo. Primero se encargará de obtener la ruta de origen y destino esto recordando la librería os, usará los comandos filename (obtiene el nombre del archivo a buscar) y destination_path (obtiene la ruta de destino). De ahí pasará a la lectura de datos en el directorio donde será encargado de buscar el nombre del archivo proporcionado en todo el micro sistema de ahí se obtiene la información importante del archivo y lo copia, a partir de aquí comenzara a buscar coincidencias con la ruta de destino, cuando encuentra en el sistema la ruta de destino comienza a sobre escribir información y finaliza el proceso

```
def copy_file_to_system(self, filename, destination_path):
    # Copia un archivo desde el sistema de archivos FiUnamFS al sistema local
    print(f"Copying {filename} to {destination_path}...")
    direc_inicio = self.superblock['direc_inicio']  # Punto de inicio del directorio en el sistema de archivos
    direct_size = self.superblock['direct_size']  # Tamaño total del directorio en bytes
    entrada_direc_size = self.superblock['entrada_direc_size']  # Tamaño de cada entrada de directorio en bytes
    cluster_size = self.superblock['cluster_size']  # Tamaño de un clúster en bytes

    with open(self.disk_file, 'rb') as f:
        f.seek(direc_inicio)  # Mover a la posición del inicio del directorio
        directory_data = f.read(direct_size)  # Leer los datos del directorio
        encontrado = False  # Bandera para indicar si se encuentra el archivo en el directorio

        # Buscar el archivo en el directorio
        for i in range(0, direct_size, entrada_direc_size):
            entrada = directory_data[i:i + entrada_direc_size]  # Obtener la entrada de directorio actual
            nombre_archivo = entrada[1:15].decode().strip()  # Nombre del archivo en la entrada
            tamano_archivo, cluster_start = struct.unpack('<I', entrada[16:20])[0], struct.unpack('<I', entrada[20:24])[0]
            # Tamaño y cluster de inicio del archivo

            # Verificar si la entrada corresponde al archivo buscado
            if nombre_archivo == filename and tamano_archivo > 0:
                encontrado = True
                break

        # Si el archivo no se encuentra en el directorio, mostrar un mensaje y salir
        if not encontrado:
            print(f"Archivo '{filename}' no encontrado en FiUnamFS")
            return

        start_point = cluster_start * cluster_size  # Calcular el punto de inicio del archivo en el disco
        f.seek(start_point)  # Mover el puntero del archivo al inicio del archivo
        datos_archivo = f.read(tamano_archivo)  # Leer los datos del archivo

        # Escribir los datos del archivo en el destino del sistema local
        with open(destination_path, 'wb') as archivo:
            archivo.write(datos_archivo)

        print(f"Archivo '{filename}' copiado a '{destination_path}'.")
```

Para la siguiente definicion def copy_file_to_fiunamfs(self, filename, origin_path) el método se encarga ahora de forma inversa, copia un archivo desde el equipo y lo agrega al micro sistema (FiUnamFS) usa las mismas variables de la instrucción anterior solo que ahora se nombra el archivo a copiar y la ruta en el equipo donde se encuentra dicho archivo, Aquí entra un nuevo método donde se busca el espacio libre en el directorio, en este caso Itera sobre las entradas del directorio para encontrar la primera entrada marcada como libre, A partir de aquí aplica el mismo método de escritura de datos, comienza a partir desde el espacio libre de directorio, finalmente el programa se actualiza conforme se escribe el archivo, modificando el espacio del clúster inicial y se marca el nombre y tamaño del archivo

```
def copy_file_to_fiunamfs(self, filename, origin_path):
    # Copia un archivo desde el sistema local al sistema de archivos FiUnamFS
    with open(self.disk_file, 'r+b') as f:
        tamano_o = os.path.getsize(filename)  # Obtener el tamaño del archivo a copiar
        cluster_libre = 5  # Número de clúster inicial para buscar espacio libre
        posicion_entrada_libre = None  # Inicializar la posición de la entrada de directorio libre

        f.seek(self.superblock['direc_inicio'])  # Mover el puntero al inicio del directorio
        for i in range(self.superblock['direct_size'] // self.FILE_ENTRY_SIZE):
            posicion_actual = f.tell()  # Guardar la posición actual
            entrada = f.read(self.FILE_ENTRY_SIZE)  # Leer una entrada de directorio
            tipo_archivo = entrada[0:1]  # Obtener el tipo de archivo
            cluster_ini = struct.unpack('<I', entrada[20:24])[0]  # Obtener el número de clúster inicial

            # Verificar si la entrada es libre y guardar su posición si es necesario
            if tipo_archivo == b'/' and posicion_entrada_libre is None:
                posicion_entrada_libre = posicion_actual
                print(f"Hay posición en {posicion_entrada_libre}")

            # Actualizar el número de clúster libre máximo encontrado
            if cluster_ini >= cluster_libre:
                cluster_libre = cluster_ini + 1
                print(f"Nuevo cluster libre: {cluster_libre}")
```

```
# Verificar si se encontró espacio libre para el archivo
if posicion_entrada_libre is None:
    raise Exception("No hay espacio")
else:
    print(f"Espacio libre en: {posicion_entrada_libre}, Cluster libre para el archivo: {cluster_libre}")

# Escribir los datos del archivo en el sistema de archivos FiUnamFS
with open(filename, 'rb') as archivo_origen_f:
    data = archivo_origen_f.read()  # Leer los datos del archivo
    f.seek(cluster_libre * self.CLUSTER_SIZE)  # Mover el puntero al inicio del clúster libre
    f.write(data)  # Escribir los datos del archivo en el clúster libre

nombre_destino = os.path.basename(filename)  # Obtener el nombre del archivo
f.seek(posicion_entrada_libre)  # Mover el puntero a la posición de la entrada libre
f.write(b'-' + nombre_destino.ljust(15).encode('ascii'))  # Escribir el nombre del archivo en la entrada
f.write(struct.pack('<I', tamano_o))  # Escribir el tamaño del archivo en la entrada
f.write(struct.pack('<I', cluster_libre))  # Escribir el número de clúster inicial en la entrada
```

Esta siguiente definición def delete_file(self, filename): sirve para eliminar un archivo del microsistema, usa la implementación de filename para buscar el nombre del archivo deseado a eliminar, a partir de aquí usa las iteraciones para que verifique que el archivo se encuentra en el microsistema, si encuentra el archivo este se sobre escribirá en esa dirección para marcar la entrada como libre, y finalizara el proceso en caso contrario que no encuentre el archivo mostrara un mensaje que dirá “Archivo no encontrado en FiUnamFS”

```
def delete_file(self, filename):
    # Elimina un archivo del sistema de archivos FiUnamFS
    direc_inicio = self.superblock['direc_inicio']  # Obtener el inicio del directorio
    entrada_direc_size = self.superblock['entrada_direc_size']  # Obtener el tamaño de la entrada de directorio
    cluster_size = self.CLUSTER_SIZE  # Obtener el tamaño del clúster

    with open(self.disk_file, 'r+b') as f:
        f.seek(direc_inicio)  # Mover el puntero al inicio del directorio
        for _ in range(cluster_size // entrada_direc_size):
            posicion = f.tell()  # Guardar la posición actual
            entrada = f.read(entrada_direc_size)  # Leer una entrada de directorio
            nombre = entrada[1:16].decode('ascii').rstrip()  # Obtener el nombre del archivo

            # Verificar si la entrada corresponde al archivo buscado
            if nombre.rstrip('\x00').strip() == filename.rstrip('\x00').strip():
                f.seek(posicion)  # Mover el puntero a la posición de la entrada
                # Marcar la entrada como eliminada
                f.write(b'/' + b' ' * 15)
                print("Archivo eliminado de FiUnamFS")
                return
    print("Archivo no encontrado en FiUnamFS")
```

La siguiente definicion def get_user_commands(self): es la encargada de mantener la interacción con el usuario en este caso va ser la encargada de mostrar el contenido necesario para que el usuario pueda usar el programa así como la interfaz de usuario, el programa va a obtener los comandos y argumentos necesarios, y ponerlos en la cola de comandos estos serán dirigidos por un hilo que se encargara de ver que las condiciones se cumplan para que realice las opciones marcadas y proceda a realizar las definiciones.

```
def get_user_commands(self):
    # Obtiene comandos del usuario y los pone en la cola de comandos
    while self.is_running:
        print("\nOpciones disponibles:")
        print("1. Listar los contenidos del directorio")
        print("2. Copiar un archivo dentro del FiUnamFS hacia el sistema")
        print("3. Copiar un archivo de la computadora hacia el FiUnamFS")
        print("4. Eliminar un archivo del FiUnamFS")
        print("5. Salir del programa")

        command = input("\nElegir opción (1,2,3,4 o 5): ")  # Solicitar al usuario que elija una opción
        if command == "5":
            self.is_running = False  # Marcar que el programa debe finalizar
            self.command_queue.put(("exit", None))  # Poner una señal de salida en la cola de comandos
            break

        args = None
        if command in ("2", "3", "4"):  # Si el comando requiere argumentos
            file_name = input("Ingrese el nombre del archivo: ")  # Solicitar al usuario el nombre del archivo
            if command == "2":
                destination_path = input("Ingrese la ruta de destino en su sistema (incluyendo el nombre del archivo): ")
                args = (file_name, destination_path)
            elif command == "3":
                origin_path = input("Ingrese la ruta de origen en su sistema (incluyendo el nombre del archivo): ")
                args = (file_name, origin_path)
            else:
                args = (file_name,)
        elif command not in ("1", "5"):
            print("Opción no válida")  # Notificar al usuario si la opción elegida no es válida
            continue

        self.command_queue.put((command, args))  # Poner el comando y sus argumentos en la cola de comandos
```

Esta definicion def process_commands(self) se encarga de realizar de forma sincronizada el procesamiento de comandos los cuales dirigirán las opciones que elija el usuario, esta es dirigida por hilos los cuales se encargaran de sincronizar las opciones para evitar problemas de concurrencia

```
def process_commands(self):
    # Procesa los comandos encolados, asegurando sincronización con lock
    while self.is_running:
        command_args = self.command_queue.get()  # Obtener el siguiente comando de la cola de comandos
        if command_args is None:
            self.list_contents()  # Si no hay comando, listar el contenido del directorio
        else:
            command, args = command_args
            with self.lock:  # Asegurar que solo un hilo ejecute el bloque de código a la vez
                if command == "1":
                    self.list_contents()  # Si el comando es listar, mostrar el contenido del directorio
                elif command == "2":
                    filename, destination_path = args
                    self.copy_file_to_system(filename, destination_path)  # Copiar archivo del FiUnamFS al sistema
                elif command == "3":
                    filename, origin_path = args
                    self.copy_file_to_fiunamfs(filename, origin_path)  # Copiar archivo del sistema al FiUnamFS
                elif command == "4":
                    filename, = args
                    self.delete_file(filename)  # Eliminar archivo del FiUnamFS

        self.command_queue.task_done()  # Indicar que la tarea del comando ha sido completada
```

La última definicion def main(self): es el punto base del programa ya que Coordina la ejecución de hilos para obtener y procesar comandos del usuario de forma concurrente, este dirige cuando comienza un hilo, cuando está en ejecución de hilos monitorea otros para que no se ejecuten mientras los hilos principales están en ejecución.

```
    def main(self):
        # Inicializa y gestiona hilos para obtener y procesar comandos
        producer_thread = threading.Thread(target=self.get_user_commands)  # Hilo para obtener comandos del usuario
        producer_thread.start()  # Iniciar el hilo

        consumer_thread = threading.Thread(target=self.process_commands)  # Hilo para procesar comandos
        consumer_thread.start()  # Iniciar el hilo

        producer_thread.join()  # Esperar a que el hilo de obtención de comandos termine
        consumer_thread.join()  # Esperar a que el hilo de procesamiento de comandos termine

if __name__ == "__main__":
    fiunamfs = FiUnamFS('fiunamfs.img')  # Crear una instancia de FiUnamFS con el archivo de imagen
    fiunamfs.main()  # Llamar al método principal para iniciar el programa
```

EJECUCION DEL CODIGO

Cuando el código se ejecuta muestra la siguiente interfaz:

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\ARMANDO\Downloads\archivos3.py

Opciones disponibles:
1. Listar los contenidos del directorio
2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa

Elegir opción (1,2,3,4 o 5): 1
```

Cuando se elige la opción 1:

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\ARMANDO\Downloads\archivos3.py

Opciones disponibles:
1. Listar los contenidos del directorio
2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa

Elegir opción (1,2,3,4 o 5): 1

Opciones disponibles:ARCHIVO: README.org     , TAMAÑO: 31222, CLUSTER INICIAL: 6, FECHA CREACION: 20240508131756,
FECHA MODIFICACION: 20240508131756


1. Listar los contenidos del directorioARCHIVO: logo.png       , TAMAÑO: 126423, CLUSTER INICIAL: 22, FECHA CREACI
ON: 20240508131756, FECHA MODIFICACION: 20240508131756


2. Copiar un archivo dentro del FiUnamFS hacia el sistemaARCHIVO: mensaje.jpg    , TAMAÑO: 254484, CLUSTER INICIAL
: 84, FECHA CREACION: 20240508131756, FECHA MODIFICACION: 20240508131756


3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa
```

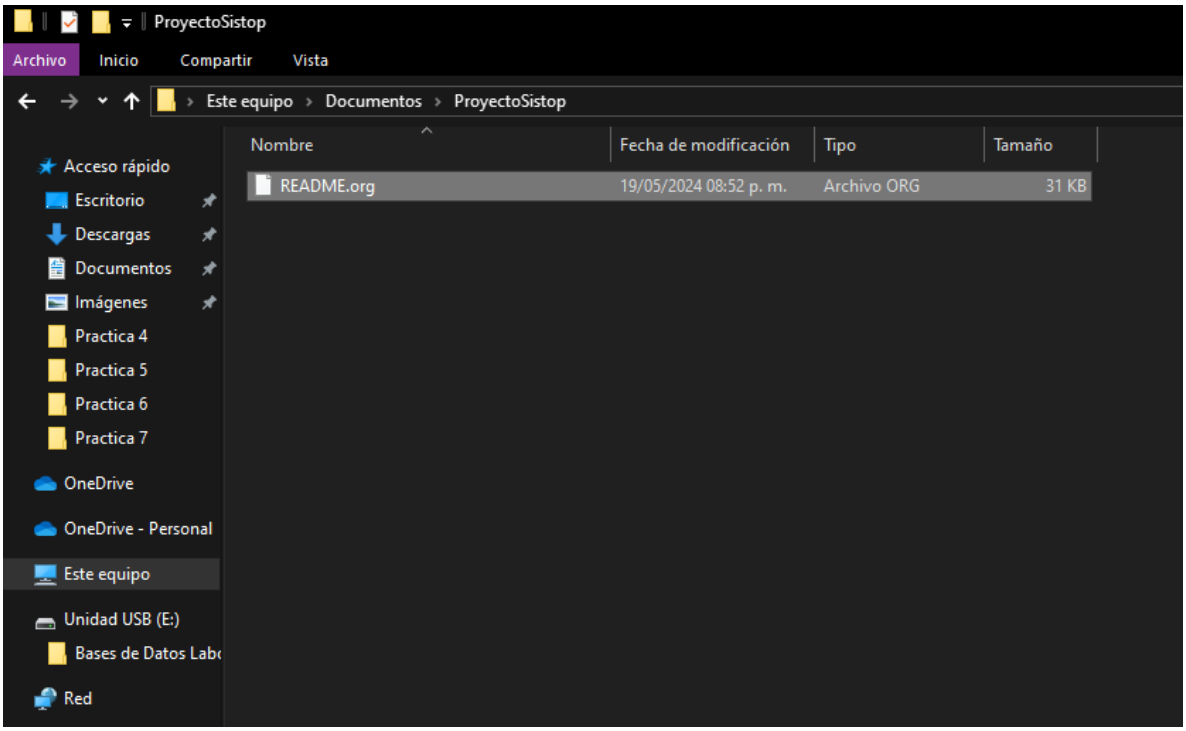
El programa muestra los archivos que se encuentran en el microsistema, en el programa tiene algunos detalles ya que el menú no se muestra de forma correcta pero el programa contiene la ejecución necesaria.

Cuando se elige la opción 2

```
Opciones disponibles:ARCHIVO: README.org     , TAMAÑO: 31222, CLUSTER INICIAL: 6, FECHA CREACION: 20240508131756,
FECHA MODIFICACION: 20240508131756


1. Listar los contenidos del directorioARCHIVO: logo.png       , TAMAÑO: 126423, CLUSTER INICIAL: 22, FECHA CREACI
ON: 20240508131756, FECHA MODIFICACION: 20240508131756


2. Copiar un archivo dentro del FiUnamFS hacia el sistemaARCHIVO: mensaje.jpg    , TAMAÑO: 254484, CLUSTER INICIAL
: 84, FECHA CREACION: 20240508131756, FECHA MODIFICACION: 20240508131756


3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa

Elegir opción (1,2,3,4 o 5): 2
Ingrese el nombre del archivo: README.org
Ingrese la ruta de destino en su sistema (incluyendo el nombre del archivo): C:\Users\ARMANDO\Documents\ProyectoSi
stop\README.org

Opciones disponibles:Copying README.org to C:\Users\ARMANDO\Documents\ProyectoSistop\README.org...

1. Listar los contenidos del directorio
Archivo 'README.org' copiado a 'C:\Users\ARMANDO\Documents\ProyectoSistop\README.org'.2. Copiar un archivo dentro
del FiUnamFS hacia el sistema

3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa
```

El programa te pide el nombre del archivo y la ruta donde quieres guardar el archivo y procede a almacenarlo en el equipo:

En la ruta de la computadora muestra el archivo seleccionado en este caso README.org



Si se elige la opción 3

```
Elegir opción (1,2,3,4 o 5): 3
Ingrese el nombre del archivo: temp.py
Ingrese la ruta de origen en su sistema (incluyendo el nombre del archivo): C:
\Users\suki\.spyder-py3\so\temp.py
Nuevo cluster libre: 7
Hay posición en 2112
Nuevo cluster libre: 87
Espacio libre en: 2112, Cluster libre para el archivo: 87

Opciones disponibles:
1. Listar los contenidos del directorio
2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa
```

```
Elegir opción (1,2,3,4 o 5): 1

Opciones disponibles:
1. Listar los contenidos del directorio
2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa
ARCHIVO: README.org    , TAMAÑO: 31222, CLUSTER INICIAL: 6, FECHA CREACION: 20240508131756,
FECHA MODIFICACION: 20240508131756

ARCHIVO: temp.py, TAMAÑO: 1059, CLUSTER INICIAL: 87, FECHA CREACION: 00000000000000, FECHA
MODIFICACION: 00000000000000

ARCHIVO: logo.png      , TAMAÑO: 126423, CLUSTER INICIAL: 22, FECHA CREACION:
20240508131756, FECHA MODIFICACION: 20240508131756

ARCHIVO: mensaje.jpg   , TAMAÑO: 254484, CLUSTER INICIAL: 84, FECHA CREACION:
20240508131756, FECHA MODIFICACION: 20240508131756


Elegir opción (1,2,3,4 o 5): |
```

El archivo nos pide el nombre del archivo y la ruta del equipo y observamos que el archivo se guarda en el micro sistema.

Si elegimos el inciso 4)

Usamos el ejemplo del inciso 1 donde selecionamos README.org y como observamos elimina el archivo

```
Opciones disponibles:
1. Listar los contenidos del directorio
2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa

Elegir opción (1,2,3,4 o 5): 4
Ingrese el nombre del archivo: README.org

Opciones disponibles:Archivo eliminado de FiUnamFS

1. Listar los contenidos del directorio
2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa

Elegir opción (1,2,3,4 o 5): 1

Opciones disponibles:ARCHIVO: logo.png       , TAMAÑO: 126423, CLUSTER INICIAL: 22, F
ECHA CREACION: 20240508131756, FECHA MODIFICACION: 20240508131756


1. Listar los contenidos del directorioARCHIVO: mensaje.jpg    , TAMAÑO: 254484, CLUS
TER INICIAL: 84, FECHA CREACION: 20240508131756, FECHA MODIFICACION: 20240508131756


2. Copiar un archivo dentro del FiUnamFS hacia el sistema
3. Copiar un archivo de la computadora hacia el FiUnamFS
4. Eliminar un archivo del FiUnamFS
5. Salir del programa
```

Y el inciso 5 finaliza el programa en ejecución