Universidad Nacional Autónoma de México

Facultad de Ingeniería

Alumno
Barriguete Rodríguez Héctor Alejandro

Sistemas Operativos

Proyecto #2: Sistema de archivos

Grupo 6

Fecha de entrega: 05 de enero de 2023

Objetivos

Desarrollar un sistema de archivos a partir del archivo *fiunamfs.img* que permita: mostrar todos los archivos, copiar archivos del sistema de archivos al nuestro y viceversa, eliminar archivos y desfragmentar el sistema de archivos.

Introducción

A partir de los conocimientos de los adquiridos en las clases de Sistemas Operativos se realizó un programa que permite realizar acciones (las mencionadas en los objetivos) sobre un sistema de archivos.

Antes de comenzar a explicar el código me gustaría mencionar que para desarrollar el programa me apoyé del código realizado por Lara Kevin y Loidi Javier, al final del documento se agregará la liga a su código.

La creación del programa se hizo utilizando Python (versión 3.10.0) y Windows 10. Solo se usa la biblioteca *os*.

Desarrollo

Primeramente, se imprime la información del archivo *fiunamfs.img*, entre dicha información que se imprime se encuentra el nombre del sistema de archivos, versión de la implementación, etiqueta del volumen, tamaño del cluster (en bytes), número de clusters que mide el directorio y número de clusters que mide la unidad completa.

El sistema de archivos debe de funcionar con una versión en específico, que es la *23.1*, si es una diferente el programa dejará de ejecutarse, para ello se agregó el archivo *versión.img*.

Si la versión coincide primero se hará una lectura de todo sistema de archivos para conocer que espacios están ocupados y cuales no, también se hace un reconocimiento de todos los clusters:

```python
#Función para obtener los indices de los clusters
def index_cluster(file_system, cluster_size):
    file_system.seek(cluster_size)

    for i in range(0, cluster_size * 4, 64):
        file_system.seek(cluster_size + i)
        lista_clusters.append(cluster_size + i)

#Función para obtener los nombres de los archivos y espacios vacios, se respeta el orden del sistema de archivos
def archivos_iniciales(file_system, cluster_size):
    lista_archivos.clear()

    file_system.seek(cluster_size)

    j = 0

    for i in range(0, cluster_size * 4, 64):
        file_system.seek(cluster_size + i)
        file = file_system.read(15).decode("utf-8")

        if file == "---------------":
            lista_archivos.append("---")
        else:
            while(file[j] == " " or file[j] == "-"):
                j += 1

            lista_archivos.append(file[j:15])
            j = 0
```

Posteriormente se muestra un menú con las acciones que puede realizar el usuario sobre el sistema de archivos:

Número	Acción
1	Mostrar contenido (archivos)
2	Copiar un archivo del sistema de archivos al equipo
3	Copiar un archivo del equipo al sistema de archivos
4	Eliminar un archivo
5	Desfragmentar
6	Salir (finalizar el programa)

1. *Mostrar contenido*

Se muestra el nombre de todos los archivos que se encuentren en el sistema de archivos, junto a su fecha de creación y última modificación. En caso de que no haya nombre de archivo se entiende que no existe un archivo en ese espacio:

```python
def mostrar_contenido(file_system, cluster_size):
    print("\n------------ CONTENIDO ------------")
    print("    Nombre\tCreacion del archivo\tUltima modificacion")
    file_system.seek(cluster_size)

    for i in range(0, cluster_size * 4, 64):
        file_system.seek(cluster_size + i)
        file = file_system.read(16).decode("utf-8")

        j = 0

        if file[j] ==  '-':
            while(file[j] == " " or file[j] == "-"):
                if j == 15:
                    break
                else:
                    j += 1

            if j < 15:
                print("-", file[j:15], end = "\t")

                j = 0
                j = 15 + 9
                file_system.seek(cluster_size + i + j)
                j = 0
```

```python
                #CREACION
                file = file_system.read(4).decode("utf-8") #Year
                print(file, end = "-")
                file = file_system.read(2).decode("utf-8") #Month
                print(file, end = "-")
                file = file_system.read(2).decode("utf-8") #Day
                print(file, end = " ")
                file = file_system.read(2).decode("utf-8") #Hour
                print(file, end = ":")
                file = file_system.read(2).decode("utf-8") #Minute
                print(file, end = ":")
                file = file_system.read(2).decode("utf-8") #Second
                print(file, end = "\t")

                #MODIFICACION
                file = file_system.read(4).decode("utf-8") #Year
                print(file, end = "-")
                file = file_system.read(2).decode("utf-8") #Month
                print(file, end = "-")
                file = file_system.read(2).decode("utf-8") #Day
                print(file, end = " ")
                file = file_system.read(2).decode("utf-8") #Hour
                print(file, end = ":")
                file = file_system.read(2).decode("utf-8") #Minute
                print(file, end = ":")
                file = file_system.read(2).decode("utf-8") #Second
                print(file, end = "\n")
```

2. *Copiar un archivo del sistema de archivos al equipo*

Función no implementada.

3. *Copiar un archivo del equipo al sistema de archivos*

Función no implementada.

4. *Eliminar un archivo*

El usuario ingresa el nombre del archivo que desea eliminar, posteriormente se busca el nombre en la lista en donde se encuentran todos los archivos registrados, si no se encuentra en la lista es porque no existe el archivo en el sistema de archivos.

En caso de existir se limpian los 64 bytes que corresponden a ese registro y se elimina de la lista de archivos que existen.

```
def eliminar_archivo(file_system, cluster_size):
    print("\n------------ ELIMINAR ARCHIVO DE fiunamfs ------------")

    file_delete = input("Ingresa el nombre del archivo a eliminar (incluya la extensión):

    if file_delete in lista_archivos:
        file_system.seek(cluster_size)

        for i in range(0, cluster_size * 4, 64):
            file_system.seek(cluster_size + i)
            file = file_system.read(16).decode("utf-8")

            j = 0

            if file[j] ==  '-':
                while(file[j] == " " or file[j] == "-"):
                    if j == 15:
                        break
                    else:
                        j += 1

            if j < 15:
                file = file[j:15]

                if file == file_delete:
                    #NOMBRE
                    file_system.seek(cluster_size + i)
                    file_system.write("---------------".encode("utf-8"))
                    #TAMAÑO
                    file_system.seek(cluster_size + i + 16)
                    file_system.write("0000".encode("utf-8"))
                    #CLUSTER INICIAL
                    file_system.seek(cluster_size + i + 19)
                    file_system.write("000".encode("utf-8"))
                    #FECHA DE CREACION Y MODIFICACION
                    file_system.seek(cluster_size + i + 24)
                    file_system.write("00000000000000000000000000000".encode("utf-8"))

                    lista_archivos.remove(file_delete)

                    print("\n¡¡¡¡¡ Archivo eliminado exitosamente !!!!!")
                    break

    else:
        print("\n!!!!! El archivo no existe en el sistema de archivos ¡¡¡¡¡")
```

5. *Desfragmentar*

Se utiliza un segundo programa, *df.py*, para implementar esta función, esto para que se pueda actualizar el sistema de archivos al momento de hacer los cambios.

Primeramente, recorremos todo el sistema de archivos para conocer todos los espacios de memoria (ocupados o no). Existe una lista (*lista_clusters*) que contiene cada espacio.

Posteriormente se hace otro recorrido para conocer todos los archivos y espacios vacíos que hay. Se cuenta con una lista (*lista_archivos*) que tiene el nombre o tres guiones (en caso de estar vacío).

Una vez que tenemos datos en las dos listas, primero recorremos *lista_archivos* para conocer los índices de todos los archivos, los índices se agregan a la lista *lista_index*. Una vez que conocemos todos los índices entramos a un ciclo para asegurar que todos los archivos se muevan (siempre y cuando haya un espacio disponible).

Lo primero que se hace es recorrer de nuevo *lista_archivos*, buscamos un espacio vacío (tres guiones), al encontrar uno guardamos el índice, después recorremos *lista_index* para saber si existe un archivo que se pueda mover a ese espacio.

Cuando si se pueda mover un archivo a un nuevo espacio, vaciamos *lista_index*, recorremos de nuevo *lista_archivos* para conocer todos los índices de los archivos. Después empezamos a modificar los 64 bits, primero los de la nueva posición y después los de la antigua, la información de esta última se cambia por lo que se encuentra en el archivo *registro_vacio.img*.

Actualizamos *lista_archivos*, para ellos debemos de limpiar la lista y recorrer el sistema de archivos. Cuando se actualice, revisamos si existe un espacio vacío entre dos archivos, de cumplirse la condición se repite todo lo anterior. Cuando ya no existan espacios entre archivos se considera como exitosa la desfragmentación.

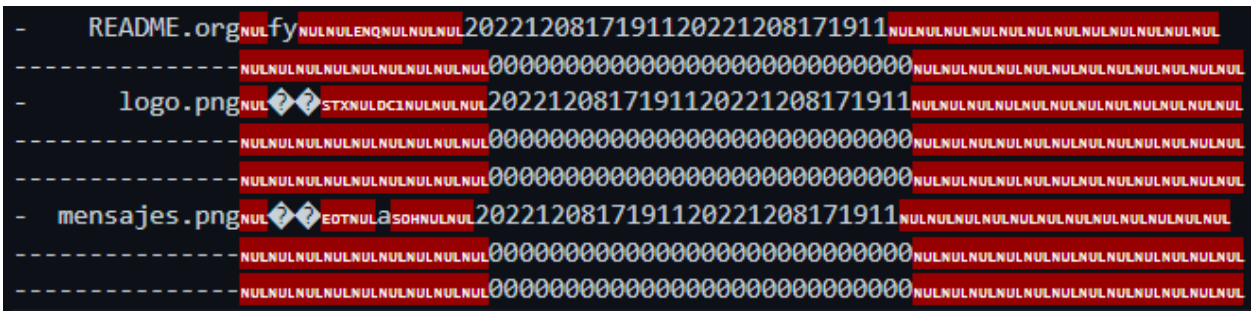
```python
global lista_clusters, lista_archivos
lista_archivos = []
lista_clusters = []

def desfragmentar(file_system, cluster_size):
    new_pos = 0
    old_pos = 0
    lista_index = []
    bandera = True
    banderaDos = True

    #Recorremos la lista de archivos para obtener los indices de los archivos
    for archivo in lista_archivos:
        if archivo != "---":
            lista_index.append(lista_archivos.index(archivo))

    while bandera == True:
        #Recorremos la lista de archivos para obtener el indice de de los espacios vacios
        new_pos = 0
        old_pos = 0

        for archivo in lista_archivos:
            if archivo == "---":
                new_pos = lista_archivos.index(archivo)

                #Recorremos la lista de indices para saber si se puede mover un archivo
                for i in lista_index:
                    #¿La nueva posición es menor a la actual del archivo?
                    if new_pos < i:
                        old_pos = i
                        break

            if new_pos != 0 and old_pos != 0:
                break

        #Actualización del file system
        if new_pos < old_pos:
            lista_index = []

            for archivo in lista_archivos:
                if archivo != "---":
                    lista_index.append(lista_archivos.index(archivo))
```

```python
            pos_one = lista_clusters[new_pos]
            pos_two = lista_clusters[old_pos]

            file_system.seek(pos_two)
            file = file_system.read(64)

            clean_registre = open("registro_vacio.img", "rb+")
            clean = clean_registre.read(64)
            file_system.seek(pos_two)
            file_system.write(clean)

            file_system.seek(pos_one)
            file_system.write(file)

            archivos_iniciales(file_system, 1024)

            for archivo in lista_archivos:
                if archivo != "---" and banderaDos == True:
                    pass
                elif archivo == "---" and banderaDos == True:
                    banderaDos = False

                if banderaDos == False and archivo == "---":
                    bandera = False
                    pass
                elif banderaDos == False and archivo != "---":
                    bandera = True
                    break

            if bandera == True:
                banderaDos = True

        print("\n******* Desfragmentación exitosa *******\n")
```

```python
def index_cluster(file_system, cluster_size):
    file_system.seek(cluster_size)

    for i in range(0, cluster_size * 4, 64):
        file_system.seek(cluster_size + i)
        lista_clusters.append(cluster_size + i)

def archivos_iniciales(file_system, cluster_size):
    lista_archivos.clear()

    file_system.seek(cluster_size)

    j = 0

    for i in range(0, cluster_size * 4, 64):
        file_system.seek(cluster_size + i)
        file = file_system.read(15).decode("utf-8")

        if file == "---------------":
            lista_archivos.append("---")
        else:
            while(file[j] == " " or file[j] == "-"):
                j += 1

            lista_archivos.append(file[j:15])
            j = 0

if __name__ == '__main__':
    print("\nDESFRAGMENTANDO...")
    file_system = open("fiunamfs.img", "r+b")

    sector_size = 256
    cluster_size = sector_size * 4

    index_cluster(file_system, cluster_size)
    archivos_iniciales(file_system, cluster_size)
    desfragmentar(file_system, cluster_size)

    file_system.close()
```

Ejecución del programa

fiunamfs.img original (al menos los primeros 9 registros):

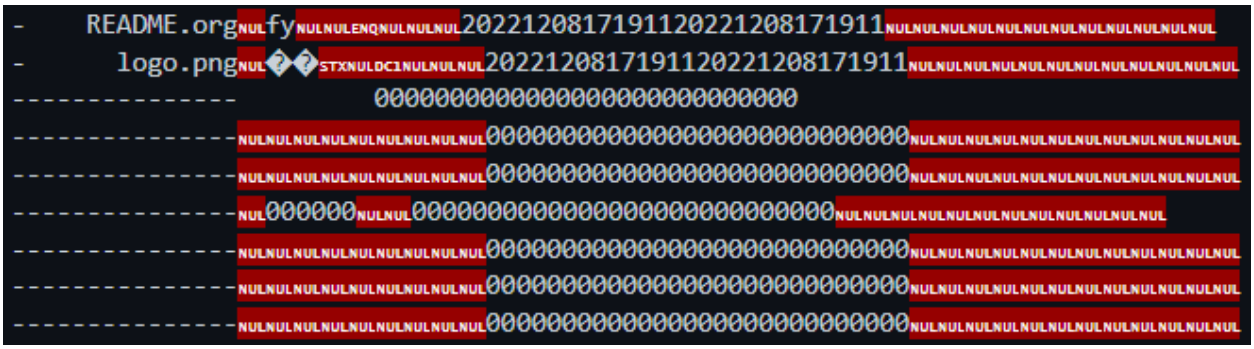


```
********** SISTEMA DE ARCHIVOS: FiUnamFs **********


Nombre del sistema de archivos: FiUnamFS
Versión de la implementación: 23.1
Etiqueta del volumen: FI-UNAM 2023-1
Tamaño del cluster (en bytes): ♦
Número de clusters que mide el directorio:


------------ MENU ------------
1. Listar contenido
2. Copiar archivo de FiUnamFS hacia este sistema
3. Copiar archivo de este sistema hacia FiUnamFS
4. Eliminar un archivo de FiUnamFS
5. Desfragmentar
6. Salir
Ingresa una opcion: 1

------------ CONTENIDO ------------
    Nombre      Creacion del archivo    Ultima modificacion
- README.org    2022-12-08 17:19:11     2022-12-08 17:19:11
- logo.png      2022-12-08 17:19:11     2022-12-08 17:19:11
- mensajes.png  2022-12-08 17:19:11     2022-12-08 17:19:11
```

```
------------ MENU ------------
1. Listar contenido
2. Copiar archivo de FiUnamFS hacia este sistema
3. Copiar archivo de este sistema hacia FiUnamFS
4. Eliminar un archivo de FiUnamFS
5. Desfragmentar
6. Salir
Ingresa una opcion: 4

------------ ELIMINAR ARCHIVO DE fiunamfs ------------
Ingresa el nombre del archivo a eliminar (incluya la extensión): mensajes.png

¡¡¡¡¡ Archivo eliminado exitosamente !!!!!
```

```
-    README.orgfy2022120817191120221208171911
---------------0000000000000000000000000000
-      logo.png2022120817191120221208171911
---------------0000000000000000000000000000
---------------0000000000000000000000000000
---------------000000000000000000000000000000
---------------0000000000000000000000000000
---------------0000000000000000000000000000
---------------0000000000000000000000000000
```

```
------------ MENU ------------
1. Listar contenido
2. Copiar archivo de FiUnamFS hacia este sistema
3. Copiar archivo de este sistema hacia FiUnamFS
4. Eliminar un archivo de FiUnamFS
5. Desfragmentar
6. Salir
Ingresa una opcion: 5

DESFRAGMENTANDO...

******** DesfragmentaciÃ³n exitosa ********
```



Conclusiones

Considero que fue un proyecto algo largo y complicado por todo lo que debe de realizarse, aunque antes había trabajado con archivos, nunca había hecho algo de esta forma.

A pesar de que no se cumplieron todas las funciones que se pidieron, considero que las realizadas funcionan adecuadamente.

Las funciones no realizadas no pudieron completarse debido a problemas que se tienen con el sistema de archivos, ya que no todas las posiciones indicadas en la explicación del proyecto contenían la información de forma adecuada. Aunque se presentó dicha situación al profesor y el mismo explicó lo que estaba ocurriendo, las modificaciones realizadas no fueron las necesarias como para que se pudiera manipular la información necesaria de manera adecuada.

Bibliografía

Código de apoyo: sistop-2020-2/proyectos/4/LoidiJavier-LaraKevin at master · gwolf/sistop-2020-2 (github.com)