



Universidad Nacional Autónoma de México
Facultad de ingeniería
Sistemas operativos

Proyecto 4: (Micro) Sistema de archivos

Profesor: Gunnar Eyal Wolf Iszaevich

Alumnos: Christian Alejandro Leyva Mercado y Velasco Pachuca Bryan

Grupo: 06

Introducción:

Para el cuarto proyecto se implementó un sistema de archivos, utilizando una especificación predefinida y un caso de referencia ya creado.

El programa deberá de ser capaz de obtener, crear y modificar la información en un micro-sistema de archivos *FiUnamFs*, como si se tratase de un sistema de archivos común.

En el programa se puede:

1. Listar los contenidos del directorio.
2. Copiar uno de los archivos del micro-sistema *FiUnamFs* hacia el sistema host que lo alberga (tu sistema).
3. Copiar un archivo del sistema host (tu computadora) hacia el micro-sistema *FiUnamFs*.
4. Eliminar un archivo del *FiUnamFs*.
5. Dada la simplicidad de la implementación, es de esperarse la fragmentación externa (ente clústers), por ello se deberá implementar la opción de revertir la fragmentación tras haber realizado operaciones de inserción y eliminación de archivos en el micro-sistema.

Descripción del entorno:

- **¿Qué lenguaje se emplea?**

Para el diseño del programa se empleó Python 3.9.10

- **¿Qué bibliotecas del lenguaje se emplean?**

Se emplearon las siguientes bibliotecas:

1. `os`: Funciones de interacción con el sistema operativo (`open()`, `write()`, etc.)
2. `sys`: Acceso a variables y funciones del intérprete de Python.
3. `mmap`: Brinda soporte para la lectura de archivos como un array de bytes.
4. `traceback`: Útil para imprimir el stack traceback del programa en un determinado momento.
5. `datetime`: Funciones de manipulación del tiempo.
6. `tabulate`: Impresión de información en forma de tablas sencillas.

****IMPORTANTE****

Python por default no cuenta con los módulos *tabulate* y *termcolor*, en caso de no contar con él, instalarlos a través de las siguientes instrucciones en terminal:

```
pip install tabulate
pip install termcolor
```

- **¿En qué sistema operativo se desarrolló?**

Terminal de Ubuntu en Windows 10 con Linux System for Windows y linea de comandos de Windows 10.

****Anotaciones****

Se implementaron todas las opciones de listar, importar, exportar y remover, pero la desfragmentación no se llegó a implementar.

Descripción del programa:

Para el funcionamiento del programa se emplearon las siguientes variables globales:

```
entVacia= '.....'
nSistemaArch = 'FiUnamFS'
versionSistArch = '1.1'
SuperBloque = {}
diskImg = None
DIMap = None
imgFilename = 'fiunamfs.img'
```

1. entVacia: Referencia de caracteres en caso de encontrarse una entrada vacía.
2. nSistemaArch: El nombre del sistema de archivos que implementa.
3. versionSistArch: El número de versión del sistema de archivos.
4. SuperBloque: Diccionario que contiene toda la información relacionada con el micro-sistema *FiUnamFs*.
5. diskImg: Se lee y guarda una copia de la imagen de disco proporcionada.
6. DIMap: Guarda la imagen del disco cómo arreglo de bytes para poder leerlo y modificarlo.
7. imgFilename: Contiene el nombre de la imagen con la especificación definida.

Las funciones utilizadas son las siguientes:

- `main()`:

```
def main():
    try:
        with open(imgFilename, 'r+b') as diskImg:
            global DIMap
            DIMap = mmap.mmap(diskImg.fileeno(), 0, access=mmap.ACCESS_COPY)
    except:
        cprint('Error: El archivo no se puede abrir. Verifica que su nombre sea ')
        return

    obtenerInfoSuperBloque()
    if SuperBloque['nombre'] == nSistemaArch:
        if SuperBloque['version'] == versionSistArch:
            sistemaArchivos()
        else:
            cprint('Error: La versión del sistema de archivos no es la 1.1', 'white')
    else:
        cprint('Error: El sistema de archivos no es: FiUnamFS', 'white', 'on_red')
```

Función inicial que primeramente valida la existencia de la imagen proporcionada por el profesor, la cual deberá poder ser leída y posteriormente cargada en *DIMap* como un array de bytes. Se utiliza una copia de la imagen para no alterar su estado inicial y, como característica futura, poder hacer un 'Rollback' de todos los cambios realizado en el micro-sistema.

La función *main()* también se encarga de obtener toda la información de la imagen proporcionada, la cual se encuentra en el primer cluster, para ello se emplea la función *obtenerInfoSuperBloque()*.

Una vez obtenida la información del súper bloque se valida que el nombre de su sistema de archivos sea FiUnamFS y la versión sea la 1.1, pues es sobre ese modelo sobre el que se estará operando.

Una vez validado el nombre y versión, se da paso a las funcionalidades del sistema de archivos, ejecutando la función *sistemaArchivos()*.

- obtenerInfoSuperBloque():

```
def obtenerInfoSuperBloque():  
    SuperBloque['nombre'] = DIMap[0:8].decode('utf-8')  
    SuperBloque['version'] = DIMap[10:13].decode('utf-8').strip()  
    SuperBloque['volumen'] = DIMap[20:35].decode('utf-8')  
    SuperBloque['tamaño'] = int(DIMap[40:45].decode('utf-8'))  
    SuperBloque['nClustersDir'] = int(DIMap[47:49].decode('utf-8'))  
    SuperBloque['nClustersCom'] = int(DIMap[52:60].decode('utf-8'))
```

Se encarga de obtener la información de la imagen proporcionada, como el nombre de su sistema de archivos, su versión, la etiqueta del nombre del volumen, el tamaño de cada clúster en bytes, el número de clústeres designados para el directorio y la cantidad total de clústeres que mide la unidad.

****Disclaimer****

El tamaño de clúster en bytes que se encuentra en la imagen es de 2048 bytes, a diferencia de la especificación teórica proporcionada por el profesor de 1024 bytes (cosa ya esclarecida en clase), lo que indicaría que cada clúster se compone de 4 sectores de 512 bytes y no de 256.

Sin embargo, no existe una gran alteración en la lógica de funcionamiento del sistema de archivos, salvo 2 situaciones:

1. La cantidad de entradas posibles de almacenar se duplica, pasando de 48 entradas a 96, no obstante, que se puedan referenciar la metainformación de los 96 archivos en la imagen de disco proporcionada dependerá del tamaño de cada archivo y el nivel de fragmentación externa presente en el disco.
2. Las fragmentación interna se vuelve el doble de notable (casi nada) pues con un clúster el doble de grande se puede llegar a utilizar menos espacio en el clúster final de cada archivo (algo realmente no preocupante).

- sistemaArchivos():

```
def sistemaArchivos():
    helpComandos = {'Comando':['ls [parametroOpcional]', 'export [noml
    salir = False
    while not salir:
        entry = input(">>> ")
        param = entry.split()
        if len(param) == 0:
            cprint('Por favor, ingresa un comando.\nEscribe \'help\'')
        elif param[0] == 'ls': ...
        elif param[0] == 'export': ...
        elif param[0] == 'import': ...
        elif param[0] == 'superinfo': ...
        elif param[0] == 'del': ...
        elif param[0] == 'help': ...
        elif param[0] == 'exit': ...
        else: ...
```

La función *sistemaArchivos()* es prácticamente la función medular del micro-sistema, pues le permite al usuario interactuar con él, para ello se simula un línea de comandos con distintas entradas que ejecutarán alguna operación sobre la imagen proporcionada, como podría ser listar su contenido, exportar un archivo del micro-sistema al sistema host, importar un archivo del sistema host al micro-sistema u obtener la información referente al súper bloque.

Al introducir un comando, se ejecutará la función correspondiente y se le mostrará al usuario el resultado.

```
>>> ls
```

n	Nombre del archivo	Tamaño del archivo	Cluster inicial	Fecha de creación	Fecha de modificación
0	README.org	29565	5	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
2	logo.png	52200	17	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
5	mensajes.png	158520	353	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022

- `ls()`:

```
def ls(param=None):
    archivos = []
    temp={}
    temp['cont'] = 'n'
    temp['nombre'] = 'Nombre del archivo'
    temp['tamaño'] = 'Tamaño del archivo'
    temp['clusterInicial'] = 'Cluster inicial'
    temp['fechaCreacion'] = 'Fecha de creación'
    temp['fechaModificacion'] = 'Fecha de modificación'
    archivos.append(temp)
    for i in range(0,64):
        desde = SuperBloque['tamaño'] + i * 64
        hasta = desde + 64
        if param is None:
            archivoLeido = getArchivos(DIMap[desde:hasta],i)
        elif param == '-comp':
            archivoLeido = getListadoComp(DIMap[desde:hasta],i)

        if not archivoLeido is None:
            archivos.append(archivoLeido)
    return archivos
```

Esta función permite listar el contenido encontrado en la imagen, a partir de ir a la dirección donde termina el súper bloque y comienza la información del directorio; para esto, se puede recorrerá todo el directorio buscando si se encuentra algún tipo de información diferente al vacío ('.....') y según el tipo de opción que se envíe se mostrarán o bien todas las entradas (`ls -com`) o solo aquellas con información referenciada a archivos (`ls`).


- `copy_export()`

```
def copy_export(filename:str,ruta:str):
    archivos = ls()
    archivo = FSgoogle(archivos,filename)
    if not archivo is None:
        if os.path.exists(ruta):
            with open(f"{ruta}/{filename}", "a+b") as export:
                desde = SuperBloque['tamaño'] * archivo['clusterInicial']
                hasta = desde + archivo['tamaño']
                export.write(DIMap[desde:hasta])
        else:
            cprint(f'Error: No se encontro la ruta: \"{ruta}\" en el siste
    else:
        cprint(f'Error: No se encontro el archivo \"{filename}\" en FiUnam
```

Permite copiar un archivo de la imagen con el micro-sistema FiUnamFS a algún directorio en el sistema host de la computadora, para esto ubica en que

clúster comienza su almacenamiento y exporta todo el contenido encontrado hasta el tamaño final que este tenga.

**Para exportar un archivo al mismo directorio donde se encuentra la imagen, únicamente colocar como ruta “./”.



```
export logo.png ./
```

- `copy_import()`:

```
def copy_import(filename):
    if os.path.isfile(filename):
        if len(filename) < 15:
            archivos = ls()
            if FSgoogle(archivos, filename) is None:
                fileSize = os.stat(filename).st_size
                nClusters = math.ceil(fileSize/SuperBloque['tamanio'])
                maxClusterFile = getMaxInitCluster(archivos)
                if not maxClusterFile is None:
                    if nClusters <= (SuperBloque['nClustersCom'] - maxClusterFile['clusterInicial'] + int(maxClusterFile['cont'])):
                        copyFileInto(maxClusterFile, filename, fileSize, nClusters, maxClusterFile['cont'])
                    else:
                        cprint(f'Error: No hay suficiente espacio para almacenar el archivo \'{filename}\'. Intenta con un archivo mas pequeño')
                else:
                    if nClusters <= (SuperBloque['nClustersCom'] - SuperBloque['nClustersDir']):
                        temp = {'clusterInicial': 5, 'tamanio': 0}
                        copyFileInto(temp, filename, fileSize, nClusters, 0)
                    else:
                        cprint(f'Error: El archivo \'{filename}\' es demasiado GRANDE. Intenta con un archivo mas pequeño')
```

La función `copy_import()` permite tomar un archivo del sistema host y moverlo al sistema FiUnamFS, para ello se debe considerar que el nombre del archivo no exceda los 15 caracteres y no se encuentre ya en el sistema de archivos, posteriormente calculará cuantos clústeres serán necesarios para almacenar toda la información requerida y procederá a su integración dentro del sistema de archivos utilizando la función `copyFileInto()`.

- `copyFileInto()`:

```
def copyFileInto(maxClusterFile,filename,fileSize,nClusters,contLast):
    initialCluster = maxClusterFile['clusterInicial'] + math.ceil(maxClusterFile['tamanio'] / SuperBloque['tamanio'] ) + 1
    finalCluster = initialCluster + nClusters
    global DIMap
    for i in range(contLast + 1,64):
        desde = SuperBloque['tamanio'] + i * 64
        hasta = desde + 64
        apartado = getListadoComp(DIMap[desde:hasta],i)
        if entVacia == apartado['nombre']:
            fechaActual = datetime.now()
            DIMap[desde + 0:desde + 15] = filename.rjust(15).encode('ASCII')
            DIMap[desde + 16:desde + 24] = str(fileSize).zfill(8).encode('ASCII')
            DIMap[desde + 25:desde + 30] = str(initialCluster).zfill(5).encode('ASCII')
            DIMap[desde + 31:desde + 45] = fechaActual.strftime("%Y%m%d%H%M%S").encode('ASCII')
            DIMap[desde + 46:desde + 60] = fechaActual.strftime("%Y%m%d%H%M%S").encode('ASCII')
            with open(filename,'rb') as file:
                desdeWrite = SuperBloque['tamanio'] * initialCluster
                hastaWrite = desdeWrite + fileSize
                DIMap[desdeWrite:hastaWrite] = file.read()
    return
```

`copyFileInto()` es la función encargada de primero registrar los metadatos del archivo en la sección correspondiente de la imagen (los clústeres designados del 1 al 4) y posteriormente escribir el archivo en los clústeres calculados dentro de la imagen.

- `rm()`:

```
def rm(filename:str):
    archivos = ls()
    archivo = FSgoogle(archivos,filename)
    if not archivo is None:
        desde = SuperBloque['tamanio'] + 64 * archivo['cont']
        DIMap[desde + 0:desde + 15] = bytes(entVacia, 'ASCII')
        DIMap[desde+16:desde+24] = bytes("".zfill(8), 'ASCII')
        DIMap[desde+25:desde+30] = bytes("".zfill(5), 'ASCII')
        DIMap[desde+31:desde+45] = bytes("".zfill(14), 'ASCII')
        DIMap[desde+46:desde+60] = bytes("".zfill(14), 'ASCII')
        desdeWrite = SuperBloque['tamanio'] * archivo['clusterInicial']
        hastaWrite = desdeWrite + archivo['tamanio']
        DIMap[desdeWrite:hastaWrite] = bytes("".zfill(hastaWrite-desdeWrite), 'ASCII')
    else:
        cprint(f'Error: No se encontro el archivo \'{filename}\'' en FiUnamFs.', 'white', 'on_red')
```

Finalmente, la función `rm()`, se encarga de remover toda la información relacionada a un archivo dentro de la imagen, como los metadatos o el contenido mismo, para ello sencillamente en lugar de leerlos los sustituye por ceros

Capturas del funcionamiento:

- **Listar el contenido del directorio**

Solo las entradas con contenido:

```
>>> ls
```

n	Nombre del archivo	Tamaño del archivo	Cluster inicial	Fecha de creación	Fecha de modificación
0	README.org	29565	5	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
2	logo.png	52200	17	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
5	mensajes.png	158520	353	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022

Todo el contenido:

```
>>> ls -comp
```

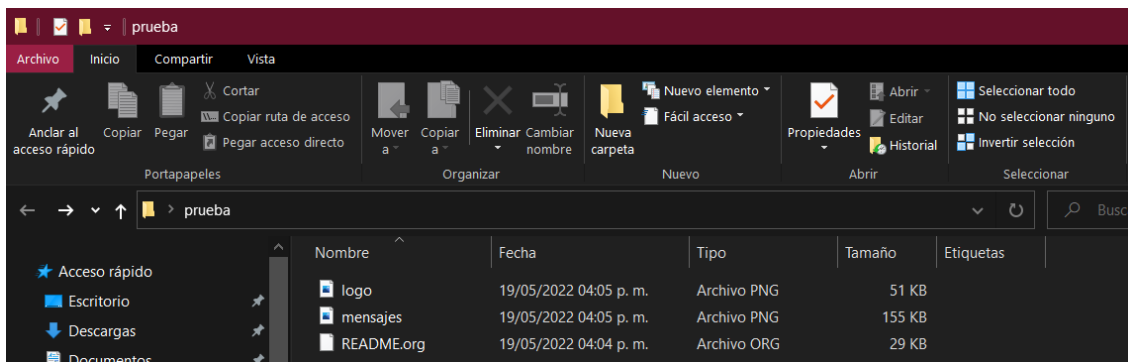
n	Nombre del archivo	Tamaño del archivo	Cluster inicial	Fecha de creación	Fecha de modificación
0	README.org	29565	5	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
1	0	0	-	-
2	logo.png	52200	17	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
3	0	0	-	-
4	0	0	-	-
5	mensajes.png	158520	353	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
6	0	0	-	-
7	0	0	-	-
8	0	0	-	-
9	0	0	-	-
10	0	0	-	-
11	0	0	-	-

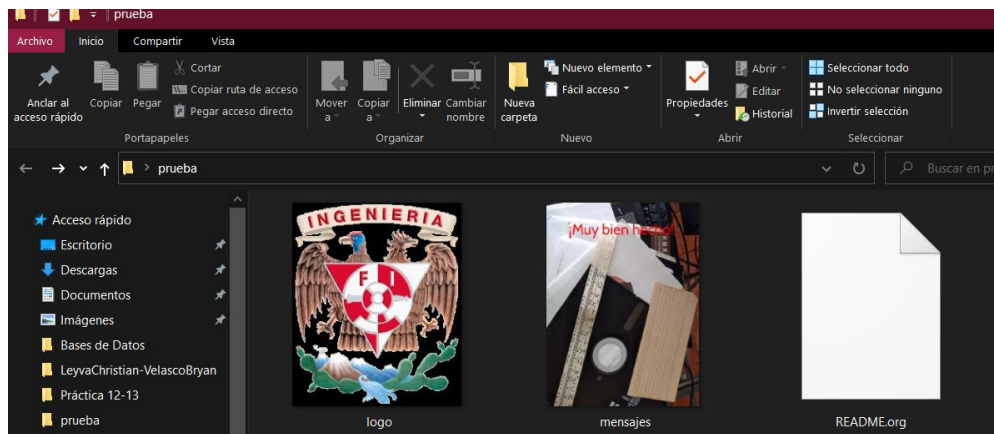
- **Exportar el contenido**

Comandos ingresados:

```
>>> export README.org C:/Users/DELL/Desktop/prueba
>>> export mensajes.png C:/Users/DELL/Desktop/prueba
>>> export logo.png C:/Users/DELL/Desktop/prueba
```

Archivos en directorio local:





- **Importar el contenido**

Desde directorio raíz:

```
>>> import C:/prueba.txt
>>> ls
```

n	Nombre del archivo	Tamaño del archivo	Cluster inicial	Fecha de creación	Fecha de modificación
0	README.org	29565	5	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
2	logo.png	52200	17	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
5	mensajes.png	158520	353	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
6	C:/prueba.txt	12	432	Thu May 19 23:15:44 2022	Thu May 19 23:15:44 2022

```
>>> exit
```

Al mismo nivel que el programa

```
>>> import ./prueba2.txt
>>> ls
```

n	Nombre del archivo	Tamaño del archivo	Cluster inicial	Fecha de creación	Fecha de modificación
0	README.org	29565	5	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
2	logo.png	52200	17	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
5	mensajes.png	158520	353	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
6	C:/prueba.txt	12	432	Thu May 19 23:15:44 2022	Thu May 19 23:15:44 2022
7	./prueba2.txt	12	434	Thu May 19 23:26:09 2022	Thu May 19 23:26:09 2022

****Debido al tiempo se dejó el nombre de la ruta como el completo por lo que para que no exceda los 15 caracteres, se recomienda dejar el archivo en el directorio más cercano a la raíz o bien en el mismo directorio donde se encuentra el programa .py**

- **Eliminar un archivo dentro del sistema FiUnamFS**

```
>>> rm C:/prueba.txt
>>> ls
```

n	Nombre del archivo	Tamaño del archivo	Cluster inicial	Fecha de creación	Fecha de modificación
0	README.org	29565	5	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
2	logo.png	52200	17	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
5	mensajes.png	158520	353	Sun May 8 13:57:06 2022	Sun May 8 13:57:06 2022
7	./prueba2.txt	12	434	Thu May 19 23:26:09 2022	Thu May 19 23:26:09 2022