Universidad Nacional Autónoma de México

Facultad de ingeniería

Sistemas Operativos

Proyecto 3

Sistema de archivos FiUnamFs

Integrantes del equipo:

López Sugahara Ernesto Danjiro

Fecha de entrega: 23/11/2023

Planteamiento del problema

Se tiene un archivo que simula el almacenamiento de un diskette con un sistema de archivos denominado FiUnamFs. Para poder acceder a los archivos e igualmente poder utilizar el sistema de archivos para copiar información o eliminar información, es necesario comprender el formato del sistema de archivos e implementar un programa que permita manipularlo.

Resolución y explicación del código

Lo primero fue identificar las formas en las que se obtendría la información. A grandes rasgos se observó que habría que obtener la información de 3 formas:

a. Cadenas de texto en formato ASCII – 8, lo cual se puede decodificar mediante LATIN – 1 y la función decode de Python.

b. Valores enteros codificados en hexadecimales de 4 bytes, para lo cual se utiliza la función unpack de la biblioteca Struct de Python.

c. Información en bruto dada en bytes para los archivos que se encuentran dentro del sistema.

Para los puntos anteriores se generaron funciones que permiten realizar dichas operaciones evitando reescribir constantemente las mismas instrucciones:

```python
# Para la información de números enteros, se deberá de utilizar unpack
def leerEnteros(cabezal,tam):
    global ruta_imagen
    # Modo de apertura 'r' para lectura
    with open(ruta_imagen,'rb') as FiUnamFS:
        # Se tiene que ubicar el cabezal
        FiUnamFS.seek(cabezal)
        # Se lee la información deseada
        contenido = FiUnamFS.read(tam)
        contenido, *resto = struct.unpack('<I',contenido)
        return contenido

# Las cadenas de texto se procesaran como ASCII-8
def leerAscii(cabezal,tam):
    global ruta_imagen
    with open(ruta_imagen,'rb') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        # Leemos la información y la decodificamos en Latin-1 -> ASCII 8 bits
        contenido = FiUnamFs.read(tam).decode('Latin-1')
        return contenido

# Para pruebas, se tiene la lectura de información de bruto
def leerInfo(cabezal,tam):
    global ruta_imagen
    with open(ruta_imagen,'rb') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        # Leemos la información de bruto
        contenido = FiUnamFs.read(tam)
        return contenido
```

Nota: La variable ruta_imagen contiene la dirección del archivo que simula el almacenamiento. En este caso se almacena en el mismo directorio que el código, por lo que la ruta simplemente se compone por el nombre del archivo.

Con base en la información dada en el planteamiento del problema, lo primero que se realizó fue la obtención de la información del volumen almacenada en el superbloque. Esto se observa a continuación:

```
# Se recupera la información del superbloque. Esta deberá de ser de acceso global para todos
identificador = leerAscii(0,8)
version = leerAscii(10,4)
etiquetaVolumen = leerAscii(20,19)
tamCluster = leerEnteros(40,4)
numClusterDir = leerEnteros(45,4)
clusterTotales = leerEnteros(50,4)
```

Los valores se describen a continuación:

1. Identificador contiene el nombre del sistema de archivos: FiUnamFs.

2. La versión contiene el número de versión del sistema: 24.1.

3. La etiqueta contiene información adicional: No quiero etiqueta.

4. El tamaño del clúster brinda el tamaño de todos los clústeres del sistema, lo que permite posicionar el cabezal de forma adecuada para lectura y escritura de información. En este caso el tamaño resulto de 2048.

Una vez obtenida la información general del sistema, se procedió a analizar la información del directorio raíz. Para esto se observó en el planteamiento que era posible determinar la información siguiente

```
numEntradas = 128
entradasDirectorio = 64
tamSectores = 256
inicioDir = tamCluster
finDir = inicioDir + 4 * tamCluster
```

La anterior se obtuvo de la siguiente forma:

1. El número de entradas es la cantidad de archivos que puede haber en el sistema. En este caso, como se tiene que el directorio toma 4 clusters y cada entrada tiene una capacidad de 64 bytes, se observa la siguiente operación:

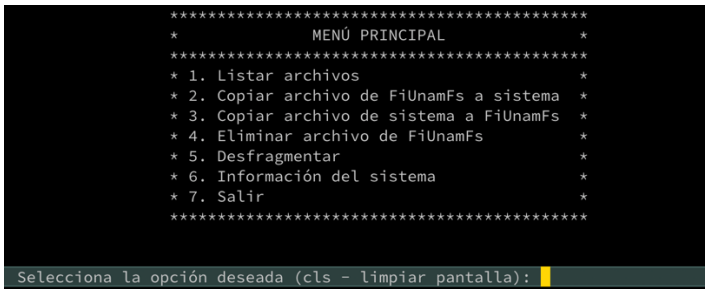
$$num_{entradas} = \frac{4 * 2048}{64} = 128$$

2. La segunda variable es para almacenar el tamaño de las entradas.

3. El tamaño de sectores se brinda en el planteamiento del problema.

4. Para el inicio y fin del directorio se obtienen a partir del planteamiento, donde se define el intervalo del primero al cuarto clúster. Para verificar lo anterior se utilizó la función de leer información y los intervalos de interés para mostrar el contenido del archivo, observando que precisamente en ese intervalo se podían observar todas las 128 entradas del directorio (algunas con información, otras vacías)

A grandes rasgos, el sistema puede realizar las siguientes tareas: mostrar la información; copiar archivos del sistema FiUnamFs al sistema personal; copiar archivos del sistema a FiUnamFs; eliminar archivos de FiUnamFs y desfragmentar. Para esto se implementó un menú y un ciclo donde se solicitan las acciones que desea realizar al usuario. Lo anterior se observa a continuación:



Nota: Antes de mencionar cómo se realizó la solución de todas las funciones, es bueno observar dos de los elementos clave de mi solución al problema. Como la base es la información del directorio, esta información se almacena en dos variables de interés:

```
archivos = {}
entradasLibres = []
```

La variable archivos contiene toda la información relevante a los archivos que brinda el directorio. Estos se encuentran almacenados de la forma: {'nombre': {datos reales}}. Igualmente se adicionó un dato extra a los archivos para determinar el clúster específico del archivo en el directorio.

La variable de entradas libres permite determinar los clústeres del directorio donde es posible ingresar la información de un nuevo archivo. Esto facilita mucho la tarea de copiar un archivo del sistema a FiUnamFs.

Para obtener toda la información anterior, se generó una función inicial que lee la información relevante de FiUnamFs:

```python
def guardarInformacionArchivos():
    # Se mostrarán únicamente los archivos que tienen un nombre específico
    # Se deberá de recorrer el directorio
    cabezal = inicioDir
    global numEntradas
    global archivos
    archivos.clear()
    numEntradas = 128
    while(cabezal != finDir):
        archivo = {}
        with open(ruta_imagen,'rb') as FiUnamFs:
            FiUnamFs.seek(cabezal)
            # Se revisa si es una entrada con contenido o vacía
            entrada = leerAscii(cabezal,1)
            if entrada == '-':
                # Se lee el resto
                archivo['nombre'] = leerAscii(cabezal + 1, 14) # Se lee el nombre real
                archivo['tam'] = leerEnteros(cabezal + 16, 4) # Se lee el tamaño
                archivo['clusterInicial'] = leerEnteros(cabezal + 20, 4) # Se lee el tamaño del cluster
                fecha_objeto = datetime.strptime(leerAscii(cabezal + 24, 13), "%Y%m%d%H%M%S")
                cadena_formateada = fecha_objeto.strftime("%Y-%m-%d %H:%M:%S")
                archivo['fechaC'] = cadena_formateada # Se lee hora y fecha de creación del archivo
                fecha_objeto = datetime.strptime(leerAscii(cabezal + 38, 13), "%Y%m%d%H%M%S")
                cadena_formateada = fecha_objeto.strftime("%Y-%m-%d %H:%M:%S")
                archivo['fechaM'] = cadena_formateada # Se lee hora y fecha de creación del archivo
                archivo['clusterDirectorio'] = cabezal
                # Se guardar la información de los archivos que tienen información
                archivos[archivo['nombre'].rstrip()] = archivo
                cabezal += 64
                numEntradas -= 1
                pass
            else:
                # Se ignora y se avanza el cabezal
                entradasLibres.append(cabezal)
                cabezal += 64
```

Igualmente, la función se utiliza cada vez que se modifica la información del sistema.

Funciones

Con lo anterior planteado, se muestra el desarrollo de las funcionalidades especificadas:

Listar archivos

En este caso, únicamente se imprime con formato la información ya recabada al inicio del programa. El código y el resultado es el siguiente:

```python
# Listar los elementos del directorio
def listarContenidos():
    # Se mostrarán únicamente los archivos que tienen un nombre específico
    print(f"\n\n\tARCHIVOS: \n\n\t{'-'* (5+14+10+10+19+10)}")
    for i,archivo in enumerate(archivos.items()):
        print(f"{i:>5}.- {archivo[0]:<14}: {archivo[1]['tam']:<10} -- {archivo[1]['fechaC']} -- {archivo[1]['fechaM']}")
    print(f"\n\t{'-'* (5+14+10+10+19+10)}")
```

```
      Selecciona la opción deseada (cls - limpiar pantalla): 1


      ARCHIVOS:

      ------------------------------------------------------------------
  0.- README.org     : 31209      -- 2023-11-16 13:03:03 -- 2023-11-16 13:03:03
  1.- logo.png       : 170192     -- 2023-11-16 13:03:03 -- 2023-11-16 13:03:03
  2.- mensaje.jpg    : 102657     -- 2023-11-16 13:03:03 -- 2023-11-16 13:03:03

      ------------------------------------------------------------------
```

Copiar archivo de FiUnamFs a sistema

Para esta funcionalidad, se requieren de dos elementos: la ruta dentro del sistema donde se copiará el archivo y el nombre del archivo dentro del sistema FiUnamFs. Con lo anterior:

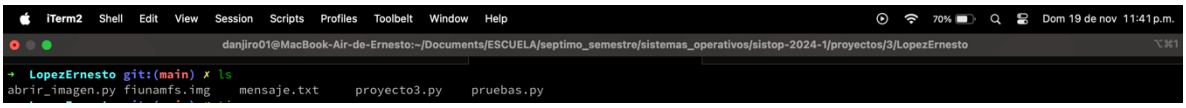
1. Lo primero que se realiza es validar tanto el nombre como la ruta.

2. En caso de que sea válida la información, se copia el contenido del archivo mediante la función de lectura general definida al principio del documento, colocando el cabezal en el clúster inicial (información que se encuentra en el diccionario de archivos que se mencionó que sería de suma importancia) y mandando el tamaño del archivo. Posteriormente se crea un archivo en la ruta especificada y se escribe el contenido del archivo del sistema FiUnamFs.

La función se observa a continuación:

```
def copiarArchivoASistema():
    print("\tIngrese la ruta donde desea copiar el archivo del sistema FiUnamFs:")
    ruta = input(" ").rstrip().lstrip()
    print("\tIngrese el nombre del archivo a copiar:")
    nombre = input(" ").rstrip().lstrip()
    # Primero se debe validar que el nombre del archivo exista
    if nombre in archivos:
        # Se encntró el archivo, por lo tanto se procede a copiar (en caso de que la ruta sea la correcta y no exista el archivo)
        if (os.path.exists(ruta) and ((os.path.exists(ruta + "/" + nombre) == False))):          VALIDACIONES
            # Se copia el archivo, para lo cual se necesitan los datos
            contenido = leerInfo(archivos[nombre]['clusterInicial'] * tamCluster,archivos[nombre]['tam'])   COPIA DEL CONTENIDO
            with open(ruta + "/" + nombre,"wb") as archivo:
                archivo.write(contenido)                                                  COPIA AL SISTEMA
            print("\n\tARCHIVO GUARDADO EXITOSAMENTE")
        else:
            print("ERROR: La ruta especificada no existe o el archivo ya se encuentra en dicha ruta.")
    else:
        print("\tERROR: No existe un archivo con ese nombre")
```

Para mostrar su funcionamiento, se observa la hora y el directorio donde se realizará la copia:



Antes de realizar la copia

```
        Selecciona la opción deseada (cls - limpiar pantalla): 2
        Ingrese la ruta donde desea copiar el archivo del sistema FiUnamFs:
 /Users/danjiro01/Documents/ESCUELA/septimo_semestre/sistemas_operativos/sistop-2024-1/proyectos/3/LopezErnesto
        Ingrese el nombre del archivo a copiar:
 mensaje.jpg

        ARCHIVO GUARDADO EXITOSAMENTE
```

Realizando la copia

Después de realizar la copia

Con lo anterior, ahora es posible visualizar el archivo. Como el archivo es tipo binario y es una imagen, se realizó un código básico en Python para abrir la imagen, con lo que se visualizó la siguiente imagen maravillosa:

Copiar archivo del sistema al sistema FiUnamFs

Esta función fue la más complicada de implementar. Para su implementación se consideraron los siguientes elementos:

1. El directorio tiene entradas libres.

2. La ruta del archivo es válida.

3. El tamaño del archivo no sobrepasa el tamaño disponible para almacenamiento del sistema. Este tamaño se basa en que hay 715 clústeres disponibles de tamaño 2048 Bytes. El número de clústeres se obtuvo bajo el esquema de que, el superbloque mide 1 y el directorio 4, y el sistema en total tiene 720.

4. El nombre del archivo es menor o igual a 14 caracteres (tamaño definido del sistema).

5. Bajo esas condiciones es posible realizar una búsqueda de espacio para el archivo. Para esto se realizó una función que busca un posible clúster de asignación al archivo (en caso de que no haya lugar disponible, se manda falso y no se realiza la copia). La función se observa a continuación:

```
def asignarEspacio(tam):
    clusterNecesarios = ceil(tam/tamCluster)
    almacenamiento = []
    cluster = 5 # Se empieza por el cluster posterior al directorio
    for archivo in archivos.items():
        almacenamiento.append((archivo[1]['clusterInicial'],archivo[1]['clusterInicial'] + ceil(archivo[1]['tam'] / tamCluster)))
    almacenamiento.sort()
    while(cluster < 720):
        if len(almacenamiento) != 0 and cluster == almacenamiento[0][0]: # Si el cabezal se encuentra en un cluster ocupado, se lo salta
            cluster = almacenamiento[0][1] + 1
            almacenamiento.pop(0) # Se quita el cluster
        else:
            if len(almacenamiento) != 0 and (cluster + clusterNecesarios > almacenamiento[0][0]): # Choca con otro archivo
                cluster = almacenamiento[0][1] + 1
                almacenamiento.pop(0)
            else: # Es posible almacenar el archivo
                return cluster * tamCluster

        if len(almacenamiento) == 0 and (cluster + clusterNecesarios) < 720: # Es posible almacenar el archivo posterior a todos los demás
            return cluster * tamCluster

    return False
```

Se obtiene la información de almacenamiento de los archivos actuales dentro del sistema. Para esto se almacena el clúster inicial y el final, ordenando de forma ascendente

Si el clúster se encuentra en la misma posición que el primer archivo, se deberá de saltar al final de esta posición

Si se encuentra un clúster, pero con los clústers necesarios sobrepasa la posición inicial de otro archivo, habrá que saltar hasta el final de dicho archivo.

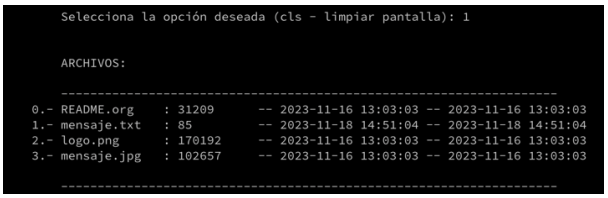
En caso de que ya no haya más archivos, se deberá de observar si el archivo entra dentro de los límites del tamaño del sistema.

Nota: Posterior al último if, se agregó un else que retorna False, ya que, en caso de que no cumpla dicha condición no tiene caso seguir buscando.

Con lo anterior, se puede visualizar el funcionamiento de la forma siguiente:

La imagen muestra el estado actual del directorio, donde se tiene la información de los tres archivos. Posteriormente, dentro del sistema creado se realiza la operación de copiado:

Posterior a realizar la copia, se observar que se pueden listar los elementos del sistema, conteniendo ahora 4 archivos:



Igualmente, recorriendo el sistema se observa que la información se encuentra almacenada de forma correcta:

Nota: Para escribir la información del directorio y del archivo al sistema, se implementaron las funciones siguientes:

```python
def escribirDirectorio(nombre,tam,cabezal,fechaModificacion,fechaCreacion):
    global ruta_imagen
    global numEntradas
    numEntradas -= 1
    nombre = nombre.ljust(14)
    directorio = entradasLibres.pop(0)
    escribirAscii(directorio,'-')
    escribirAscii(directorio + 1,nombre)
    escribirEnteros(directorio + 16, tam)
    escribirEnteros(directorio + 20, ceil(cabezal/tamCluster))
    escribirAscii(directorio + 24,fechaCreacion)
    escribirAscii(directorio + 38,fechaModificacion)

def escribirInfo(cabezal,contenido):
    global ruta_imagen
    global numEntradas
    with open(ruta_imagen,'rb+') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        FiUnamFs.write(contenido)
    guardarInformacionArchivos()
```

Igualmente, las funciones para escribir ASCII y enteros de forma respectiva:

```
def escribirDirectorio(nombre,tam,cabezal,fechaModificacion,fechaCreacion):
    global ruta_imagen
    global numEntradas
    numEntradas -= 1
    nombre = nombre.ljust(14)
    directorio = entradasLibres.pop(0)
    escribirAscii(directorio,'-')
    escribirAscii(directorio + 1,nombre)
    escribirEnteros(directorio + 16, tam)
    escribirEnteros(directorio + 20, ceil(cabezal/tamCluster))
    escribirAscii(directorio + 24,fechaCreacion)
    escribirAscii(directorio + 38,fechaModificacion)

def escribirInfo(cabezal,contenido):
    global ruta_imagen
    global numEntradas
    with open(ruta_imagen,'rb+') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        FiUnamFs.write(contenido)
    guardarInformacionArchivos()
```

```
def escribirAscii(cabezal,contenido):
    global ruta_imagen
    with open(ruta_imagen,'rb+') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        FiUnamFs.write(contenido.encode('Latin-1'))

def escribirEnteros(cabezal,contenido):
    global ruta_imagen
    with open(ruta_imagen,'rb+') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        FiUnamFs.write(struct.pack('<I',contenido))
```

Nota: Cuando se realiza una sentencia del tipo ceil(valor/tamCluster) es debido a que se desea almacenar u obtener el valor de un clúster y el dato valor está dado en bytes, por lo mismo, al realizar la división se busca obtener un clúster específico. Como la división puede resultar en un valor decimal, el hecho de que tome un valor decimal significa que está siendo ocupado todo el clúster, por lo que se debe de redondear hacia arriba.

Eliminar archivo

Para eliminar un archivo, lo primero que se realiza es la validación de la existencia de dicho archivo. Si el archivo es válido, se realiza lo siguiente:

1. Se obtiene toda la información asociada a dicho archivo.
2. Se elimina el contenido y el directorio.
3. Se actualiza el diccionario de archivos.

Esto se observa en el código siguiente:

```
def eliminarArchivoFiUnamFs():
    print("Ingrese el nombre del archivo que desea eliminar:")
    nombre = input(" ").rstrip().lstrip()
    # Primero se debe validar que el nombre del archivo exista
    if nombre in archivos:
        # Se elimina el archivo
        informacion = archivos[nombre]
        eliminarDirectorio(informacion['clusterDirectorio'])
        eliminarInfo(informacion['clusterInicial'] * tamCluster,informacion['tam'])
    else:
        print("\tERROR: No existe un archivo con ese nombre")
    print("\tArchivo eliminado exitosamente")
    guardarInformacionArchivos()
```

Las funciones respectivas para eliminar la información son las siguientes:

```
def eliminarDirectorio(cabezal):
    global numEntradas
    global entradasLibres
    entradasLibres.append(cabezal)
    entradasLibres.sort()
    numEntradas += 1
    escribirAscii(cabezal,'/..............')
    escribirAscii(cabezal + 24,'00000000000000000000000000000000')
    with open(ruta_imagen,'rb+') as FiUnamFs:
        FiUnamFs.seek(cabezal + 16)
        FiUnamFs.write(b'\x00' * 9)
        FiUnamFs.seek(cabezal + 52)
        FiUnamFs.write(b'\x00' * 12)

def eliminarInfo(cabezal,tam):
    with open(ruta_imagen,'rb+') as FiUnamFs:
        FiUnamFs.seek(cabezal)
        FiUnamFs.write(b'\x00' * tam)
```

Se actualiza la información relativa a los directorios

Se limpia la información dentro del directorio

Con lo anterior, se puede observar el funcionamiento de la eliminación:

```
        Selecciona la opción deseada (cls - limpiar pantalla): 4
Ingrese el nombre del archivo que desea eliminar:
 logo.png
        Archivo eliminado exitosamente
```

```
 0.- README.org     : 31209       -- 2023-11-16 13:03:03 -- 2023-11-16 13:03:03
 1.- mensaje.txt    : 85          -- 2023-11-18 14:51:04 -- 2023-11-18 14:51:04
 2.- mensaje.jpg    : 102657      -- 2023-11-16 13:03:03 -- 2023-11-16 13:03:03
```

Desfragmentar

Para la desfragmentación se consideró nuevamente utilizar el diccionario de archivos para obtener los tamaños y clústeres de cada uno de los archivos. Con esto, se ordenó con base en el clúster inicial de cada archivo para reordenar la información de tal forma de que no hubiera sobreescritura y se aprovechara mejor el espacio de almacenamiento. La función es la siguiente:

```python
def desfragmentar():
    # Se tiene mucha fragmentación externa en los archivos, por lo que lo mejor será desfragmentar el almacenamiento
    # Vamos a ordener los archivos por su cluster inicial, comenzando por el primer cluster posterior al directorio
    """
        Como se irán moviendo los archivos, habrá que modificar su respectivo directorio cambiando el cluster inicial
        Primero se necesita la información de todos los archivos de forma ordenada

        OJO: El método de desfragmentación que estoy realizando deja basura, sin embargo, se sobreescribirá sobre dicha basura
    """
    print(archivos)
    informacionArchivos = []
    for archivo in archivos.items():
        informacionArchivos.append([archivo[1]['clusterInicial'],archivo[1]])
    informacionArchivos.sort()
    cluster = 5 # Primer cluster posterior al directorio
    for archivo in informacionArchivos:
        # Se mueve toda la información del archivo a la primera posición
        if cluster != archivo[1]['clusterInicial']:
            rescribirDirectorio(archivo[1]['clusterDirectorio'],cluster)
            with open(ruta_imagen,'rb') as FiUnamFs:
                FiUnamFs.seek(archivo[1]['clusterInicial'] * tamCluster)
                contenido = FiUnamFs.read(archivo[1]['tam'])
            escribirInfo(cluster * tamCluster,contenido)
            cluster += ceil(archivo[1]['tam'] / tamCluster)
            print(cluster)
        else:
            cluster += ceil(archivo[1]['tam'] / tamCluster)
    print(archivos)
```

Se obtiene la información de los archivos y su respectivo clúster inicial. Esta lista se ordena para ir posicionando con respecto al clúster más pequeño

Si el clúster es diferente del clúster inicial, este será el nuevo clúster inicial del archivo y se deberá de actualizar el directorio

Se obtiene la información del archivo y se escribe en la nueva posición

El funcionamiento se muestra a continuación:

```
        Selecciona la opción deseada (cls - limpiar pantalla): 5
{'README.org': {'nombre': 'README.org     ', 'tam': 31209, 'clusterInicial': 6, 'fechaC': '2023-11-16 13:03:03', 'fechaM': '2023-11-16 13:03:03', 'clusterDirectori
o': 2048}, 'mensaje.txt': {'nombre': 'mensaje.txt    ', 'tam': 85, 'clusterInicial': 5, 'fechaC': '2023-11-18 14:51:04', 'fechaM': '2023-11-18 14:51:04', 'clusterD
irectorio': 2112}, 'mensaje.jpg': {'nombre': 'mensaje.jpg    ', 'tam': 102657, 'clusterInicial': 154, 'fechaC': '2023-11-16 13:03:03', 'fechaM': '2023-11-16 13:03:
03', 'clusterDirectorio': 2368}}
6
22
73
{'README.org': {'nombre': 'README.org     ', 'tam': 31209, 'clusterInicial': 6, 'fechaC': '2023-11-16 13:03:03', 'fechaM': '2023-11-16 13:03:03', 'clusterDirectori
o': 2048}, 'mensaje.txt': {'nombre': 'mensaje.txt    ', 'tam': 85, 'clusterInicial': 5, 'fechaC': '2023-11-18 14:51:04', 'fechaM': '2023-11-18 14:51:04', 'clusterD
irectorio': 2112}, 'mensaje.jpg': {'nombre': 'mensaje.jpg    ', 'tam': 102657, 'clusterInicial': 22, 'fechaC': '2023-11-16 13:03:03', 'fechaM': '2023-11-16 13:03:0
3', 'clusterDirectorio': 2368}}
```

Para validar que la información sigue estando correcta, se imprimió parte de la información:

```python
print(leerInfo(22 * 2048,1000))
```

Obteniendo lo siguiente:


```
b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\x00\xff\xe1\x00\xceExif\x00\x00II*\x00\x08\x00\x00\x00\x08\x00\x0e\x01\x02\x00\x01\x00\x00\x00\x00\x
0\x00\x00\x12\x01\x03\x00\x01\x00\x00\x00\x01\x00\x00\x00\x1a\x01\x05\x00\x01\x00\x00\x00n\x00\x00\x00\x1b\x01\x05\x00\x01\x00\x00\x00v\x00\x00\x00(\x01\x03\x00\
01\x00\x00\x00\x02\x00\x00\x001\x01\x02\x00\r\x00\x00\x00~\x00\x00\x002\x01\x02\x00\x14\x00\x00\x00\x8c\x00\x00\x00i\x87\x04\x00\x01\x00\x00\x00\xa0\x00\x00\x00\
00\x00\x00\x00,\x01\x00\x00\x01\x00\x00\x00,\x01\x00\x00\x01\x00\x00\x00GIMP 2.10.34\x00\x002023:11:16 12:41:37\x00\x02\x00\x86\x92\x07\x00\x08\x00\x00\x00\xbe\x
0\x00\x00\x01\xa0\x03\x00\x01\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff\xe1\x0c\xcfhttp://ns.adobe.com/xap/1.0/\x00<?xpacke
 begin="\xef\xbb\xbf" id="W5M0MpCehiHzreSzNTczkc9d"?> <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 4.4.0-Exiv2"> <rdf:RDF xmlns:rdf="http://www.w3.org/1
99/02/22-rdf-syntax-ns#"> <rdf:Description rdf:about="" xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/" xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEven
#" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:GIMP="http://www.gimp.org/xmp/" xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmpMM:DocumentID="gimp:docid:gimp:1
06d884-fd4f-48f5-95ec-99467e39ef0e" xmpMM:InstanceID="xmp.iid:5cc28b8b-046c-4e7d-b67a-7e1660a51ea1" xmpMM:OriginalDocumentID="xmp.did:45ee682a-63b0-4eec-872d-6ae
e7136dc0" dc:Format="image/jpeg" GIMP:API="2.0" GIMP:Platform="Linux" GIMP:TimeStamp="17001'
```


Nota adicional: Me gustaría mencionar, aclarar y agradecer el apoyo que me brindó el trabajo realizado por Muñoz Tamés María y Tafolla Esteban, ya que no me quedaba del todo claro la parte de lectura y escritura de enteros y cadenas. Con base en su implementación, pude guiarme en ese punto y de ahí tomar vuelo y programar de forma adecuada todas las funciones que aquí planteo.